BIONETS

WP 2.2 – PARADIGM APPLICATIONS AND MAPPING

D2.2.2 Framework for Distributed On-line Evolution of Protocols and Services, 2nd Edition

Reference:	BIONETS/CN/wp2.2/v2.1
Category:	Deliverable
Editor:	Lidia Yamamoto (UBASEL)
Authors:	Daniele Miorandi (CN), Paolo Dini (LSE), Eitan Altman and Hisao Kameda (INRIA)
Verification:	David Linner (TUB); 2nd Edition: Fran coise Baude (INRIA)
Date:	June 14, 2007
Status:	Final
Availability:	Public

EXECUTIVE SUMMARY

The BIONETS project aims at offering bio-inspired approaches to autonomic services in pervasive environments. Evolution is an important property of autonomic systems: without evolution, the system cannot fully cater for its own operation, therefore is not fully self-managed, thus not fully autonomic.

Evolution in an autonomic communication system should happen in an on-line and distributed way. It is an open-ended, long-term process in which the system is continuously undergoing self-optimisation, i.e., transforming itself to comply to significant changes in requirements or in the environment. Evolution goes beyond short-time reactive adaptations that are pre-programmed in the system and for which well-known techniques can be employed. This is a challenging research topic: currently little has been done in evolving complex interacting systems during their own operation.

In this deliverable we report our first insights in this domain, as an outcome of Task 2.2.2 in WP2.2. We present a generic framework for the evolution of protocols and services which will be then specialised for the case of high-level services in WP3.2, and of network protocols in WP1.2.

Evolution in our context is the result of a combined process involving transformation operations, fitness evaluation, selection with competitive pressure, and communication. Contrary to traditional evolutionary computing, communication is at the centre of the evolution process: it is used to exchange information for transformation purposes, to spread good service variants, and for fitness evaluation feedback.

We study evolution from two perspectives: micro-evolution and macro-evolution. Micro-evolution looks at local small-scale operations, while the macro-scale studies the global impact of the micro-scale actions. These two views complement each other and thus must be integrated. For that we adopt an approach that combines genetics and chemical computing. At the micro-scale, we are working on a chemical programming language in which program transformations can be expressed in a natural way. Systems biology, evolutionary games and stochastic processes provide tools for analysis of the system at a macro-scale. As a first attempt to bind the two scales together, a graph model for evolutionary interactions is proposed, which takes into account the specific communication characteristics of the BIONETS network.

The next steps are now to apply these insights to services in BIONETS WP3.2 and protocols in WP1.2, to evaluate the performance of evolutionary protocols in WP1.3, and to elaborate on adaptive and evolutionary security within WP4. Some of these activities are already in progress, for instance, the integration of evolution within the service lifecycle: BIONETS services are being designed such that they are constantly being evaluated and selected. This creates a competitive pressure that provides an incentive for the service to perform self-optimising transformations that may ultimately lead to truly autonomic services.

DOCUMENT HISTORY

Version History

Version	Status	Date	Author(s)
0.1	Draft	7.07.2006	Lidia Yamamoto (UBASEL)
0.2	Draft	10.07.2006	Daniele Miorandi (CN), Lidia Yamamoto (UBASEL)
0.2b	Draft	7.09.2006	Lidia Yamamoto (UBASEL)
0.3	Draft	13.09.2006	Daniele Miorandi (CN)
0.4	Draft	22.09.2006	Eitan Altman and Hisao Kameda (INRIA)
0.5	Draft	6.10.2006	Lidia Yamamoto (UBASEL)
0.6	Draft	26.10.2006	Daniele Miorandi (CN), Lidia Yamamoto (UBASEL)
0.7	Draft	31.10.2006	Daniele Miorandi (CN)
0.8	Draft	02.11.2006	Lidia Yamamoto (UBASEL)
0.9	Draft	03.11.2006	Lidia Yamamoto (UBASEL)
1.0	Final 1st Ed.	07.11.2006	Paolo Dini (LSE), Lidia Yamamoto (UBASEL)
1.1	Submitted	09.11.2006	David Linner (TUB), Daniele Miorandi (CN)
2.0	New Revision	07.06.2007	Lidia Yamamoto (UBASEL)
2.1	Final 2nd Ed.	13.06.2007	Daniele Miorandi (CN), Lidia Yamamoto (UBASEL)

Summary of Changes

Version	Section(s)	Synopsis of Change
0.1	all	TOC proposal and initial paragraphs
0.2	most	comments by Daniele Miorandi taken into account
0.3	2, 4.1, 5.2	Draft added
0.4	3.2, 5.4	game contrib. (INRIA) added
0.5	most	big restructuring: now based on micro to macro levels
0.6	4, 8, 9	updates on bio, evo, chem and stochastic models; conclusions
0.7	1,4	alignment with the new structure
0.8	5.2, 5.3	finished these sections
0.9	Exec. Summ., 7	finished sect. 7, added Exec. Summ., final touch
1.0	5.2.5, 6.1, 7.1	new LSE frag parser contrib, review comments by David Linner
1.1	front page, 7.2	final review
2.0	8	new revision after 1st project review
2.1	8	evo vs. adapt, config/compos/funct evo

SUMMARY

Contents

1 Introduction

The BIONETS project is set up in the context of pervasive environments in which billions of devices exchange information in a decentralised way. The scale and complexity of such environments is such that they cannot rely on manual operation and management. An *autonomic* or *self-managing* approach is needed. BIONETS investigates bio-inspired solutions to autonomics in such context.

We argue that *evolution* is an important property of autonomic systems: if the system is not able to evolve to match new needs and situations, then someone must come and update or replace it when it becomes unsuitable, therefore the system is not fully autonomic, as it cannot fully cater for its own operation.

BIONETS investigates solutions based on genetic transformations and natural selection, in order to devise methods to make services evolve in pervasive environments. Evolution should happen in an unsupervised way, triggered only by the need of the service to "survive" in the environment by achieving a high fitness, implying high user satisfaction marks.

This is a challenging research topic, as currently little has been done in evolving complex interacting systems during their own operation, and in a totally decentralised manner.

In this deliverable we report our first insights in this domain, as an outcome of Task 2.2.2 in WP2.2. We put together a generic framework for the evolution of protocols and services which will be then specialised for the case of high-level services in WP3.2, and of network protocols in WP1.2. Although the evolution of *services* is the focus of BIONETS, evolving network *protocols* is also important, since the underlying protocols in an autonomic system must also be able to evolve to match the needs of the evolving services.

The deliverable is organised as follows: We start with a description of the general context of BIONETS in Section 2, in order to motivate the need for automated evolution. Section 3 then states the problem we are trying to solve, our assumptions to simplify it, the requirements for the proposed solutions, together with some basic definitions. Section 4 breaks the problem into two levels at which basic building blocks for designing self-evolving services need to be defined and characterised. Section 5 presents an overview of the selected building blocks at the micro level, and our insights on potential ways to use them to obtain solutions. The same analysis is carried out for evolution at the macro level, and reported in Section 6. Section 7 presents a generic evolution framework using some of these building blocks. This framework is a placeholder for more specific instantiations of services and protocols that could then be studied in WP3.2 and WP1.2, therefore constitutes a relevant source of input for these workpackages.

UPDATE: 2nd Edition, June 2007: ***A new section was added upon request from our reviewers during the First Project Review in Zurich, March 2007: Section 8 presents a research roadmap for service and protocol evolution in the BIONETS context using the framework described in Section 7.***

2 Pervasive Environments

The target scenarios are pervasive computing/communication environments [126], characterised by a huge number of embedded devices with sensing/identifying capabilities, able to provide the necessary support to context-aware services. These scenarios are characterised by scale and complexity figures which challenge the current approaches to networking and service provisioning/management.

First, in terms of networking it is practically not feasible to build a *fully connected* system. This comes, on one hand, from the unpredictability of channel fluctuations and device mobility patterns, which lead to the necessity of designing a robust system, able to work without taking connectivity for granted. On the other hand, this comes as a natural *design choice*, based on some theoretical work which showed that (i) fully connected networks scale poorly [56] (ii) giving up the connectivity constraint a scalable network model.

In particular, Gupta and Kumar showed in their seminal work that, given a wireless network consisting of n devices, half of them acting as sources and the other ones as destinations, uniformly distributed on a plane, the per-connection throughput $\lambda(n)$ scales $\Theta\left(\frac{1}{\sqrt{n \log n}}\right)$ with probability approaching 1 as $n \to +\infty$. This results was obtained considering the optimal scaling law for the transmission power of devices (considered fixed for all nodes). Taking power control into consideration does not help much, leading to a scaling of the form $\lambda(n) = \Theta\left(\frac{1}{\sqrt{n}}\right)$ [128]. In both cases, we see that $\lambda(n) \to 0$ as $n \to +\infty$. This result arose a fundamental issue in the networking community, since it claims that wireless networks cannot scale. This tackles the basic idea underlying all research on pervasive computing, i.e., that myriads of devices could be connected via wireless links. A more careful analysis of the results in [56] shows where the issue comes from. Indeed, the limits on the throughput comes from the necessity of keeping a rather large transmission power (such that the communication range can scale as $\sqrt{\frac{\log n + c(n)}{n}}$, $c(n)$ diverging as n grows) in order to keep the network connected [98]. Such transmission power limits the spatial reuse of the network, leading to the pessimistic results in [56]. A way to overcome such problem was first proposed by Grossglauser and Tse [55], which showed how, exploiting nodes mobility and employing a two-hop relaying strategy, a throughput of the order $\Theta(1)$ could be achieved. This is obtained giving up the connectivity requirement and exploiting the devices mobility to "spread" messages in the network. The picture was then closed by Franceschetti et al. [45], who showed that even in a static network, allowing a small number of devices to be disconnected, a scalable transport network system can be build.

The BIONETS system is therefore designed to operate in the absence of a fully connected network. A snapshot of the network would look like an archipelago, where a myriad of islands (each composed by a limited number of connected nodes) will exist. The mobility of (a subset of) the nodes will be used to ensure system-wide spreading of information by means of a suitable opportunistic forwarding scheme [35, 65]. These techniques are based on store-copy-and-forward operations, where a node which gets one message not destined to it stores it and releases a copy of the message to (a subset of) the nodes it encounters, giving rise to a sort of epidemic spreading. In such system, there is a clear tradeoff between the delay that can be tolerated by an application

and the level of "connectivity" seen by the application itself (in that over a longer time interval a large number of meetings among nodes takes place). A more detailed picture of the BIONETS underlying network architecture can be found in [30], and in BIONETS deliverables D1.1.1 [89] and D1.2.1 [97].

Second, we need to face the problems related to service provisioning over such large-scale disconnected networks. Conventional centralised (or infrastructure-based, e.g., Web Services) solutions cannot work, since they do not fit well the disconnected architectural assumption of BIONETS. Moreover, aside from the provision of individual services, system-wide control of the behaviour of the running services would still be needed, a task of unsurmountable complexity. One possible approach to cope with such issues is to resort to *autonomicity* as the paradigm for designing services able to self-organise into a purposeful system. While the self-$*$ properties of autonomic systems [70] would provide the necessary features for obtaining an unsupervised system able to achieve a given desired behaviour, there is no clear consensus in the research community on how to actually design systems able to show these properties. One of the most promising research lines is to draw inspiration from *biology* to introduce novel design guidelines for systems able to show an autonomic behaviour [88]. The reasoning behind such choice is that nature has been successful in dealing with scale and complexity issues, leading to the arising of ecosystems able to self-sustain in the absence of a central control. In particular, *evolution*, understood as the ability to develop new functionalities, represents a key requirement for autonomic services, in order to let them be able to adapt to new, unpredictable, operating situations.

The BIONETS project is therefore seeking ways of introducing services able to evolve without requiring manual intervention, while at the same time being resilient to network failures and disconnections.

3 Problem Statement and Roadmap

3.1 Definitions

First of all, we must distinguish between *adaptation* and *evolution*. In nature, living organisms are able to *adapt* to their environment to some extent, e.g. to escape predators or to search for food. However, if conditions change drastically, the adaptation capabilities reach their limits, for example, a fish cannot survive outside water. In the long term, however, primitive fishes have *evolved* to different organisms able to walk on earth and breath air, thus evolving new functionalities not present before.

Coming back to software systems, we define *adaptation* as the ability of a system to tune its behaviour to the current environment in a short time scale, following a hard-wired closed-loop algorithm. *Evolution* is a longer-term form of adaptation that leads to new functionality not previously engineered into the system. Evolution implies *generations*, with individuals changing in discrete steps from one generation to the next. Individuals from different generations may have different sets of functionalities.

3.2 Problem Statement

Many adaptive algorithms and protocols exist today. One of the most famous examples is TCP congestion control [64], in which a source adapts its sending rate to fit the bottleneck bandwidth in the network. There is already a considerable body of research in adaptive schemes, with solid results, so the problem is relatively well understood. The same is not true for evolution. We still do not know exactly how to build software systems that are able to truly evolve by themselves once deployed. Most of the evolution tasks are handled by humans, in the form of software updates that bring new features or fix bugs. Automated evolution would ultimately lead to systems that are able to reprogram themselves automatically in response to new user requirements or radical environment changes. The benefits would be significant, with reduced reaction time to needed updates, leading to increased reliability and ease of use. However this is a challenging research problem with no generic short-term solution in sight.

In BIONETS we would like to obtain services that evolve by themselves (at least to some extent) to match new needs, and that at the same time cause the underlying network protocols to evolve to match the new services. Better evolved services should spread further in the network, reaching regions where they are needed, while bad or obsolete services should naturally die out. All this should happen during the operation of the services, without disrupting it. We are aware that the problem is complex and challenging. Therefore we face it by making some simplifying assumptions and then adopting a step-by-step, incremental approach.

3.3 Assumptions

We start with working service and protocol implementations. Therefore we do not attempt at solving the *program synthesis* problem (generating programs automatically [80, 103]), but focus on the *on-line evolution* problem which consists in *improving* programs during their operation, by

spawning *new generations* of programs which are better than before, without relying on off-line reengineering and reimplementation.

We focus on *services* for pervasive environments as a first priority, and secondly on the underlying network *protocols* supporting these services. Therefore we do not attempt to solve the on-line evolution problem for generic software, but only for a specific category of software: *communication software*, communication being there either to support user communication through higher-level services or lower-level network protocols.

3.4 Roadmap

We break the problem of on-line service and protocol evolution into a number of steps of increasing impact and complexity. The steps are shown in Figure 1 and can be described as follows:

(i) *Self-optimisation of parameters at run-time*: In this first step, the service implementation does not change, but the system tries to find the optimum combination of parameter values that is best suited to a given environment. It is well-known that tuning parameters in protocols is a difficult problem: the number of parameters may be large, and it is often difficult to find rules to always obtain the best operating points for all situations. We envisage a solution based on an asynchronous distributed genetic algorithm approach, as described in Section 7. The set of parameters associated to a given service represents the genome that is exchanged in an opportunistic fashion when devices meet, giving rise to a new generation of services. The convergence properties of such mechanism have been preliminarily analysed in [32] and are summarised in Section 6.2.

(ii) *Automatic composition of service/protocol building blocks*: In this second step, the system seeks the optimal combination of existing building blocks to obtain the desired service or protocol. As with the first step, the implementation inside each block does not change. However, now the overall composition of the service/protocol changes, which can be seen as an implementation change at scale of the complete service/protocol. This task can build on our preliminary genetic programming experiments [130] and on results from the DBE project [42]. The experiments in [130] are summarised in Section 5.3.1, and have revealed that a form of genetic operator used in nature, namely crossover by homologous recombination, is a useful metaphor for recombining building blocks in such a way that the chance of obtaining a viable individual (i.e. working program) is high.

(iii) *Automatic generation of inter-component glue logic*: This step is complementary to step (ii) above. While step (ii) assumes that the service/protocol interfaces do not change, step (iii) allows for the interfaces to evolve as well. This requires some glue logic to be automatically generated, in order to neutralise potential syntactic incompatibilities between components. Some related work on evolving interfaces for compatibility can be found in [109]. However the process is not yet fully automated. In BIONETS, an approach based on genetic programming could be used to generate the small code fragments necessary to interconnect components. This is still to be investigated, also in connection with the DBE project [42].

(iv) *Automated code generation*: This is the most ambitious step, in which full building blocks could be generated from scratch. This is by far the most ambitious and risky step, and the solutions that BIONETS might provide here will certainly be only limited. WP2.2 Task 2.2.3 (starting M13) will deal with this topic, using an approach inspired by gene expression in cells, applied to the automatic generation of code from models: the DNA represents the high-level semantics of a component, that is manipulated by genetic operators, and gene expression-like mechanisms translate these functional specifications into running code. Our preliminary experiments [130] indicated that a promising research direction might be to combine genetic programming with deterministic and/or formal techniques for generating code automatically, so that new code could be generated in response to a change in specification. Moreover missing or damaged parts could be automatically regenerated from the same specifications, analogous to the way genes are expressed when needed. This would add a self-healing dimension to the system. Furthermore, chemical models as described in Sections 5.2 and 5.3 can on one hand provide programming languages that are robust to transformations, and on the other hand the tools to analyse and steer the evolution process in order to make sure it converges to the desired operating point. This will be further discussed in Section 5.2.

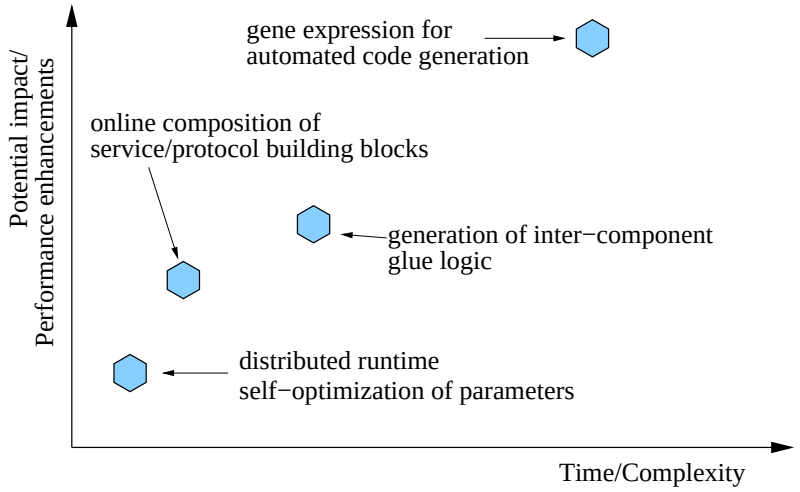


Figure 1: Roadmap for service/protocol evolution in BIONETS and beyond.

A preliminary version of this sequence of steps appeared in the BIONETS project proposal, and was later refined and published in [88]. We now have a better understanding of this breakdown structure, which is reflected in the new description above. Probably not all the aspects of each step will be covered within the project, but at least some stepstones and foundation work will be laid, and evolution experiments will be performed.

4 Building Blocks for Self-Evolving Communication Systems

When trying to identify the building blocks for designing and implementing self-evolving communication systems, we can distinguish between two "levels" at which evolution may take place. The first one is the micro-level, which refers to evolution taking place at the individual service/program level, while the second one refers to a system-wide scale. Evolution at the macro level comes as a consequence of evolution at the micro level, coupled with interactions among micro-level components.

As a first step towards identifying useful building blocks, we thoroughly reviewed several potential approaches in the related literature, and explored in more depth those which appeared more promising.

In a second phase we tried to extract the essence of on-line evolution by bridging together biology and computer science: evolutionary and chemical computing lie at this intersection, and the advantage of combining them together is to have a system that can easily manipulate code and data in a seamless way, and in which code transformations are a rule rather than an exception.

In next two sections (Sections 5 and 6) we synthesise the outcome of these initial exploration and selection phases for the micro and macro levels, respectively. We review existing approaches from literature as well as from our own previous research work, and evaluate potential elements of a solution to the on-line evolution problem. We start by describing micro- and macro-evolution in more detail in Section 4.1 below.

4.1 Evolution at Microscopic and Macroscopic Scales

In a pervasive environment, distributed on-line evolution can be viewed from a *macroscopic* or *microscopic* perspective. At a microscopic level we see individual programs implementing a given service or protocol. A macroscopic perspective offers a system-wide view, in which different services coexist and interact (cooperating or competing) as in a digital ecosystem [42]. The two scales have different dynamics and require different tools to be modelled and analysed. At the *micro* scale, we deal with program transformations, from self-tuning of some running parameters up to self-generation of code. At the *macro* scale we observe how successfully services replicate and propagate throughout the network, how competing services share resources, whether and how cooperation is built in some regions, etc.

Both perspectives are needed to engineer a system that can be then left to evolve on its own. Since we are dealing with a large-scale complex system, where a large number of mutually interacting dynamical entities coexist, the study of the micro level is not sufficient for understanding/predicting the behaviour on a system-wide scale. On the other hand, the study of the macroscopic behaviour alone is not sufficient to derive the necessary program transformations that will lead to optimum performance at both macro and micro levels. Therefore both perspectives must be integrated in our evolution framework, as will be discussed in Section 7.

We can therefore divide our evolution research into two parts:

- *Micro-evolution*: The study of micro-behaviours and the necessary program transformations to make individual programs evolve. The potential building blocks selected here will be described in Section 5.
- *Macro-evolution*: The study of the global system behaviour given an ecology of interacting programs with different strategies. It will be discussed in Section 6.

The sections that follow will therefore follow this structure.

5 Micro-Evolution Building Blocks

Given that we are dealing with *on-line* or *runtime* evolution, it is essential that the system is able to produce new generations of programs that are superior or at least as good as their predecessors. At the micro-level we investigate techniques to manipulate and transform the implementation of protocols and services at runtime, in order to provide an *open ended* evolutionary framework for a continuous evolution of the system in response to changes in the environment or in user requirements.

We start by studying *genetic algorithms* (GAs) and *genetic programming* (GP) in Section 5.1, since these are the most obvious bio-inspired algorithms for evolving programs and their parameters, and for which we have already done some preliminary work (see Section 5.3.1 for a summary). GAs are limited to parameter evolution, thus can cover essentially phase (i) of our framework roadmap (see Section 3.4). In order to reach phases (ii), (iii) and (iv), we need a technique that is able to perform actual code transformations, such as GP. However, we see classical linear or tree-based GP as not suitable for on-line evolution of programs, since the programs generated by crossover or mutation may have a low rate of success [76]. We foresee that a *chemical language* lends itself more easily to transformations by genetic operators, and the resulting programs can be more robust to potentially defective execution paths [121]. We have therefore paid considerable attention to *chemical computing* as presented in Section 5.2. Some chemical computing approaches such as Artificial Chemistries [44] have the additional advantage of lending themselves to macro-level studies by employing techniques similar to those used in systems biology. This provides useful tools to understand the macro-behaviours stemming from micro-scale evolution algorithms.

This section also explains and justifies our on-line evolution approach inspired by genetics and bio-chemical reaction pathways in cells, including gene expression and (auto-)catalytic networks, which led to the framework explained in Section 7.

5.1 Evolutionary Computing

Evolutionary Computing [47], encompasses a range of problem-solving techniques collectively called *Evolutionary Algorithms (EAs)* which are inspired by biological evolution principles such as genetics and natural selection. These are algorithms for searching optimum solutions within a given search space of all possible solutions. EAs employ ideas from genetics and natural selection to perform a *beam search* that restricts the search space to a "beam" area of retained promising solutions. They are typically applied when the search space is so vast that conventional optimisation techniques cannot be efficiently applied.

The main evolutionary algorithms are: *Genetic Algorithms (GAs)*, *Genetic Programming (GP)*, *Evolutionary Programming (EP)* and *Evolution Strategy (ES)*. They all model candidate solutions as a population of individuals with a genotype that is transformed and evaluated against a given fitness criterion until an optimum solution is found. The difference among them lies in the way candidate solutions are represented, and on the search operators applied to obtain new solutions.

In a Genetic Algorithm (GA) [60] candidate solutions are represented as a population of individuals whose genotype is a fixed-length binary string. The goal of a GA is to find the optimum value

of such binary string that optimises a given fitness criterion. An initial population of candidate strings is generated and evaluated against the fitness criterion. The best strings (highest fitness) are then mutated and recombined (crossover) to produce new strings that are then reevaluated, and so on, until an optimum solution is found, or until some stop criterion is satisfied (for example, distance to the optimum closer than a threshold).

Genetic Programming (GP) [18, 73, 75] applies the GA idea to evolve computer programs automatically. GP can be considered as a sub-field of *machine learning* in the sense that it seeks to obtain populations of programs that improve automatically [18], so can somehow "learn" good behaviours. A GP algorithm is essentially the same as a GA, but instead of simple fixed-length binary strings, the candidate solutions are variable-length computer programs, and their genotype is the representation of the program implementation itself. GP typically evolves programs encoded in a linear (similar to assembly language) or tree language (similar to functional languages such as Lisp), but other representations are also possible, such as general graphs [100], finite state machines [7, 110, 111], neural networks [93], and more recently, chemical programs [82, 130].

Programs synthesised by genetic programming are known to have even generated patented or patentable inventions [74]. However, GP has been mostly applied to obtain solutions in an off-line manner: once deployed, the programs evolved by GP in general do not continue to evolve during their operation. Comparatively little has been achieved in the evolution of running systems, but some significant results can be found in domains such as evolvable hardware [112] and robotics [6, 93, 117].

Evolutionary Programming (EP) [51] is similar to GA but focuses on the evolution of finite state machines and does not employ crossover to generate new solutions. Instead, it concentrates on mutation as the main search operator. *Evolutionary Strategy (ES)* [25, 106] is similar to EP in the sense that mutation is the main operator, but focuses on evolving solutions represented as real vectors. We consider EP and ES as specific variations of the more generic GA and GP forms, so EP and ES will not be further discussed in this document.

We now look at GAs and GP in more detail.

The difference between GA and GP lies in the representation of the evolved solutions, and sometimes this distinction can be blurred: GA bit strings can represent parameters of a protocol or service; or the identification of links and nodes in a graph or state machine [7, 100, 110, 111]; or the encoding of which building block is present or absent in a solution, in order to evolve combinations of building blocks [42]. So the approach of evolving protocols as finite state machines as in [110, 111] can be seen as a form of GP, but solution candidates were represented as bit strings, so it can be regarded as a GA too. Similarly, the approach in [130] for evolving protocols by combination of their building blocks can be regarded as GP since the representation used was a variable-length chemical program. However, recombining previously existing modules can also be achieved by encoding each possible module as a bit in a string, and thus applying GA. Similarly, in robotics it is very common to produce robot controllers by evolving the weights of neural networks [93]: since by evolving these weights one is producing new behaviour in a robot, this can be regarded as a form of GP, however the representation of the candidate solutions can well be achieved with a simple GA bit string. Rather than trying to classify an approach into GP or GA, it is more useful

to look at the complexity and flexibility of the representations and their suitability for problem at hand. A continuous space of evolutionary computing techniques is more accurate, in which solutions have a degree of adaptability given by the portion of the program that is represented as an evolvable genotype.

GAs and GP follow essentially the same steps:

1 **Representing candidate solutions.** Candidate solutions to a proposed problem are modelled as a population of individuals. Each individual (candidate solution) is encoded using a genetic (genome-like) representation or genotype. The genotype usually takes the form of a bit stream for GAs and a program tree for GPs.

2 **Fitness evaluation.** Each generated candidate solution is rated by means of a *fitness* function that evaluates the suitability of the solution to solve the proposed problem.

3 **Selection.** A *selection* mechanism operates on the result of the fitness evaluation to select a subset of solutions to "survive" to the next generation.

4 **Generating new candidate solutions.** One or more *genetic operators* (mutation and crossover being the most common) modify the genome to generate new candidate solutions. A *mutation* is a modification of a small portion of the genotype chosen at random. *Crossover* consists in randomly selecting genotype segments in two individuals and swapping these segments between them. The result are two new individuals which are inserted in the population.

5 **Iterative search and optimisation.** The generation of solutions, evaluation and selection process continues in an iterative way until a solution with maximum fitness is found, or until the fitness of the generated solutions remains stable or shows little improvement over generations. The solution with maximum fitness is then taken as the output of the algorithm.

GAs and GP are usually performed as off-line tasks in a centralised manner. This does not fit the nature of BIONETS services, where the evolution process is fully distributed and needs to be performed in real time.

Extensions of GAs and GP to a distributed context include Distributed Genetic Algorithms (DGAs) [24] and Parallel Distributed Genetic Programming (PDGP) [100]. DGA is essentially a synchronous extension of GA to support parallel computing, and PDGP is a type of DGA that operates on graphs. These systems are not meant for on-line operation.

An example of a hybrid on-line/off-line GAs is the Environment Identifying Genetic Algorithm (EIGA, [90]), in which an on-line module takes care of adaptation to the real-world environment using a neural network, while the off-line part performs more intensive parallel search on a population of individuals, based on feedback from the on-line module. Superior individuals found at the off-line module are sent to the on-line module at regular intervals. This approach is promising since only the best solutions found off-line are executed in the real world, protecting the system from most unsuitable variants. However, in a pervasive environment such as the one considered for BIONETS, it might be difficult to find a sufficiently powerful U-Node able to execute the intensive

off-line simulations required to evaluate the fitness of a whole population of candidate solutions. A grid-like DGA would be needed, so that candidates could be dispatched to different machines for off-line (simulated) evaluation. The resulting fitness could then be recollected for evaluation and selection of best candidates. Although in principle feasible, we feel that such a solution could make the system unnecessarily heavy by requiring a grid middleware and a lot of coordination, and as a result it could discourage users from adopting it.

The position paper [46] argues that evolutionary computing is a promising technique for achieving the much wanted self-managing properties of autonomic systems, and present an evolutionary approach to runtime self-optimisation in a distributed system. The position is illustrated with a web service example, in which several web servers offer the same service to a large number of visitors. Each web server behaves according to a parameter vector that determines a session handling strategy. The objective is to maximise the quality of service experienced by the visitors, which is a function of the delay to obtain the service and the degree of request satisfaction. An evolutionary approach is presented in which the population to be evolved is the set of parameter vectors, and servers using a given parameter vector are evaluated on-line during the web session. The paper points out the asynchronous nature of such evolutionary approach, which resembles natural evolution more closely than traditional EAs: at a given point in time in a runtime EA, each individual may be in a different state (some being mutated, some reproducing, some being evaluated), while in traditional EAs, all operations are usually performed in a predefined sequence. The position defended in [46] is totally in line with BIONETS goals for online evolution. However, there seems to be no indication so far of whether the approach has been already implemented or not.

Few results are available that shown on-line evolution of distributed software. The Bio-Networking Architecture Project (Bio-Net, [26]) is an example. It is building a biologically-inspired middleware platform to support adaptive agent-based distributed services [118]. It applies GAs in a decentralised way to evolve the behaviour of agents that provide network services [91, 92]. Since this is one of the research projects that most closely resembles BIONETS (even by the almost perfect coincidence of names) we now describe the evolution approach of Bio-Net in more detail.

In the Bio-Net platform [91, 92], an agent invokes behaviour x (such as replication, reproduction, migration, and death) based on factors that include the agent's internal state (such as age, energy level, user demand for its services, distance from the user) and environmental conditions (available resources, agent population size on the platform, etc.). Each factor v_i has a weight w_{xi} associated to behaviour x. A behaviour is invoked when the weighted sum of factors exceeds a threshold θ_x, i.e. when $\sum v_i \cdot w_{xi} > \theta_x$. Note that this behaviour representation is similar to neurons in artificial neural networks: a neuron "fires" when the weighted sum of input signals exceeds a threshold.

The weights w_{xi} represent the genome of an agent. During reproduction, the genome of two parent agents are recombined by crossover, and then mutated to produce new behavioural weights in their offspring. Although at the present stage mainly simulation results are available, the platform design aims at on-line evolution. Their results show that evolution can improve agent performance, and that agents are able to adapt to their environment.

Although promising and encouraging, the Bio-Net platform does not tackle the problems that we must solve in BIONETS: it is not designed for opportunistic disconnected networks, does not

support the evolution of low-level network services, and essentially restricts itself to phase (i) of our four-phase roadmap (Section 3.4). Moreover, so far Bio-Net does not focus on resilience and fault-tolerance aspects, and does not provide tools for the study of macro-behaviours emerging from evolution. Therefore it is difficult to evaluate how to apply their results to the general case of autonomic systems, in order to show that the system always provides the intended behaviour in spite of failures and other disruptions.

5.2 Chemical Computing

The term *chemical computing* refers to two different areas [28, 43]: The first one is to derive computation models inspired by chemical reactions, which nevertheless run on traditional computers. The second one is the use of real (organic) molecules and (bio-)chemistry knowledge to build computational devices, e.g. in molecular/DNA computing. We focus on the first area, but the second one is also a rich source of inspiration and useful information about modelling artificial chemical systems. We discuss the main chemical computing models and their possible implications for BIONETS.

Chemical computing models have been applied to diverse fields such as image processing applications [21], operating systems, compilers, dynamic software reconfiguration [127], multi-agent systems [62], distributed computing [119], and, more recently, robotics [134], grid computing [13], and autonomic computing [22].

We have selected three main chemical computing models for their relevance to BIONETS: *Gamma*, *Membrane Computing*, and *Artificial Chemistries*.

5.2.1 The Gamma Formalism

Gamma [12, 20] was proposed in 1986 as a programming formalism based on a chemical reaction metaphor. More recently, γ-calculus has been introduced as a formalism that extends the original Gamma model to a higher-order calculus [11]. In [22] this new calculus has been applied to specify Autonomic Computing systems, including a mailing system as an example.

In Gamma [11, 12, 19, 21], computations are modelled as interactions among atomic values that "float" freely in a chemical solution. These values are represented as elements in a *multiset*, an unordered set within which elements may occur more than one. The number of occurrences of a given element within the multiset is called the *multiplicity* of the element. The multiset contains the data to be processed as well as the reaction rules. Reaction rules specify a reaction condition and an action. Computations replace elements satisfying the condition by those specified in the action. A computation terminates when no more chemical reactions can take place. As an example, this is a reaction rule to compute the maximum of a set of numbers:

replace x**,** y **by** x **if** $x \geq y$

When inserted in a multiset containing several numbers, this rule will replace any two numbers x and y that satisfy the condition $x \geq y$ by x. This computation will proceed until only the greatest of all numbers is left. Note that the rule imposes no order in which numbers should be compared. Moreover, if more numbers are injected into the multiset after the computation is finished, the

reaction rule will immediately restart "consuming" the smaller numbers until, again, only the maximum one is left. The rule could also be applied in parallel, by comparing disjoint pairs of numbers. This model therefore enables highly parallel programs to be expressed in a way that is very close to their specification, i.e. without the artificial sequentiality constraints imposed by traditional programming languages. The authors show that this property makes gamma systems particularly suited as a basis for automated program synthesis. This is an interesting property when aiming for on-line evolution, since programs are more robust to the addition, deletion and modification of input parameters and even portions of code on the fly.

5.2.2 Membrane Computing

Membrane Computing [28, 96] is another chemical computing model in which computations (chemical reactions among *objects*) occur inside a cell-like *membrane structure*. Membranes can be recursively nested, and the outside-most membrane is called the *skin membrane*. As in Gamma, objects are represented as elements of a multiset. They can be transformed into other objects and can cross membranes. A membrane can dissolve under the effect of one of its objects; when this happens the content of the membrane is release to its parent membrane. The resulting computing device is called a *P System*. Similar to Gamma, computations inside a P System finish when no more reactions can take place inside its constituent membranes.

Membranes are an important abstraction to enable chemical computing models to scale to large and complex computations. They not only encapsulate complexity from the point of view of rules and name space, but also from the point of view of the (virtual) reaction vessel: if no encapsulation mechanism is available, the implementation of a reaction vessel in a classical von Neumann architecture becomes extremely inefficient: every time step, the execution engine has to check which reaction rules are applicable to which elements of the multiset. For large multisets and complex reaction rules the execution engine would simply not scale.

Moreover, membranes provide a natural mechanism of communication between different execution engines, which is an extremely useful abstraction in BIONETS, due to its inherent networked structure. Communication is modelled as the exchange of chemicals between membranes. If we allow membranes to be situated at different nodes, we can then implement a communication channel between these remote membranes, and ship objects between membranes. As will be discussed in Section 5.3, the notion of communication is inherent within the fraglets chemical programming environment, in which not only passive objects can travel to other nodes but also reaction rules, the "code" for chemical systems. This offers a natural way to dynamically disseminate and deploy newly evolved programs.

5.2.3 Artificial Chemistries

In *Artificial Chemistries* [44], computations are modelled as chemical reaction networks which are represented in the same way as real chemical reaction networks in systems biology: as bi-partite graphs with two types of nodes: substrates and reaction rules. Contrary to Gamma and Membrane Computing which focus on micro-evolution at the level of programming language theory and

execution metaphors, Artificial Chemistries aim at offering methods at the macro-level, in order to understand large-scale reaction systems, either natural or artificial. Chemical organisation theory [81] is used to reduce the complexity of the system from a large network to smaller organisations.

In [57] neural networks are implemented using a reversible reaction mechanism in an artificial chemistry that closely emulates real chemistry. Each neuron consists of eight chemical species coupled by four reversible reactions. The resulting concentrations of chemicals determine the behaviour of the neuron (firing or not), and these are connected to other neurons forming a network. Logical gates are implemented using this system, and a possible extension toward a universal Turing machine is discussed.

Evolving chemical programs has been shown to be possible in artificial chemistries: In [134] a robotic control system is implemented using an artificial chemistry. The structure of the reaction graph evolves by genetic programming. The genetic operations on the graph are a form of mutation (adding or removing nodes and links) and crossover (exchanging portions of two graphs) that obeys mass conservation laws and reaction balance, so that the graphs are kept consistent across transformations. In [82] the authors investigate how chemical evolution appears in a chemical computing system, using organisation theory. This is a step towards understanding global (macro) system behaviour as in systems biology as will be discussed in Section 6 below.

The evolution potential of chemical computing makes it an essential building block for the BIONETS architectural framework for evolutionary services and protocols. This will be discussed in more detail in Section 7.

5.2.4 Fraglets

The *Fraglets* chemical programming language [120] had been proposed prior to BIONETS, as part of our search for feasible ways to achieve automated synthesis and evolution of protocol implementations. The language is based on a chemical model where "molecules" interact with each other or undergo some internal transformation. An implementation of a fraglet interpreter in C is available for download at the Fraglets web site [52].

In this section we provide an introduction to the language, in order to assemble the necessary elements for the discussions that follow, and to motivate the proposed evolution framework.

Fraglets stand for "computation fragments" [120], and are used for representing both data carriers and executable rules (code) in a unified way. A fraglet is a string of symbols $[\, s_1 : s_2 : \ldots : s_n \,]$ representing data and/or protocol logic. It is a fragment of a distributed computation, that may be carried in packets or stored inside a network node. Like molecules, fraglets operate on each other and undergo transformations like splitting up or transmitting themselves to a remote node. The result can be regarded as a distributed chemical reaction network, where the execution of one fraglet rule leads to the creation of the next fraglet to be executed. Code and data are then constantly being transformed and regenerated to perform a given task, typically related to communication protocols.

When fraglets are carried in packets that traverse a network, the successive fraglet symbols can be interpreted as successive header fields in today's regular data packets. Upon arrival, a fraglet

packet is injected into the local fraglet store or context, where an execution environment processes the fraglet "headers" in a similar way as header processing treats packets in traditional networks.

The fraglet processing engine continuously executes tag matching operations on the fraglets in the store, in order to determine the actions that should be applied to them. Fraglet operations, except for the transmission, have the property that they can be carried out in constant time.

The fraglet store is a multiset, as in Gamma (see Section 5.2.1): several instances of the same fraglet may be simultaneously present. This is indicated by a suffix counter value as in $[\,data : item\,]k$ (meaning that k copies of fraglet $[\,data : item\,]$ are stored in this context).

The fraglet instruction set currently contains two types of actions: a transformation of a single fraglet, and a "chemical reaction" between two fraglets. Table 1 (from [130], earlier versions appear in [120, 121]) shows the reaction and transformation rules defined so far.

Table 1: Fraglet reaction and transformation rules

Reaction	*Input*	*Output*	*Semantics*
`match`	$[\,match : s : tail_1\,]$, $[\,s : tail_2\,]$	$[\,tail_1 : tail_2\,]$	concatenates two fraglets with matching tags
`matchp`	$[\,matchp : s : tail_1\,]$, $[\,s : tail_2\,]$	$[\,tail_1 : tail_2\,]$ $[\,matchp : s : tail_1\,]$	persistent match (preserves `matchp` rule)
Transf.			
`dup`	$[\,dup : t : u : tail\,]$	$[\,t : u : u : tail\,]$	duplicates a symbol
`exch`	$[\,exch : t : u : v : tail\,]$	$[\,t : v : u : tail\,]$	swaps two symbols
`split`	$[\,split : t : \ldots : * : tail\,]$	$[\,t : \ldots\,]$, $[\,tail\,]$	breaks fraglet at $*$ position
`send`	${}_A[\,send : B : tail\,]$	${}_B[\,tail\,]$ (unreliably)	sends fraglet from A to B
`wait`	$[\,wait : tail\,]$	$[\,tail\,]$ (after interval)	waits a predefined interval
`nul`	$[\,nul : tail\,]$	$[\,]$	fraglet is removed

The semantics of the reaction and transformation rules are:

- *dup*: Duplicates the symbol at the third position (u); the second field (t) becomes the new fraglet's head symbol.
- *exch*: Swaps the symbols at the 3rd and 4th position (u and v respectively).
- *new*: Creates a new symbol n_{i+1} which is unique in this context.
- *split*: Breaks the fraglet into two parts at the first marker position ($*$).
- *send*: Sends the fraglet unreliably to the destination context specified by the second symbol (B). The subscript prefix ${}_N[\ldots]$ denotes the context where the fraglet is stored.
- *match*: A fraglet starting with a *match* symbol reacts with another fraglet whose head matches the symbol immediately after the *match* symbol. The reaction consumes both headers and produces the concatenation of the tails of both fraglets, starting with the one whose header was the *match* symbol.

- *matchP*: Persistent variant of *match* which keeps a copy of the initial *matchP* rule in the store. It thus acts like a "catalyst", i.e. a reactant that is not consumed in the reaction process.

Simple protocols as well as relatively complex ones have been implemented using this small and apparently limited instruction set. In [120] a simple confirmed delivery protocol (CDP) is shown, which ships fraglets from a source to a destination node. It waits for confirmation of reception before shipping the next fraglet. In the same paper [120] a more complex example is shown, which uses a credit-based flow control mechanism such that several packets can be sent before blocking for an acknowledgement becomes necessary. This protocol supports reordering of messages, but not messages losses. Losses are supported in the Reliable Delivery Protocol (RDP) protocol [130], developed to test the automatic evolution potential of fraglets programs.

In order to illustrate fraglets with concrete examples, we show CDP and RDP in Figures 2 and 3. CDP works as follows: Node A sends an input *data* fraglet to node B; when B receives the data it delivers it to the application and returns an *ack* fraglet to node A. RDP has the ability to retransmit lost packets after a timeout expires. For this purpose, it stores a local copy of the application's payload, sets a waiting timer. When the timer expires, and the corresponding local copy of the information is still stored, the packet is retransmitted. When an acknowledgement is received, the local copy is destroyed. This causes any pending retransmissions scheduled for the item to be cancelled.

Note that CDP and RDP are divided into "modules" marked by an *"m"* followed by the module name. These markers indicate crossover borders for the genetic programming experiments discussed in Section 5.3.1, and are simply ignored during execution.

m send	**m receive**
$_A[matchp : data : send : B : deliver]$	$_B[matchp : ack : send : A : deliver : ack]$

Figure 2: Implementation of a Confirmed Delivery Protocol (CDP) in fraglets

Fig. 2 shows the fraglet code for CDP, both sender and receiver sides. When presented with an application payload of the form $_A[data : payload]$, the first *matchp* rule in the *send* module will be activated, and the resulting reaction will produce a rule $_A[send : B : deliver : payload]$, which will send the fraglet $[deliver : payload]$ to B, where the *deliver* tag will cause *payload* to be delivered to the application. The application will respond by injecting a $_B[ack]$ fraglet, which will react with the *matchp* rule of the *receive* module, causing the *ack* to be delivered to the source application on node A. Note that the *deliver* tag can be implemented as a predefined rule that takes the tail symbol string out of the fraglet environment (towards an external application), or can be caught by a $[matchp : deliver : ...]$ rule as part of a fraglet application.

The RDP implementation is shown in Fig. 3. Note that this implementation presents exactly the same interface with the application as CDP, so that both protocols could be interchanged in a transparent way in the genetic experiments of [130]. A $[data : payload]$ fraglet injected by the application activates the *send* module, producing two fraglets: $[retransmit : payload]$ and $[mack : payload]$. The first one triggers a retransmission loop (*retransmit* module). The second

m send
$[matchp : data : dup : data3]$
$[matchp : data3 : exch : data2 : mack]$
$[matchp : data2 : exch : data1 : *]$
$[matchp : data1 : split : retransmit]$
$[matchp : mack : exch : mack5 : nul]$
$[matchp : mack5 : exch : mack4 : *]$
$[matchp : mack4 : dup : mack3]$
$[matchp : mack3 : exch : mack2 : wait]$
$[matchp : mack2 : exch : mack1 : split]$
$[matchp : mack1 : match : ack : split :$
$deliver : ack : * : match]$

m retransmit
$[matchp : retransmit : dup : t91]$
$[matchp : t91 : exch : t92 : t94]$
$[matchp : t92 : exch : t93 : *]$
$[matchp : t93 : split : transmit]$
$[matchp : t94 : dup : t95]$
$[matchp : t95 : exch : t96 : retransmit]$
$[matchp : t96 : dup : t97]$
$[matchp : t97 : exch : t98 : *]$
$[matchp : t98 : split : wait : match]$
$[matchp : transmit : send : B]$

Figure 3: Implementation of a Reliable Delivery Protocol (RDP) RDP in fraglets (sender side). The sink side is identical to CDP.

one triggers a series of reactions which produce a new rule able to treat an incoming *ack* and cancel any corresponding retransmission.

5.2.5 Software Catalysis

We can establish a parallel between fraglet execution traces and graph representations of real chemical reaction networks as commonly done in systems biology. This has been initially discussed in our previous work [123] and is expanded in this section.

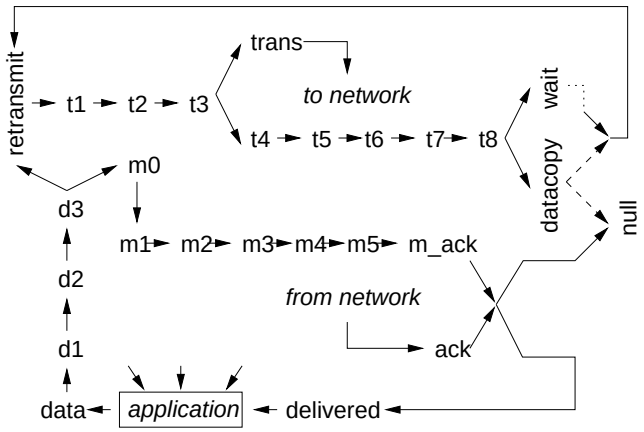


Figure 4: RDP Execution Trace as a "Chemical Reaction Network"

Figure 4 shows a graph where each node contains a token that represents a stage in the chemical execution of the RDP code of Figure 3. Arrows represent transitions that trigger further reactions. During the execution, the original fraglets of Figure 3 react and transform themselves according to the rules of Table 1, resulting in new fraglets not present in the original multiset. For readability purposes, instead of showing the full fraglet (which could be a very long string), each of these intermediate fraglets has been assigned a label on the graph of Figure 4.

The token *data* triggers the main execution cycle, activated by data payloads obtained from the application, as in Figure 3. A series of reactions then proceed until node *d3* splits into an acknowl-

edgement waiting path (starting with token *m0*) and a retransmit cycle (token *retransmit*). The ack waiting path ends up with the production of a rule that waits for the arrival of an acknowledgement, and reacts to it by producing an inhibitor for the retransmission cycle. This inhibitor kills the pending retransmission request (which holds a copy of the data to be retransmitted, the *datacopy* token) by reacting with it and generating a *null* (empty) fraglet as a byproduct. If there is a packet loss, after a timeout the stored copy of the data payload will be retransmitted, and a new wait cycle is initiated.

We should note that most of the transitions indicated in Figure 4 require a persistent fraglet (*matchp*) to trigger the reaction. Indeed, Figure 3 shows that the implementation of RDP is fully based on *matchp* rules which "catalyse" reactions in the system without being consumed in the process. The system is therefore based on a *"catalytic"* process steered by fixed *matchp* rules. This solution presents a major drawback: as most programs written in traditional programming languages, it cannot be easily modified during runtime: a *matchp* rule is persistent, and therefore cannot be easily erased or modified. One could devise a *match* rule that would "eat up" *matchp*'s (a naive one being simply [*match* : *matchp*]) but such a rule would obviously not be able to steer the system towards good transformations that could make it evolve. Replacing all *matchp*'s by *match*es does not work either, since these *match*es are consumed during the reaction without being replaced, breaking the program flow.

Our target is a system that does not depend on fixed fraglets but creates those code fragments that it needs for execution as it goes. Most of the code would then be regenerated at run-time according to various inhibition and activation code fragments that regulate *code expression* (in analogy to biological *gene expression*), according to local constraints and code exchanged with the other nodes in the network. Such a system would be much more adaptive and resilient, as control mechanisms could be incorporated to it in order to monitor and steer the system such that rules would be generated only when needed, and bad rules would be consumed without being regenerated.

A first step in this direction is to rely solely on the *match* rule and its multiplicity counter. A *match* rule is consumed in the reaction, providing a first trace of *mass conservation* in chemical computing, in analogy to real chemistry. Mass conservation has been overlooked in chemical computing since it is not needed to provide the necessary computations. However, we feel that it is an important abstraction to ensure proper *resource management and control*, for example, to make sure that a program will not monopolise the CPU, or allocate too much memory, or overflow the communication channels.

The initial fraglet code could then be regarded as the *"genome"* which is expressed at regular intervals by injecting a given finite *"concentration"* of *match* rules in the execution environment. The concentration of a rule is given by its multiplicity counter. *"Enzymes"* (also encoded as fraglets) would then inspect and direct system execution to promote the expression of certain *"genes"* that are needed at a given moment, in a mechanism inspired by actual gene expression in living beings.

Such a network able to steer and cater for its own software production would then move from a *catalytic* to an *autocatalytic* stage [123], a self-sustaining network in which the production of its

running code fragments is steered by other code fragments in the network. The model for such *autocatalytic software* is that of a highly dynamic system that is constantly rewriting itself, able to support the changes needed for evolution and adaptation, and to protect itself from lost or corrupted code parts by reexpressing valid genes at certain target *concentrations*, from a "code pool" repository. We would then talk in terms of *concentrations of programs, rules or instructions* which would then be controlled in order to achieve a desired equilibrium. In such a dynamic system the equilibrium point is a moving target that may change to reflect new requirements or needs coming from the users.

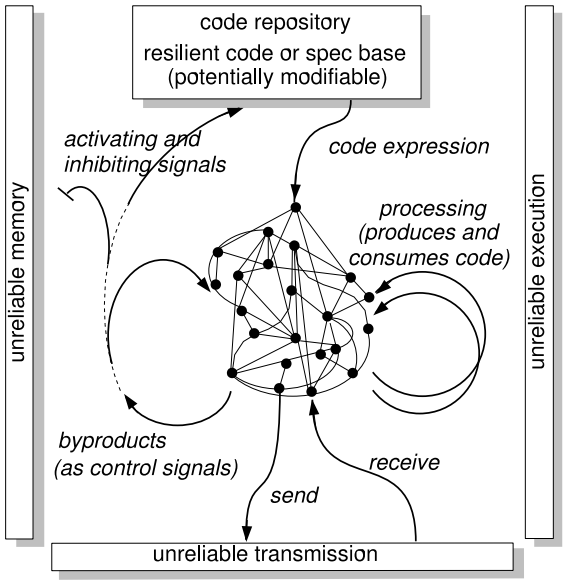


Figure 5: Autocatalytic Software Execution Model

Figure 5 depicts the execution of autocatalytic software as proposed in [123]. Execution is shown for a single node where the system must cope with unreliable storage, execution and transmission (as opposed to traditional network protocols which usually cope well with unreliable transmissions or complete node failures, but assume reliable execution and storage). We have introduced this model in [123] as an extension of previous work [122] on self-modifying code as a tool for resilient software. Byproducts of chemical computing reactions serve as activator or inhibitor signals to control code expression, by blocking, boosting or modifying code consumption-production cycles. These execution byproducts could also affect the initial code base introducing persistent code transformations and thus long-lasting changes aiming at evolution. The execution flow naturally extends itself to a distributed context; the send and receive arrows in 5 represent code being shipped between nodes.

As a first step to test the software catalysis hypothesis, we have developed a fraglet parser and visualiser for the Mathematica tool, which allows us to measure and plot the "concentrations" of given fraglets in time. The parser takes as input the trace messages produced by the fraglet interpreter, which report the reaction and transformation rules that are applied at each step of

execution. The parser then calculates the number of each kind of fraglet at each time step, and detects any new fraglets appearing during the course of the simulation. It then uses this information to plot the measured concentrations.

We have applied the parser to a real cell cycle, that of the p53 transcription factor [36]. The main part of the cycle has been implemented in fraglets, showing at first that fraglets can not only be used to implement network protocols, but also to model real chemical reaction networks. Figure 6 shows an example of parser output, with the concentrations of two proteins highlighted: `p53` and `mdm2`. Note that these concentrations do not yet reflect those achieved on a real biological model, but are only intended as examples to show how the parser works.

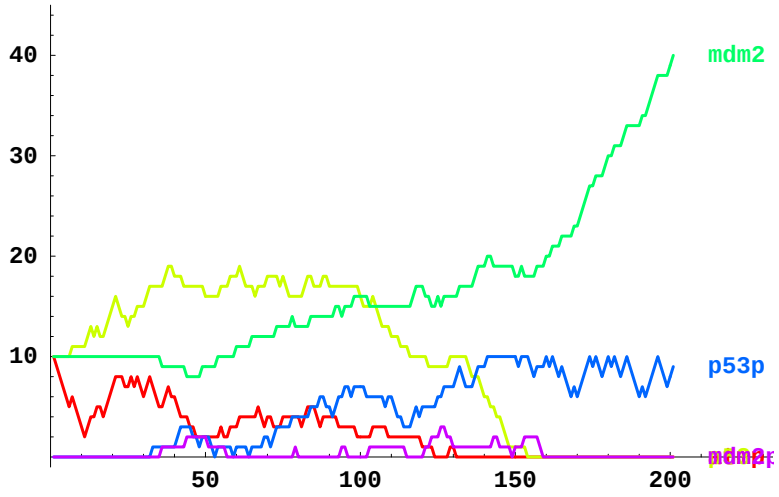


Figure 6: Fraglet parser visualisation of protein concentrations on the p53 regulation cycle. x axis: time (in simulation steps); y axis: concentration (or multiplicity counter, in number of fraglets).

The p53 protein responds to DNA damage signals by controlling the transcription of genes that either enable the damaged DNA to be repaired, or triggers apoptosis (programmed cell death) when no repair is possible. P53 is itself regulated by another protein called *mdm2*. When the P53-mdm2 regulatory cycle malfunctions, a cell that should terminate itself might instead reproduce, which might lead to cancer growth. P53 regulatory malfunction is associated with about half of all cancers.

Because the P53 regulatory cycle is studied intensely by thousands of researchers around the world, there is a huge wealth of data about it. This can be helpful in the construction of a Fraglet-based model that attempts to reproduce the gene expression dynamics of this particular cycle as an example. This will in turn provide us insights on how to apply this to software code expression dynamics.

An idea is to take inspiration from the p53 model to produce fault-tolerant software that in response to a "damage" signal from a component (e.g. too low fitness or too high resource consumption) would trigger a self-healing procedure that would first attempt to fix the problem, and if unsuccessful, would terminate the process and restart a new copy (with a potentially different

version) reloaded from a repository. The difference with classical fault-tolerance measures is that here we would work with concentrations of components instead of binary presence/absence of a given component, and the self-healing cycle itself would be subject to regulation, like the p53 protein is subject to mdm2 regulation.

In Section 7 we will show a higher-level catalytic model in the context of service evolution, in which services can be seen as catalytic substances whose concentration increase or decrease according to how good or "fit" they are in providing the desired functionality. *Autocatalytic sets* (ACSs) are considered in our evolution framework (see Section 7.6), although we have not yet been able to show exactly how autocatalytic processes could be engineered into the system, or alternatively, how they could emerge from the evolutionary process. Related work on evolving artificial regulatory networks [15, 16, 78] and artificial chemistries [82, 134] show promising results in this direction, although still only for passive representations that do not directly manipulate actual program code. Moreover, most of these results either remain at a very abstract level or aim at other application domains (systems biology, robotics), and are therefore difficult to be directly mapped to the concrete development of communication services and protocols in BIONETS.

Research on gene expression mechanisms for code generation is envisaged in BIONETS within Task 2.2.3, starting at M13. The ideas above will then be pursued in depth, in order to move from the current conjecture stage to actual facts. Another planned extension is the conversion of the fraglets system from a pure network-oriented language to a generic one that also supports higher-level services for the purpose of service evolution in WP3.2.

5.3 Evolution of Chemical Programs

As discussed in Section 5.2, Chemical Computing shows many attractive traits for on-line distributed evolution:

- Program transformations (either involving just passive data or also code) are naturally supported since execution proceeds as chemical reactions consuming and producing program elements.
- The parallelism of a multiset reaction vessel naturally supports the addition, deletion and modification of programs in a non-disruptive way: these are just like substituting chemical compounds in the reaction vessel; reactions do not have to stop because of that.
- The problem of discontinuity of the search space in genetic programming (a small change in a program may cause a big leap in fitness) is minimised by allowing program or instruction concentrations to be expressed as multiplicity of elements in a multiset. This is a way of incorporating redundancy in the system, that may cause it to become more resilient to misbehaving programs. This may lead to more stable and predictable transformations in which a small change in a program should lead to a comparably small leap in fitness. This cannot be easily achieved with conventional programming languages in which changing a single line of code in an otherwise perfectly working program may cause it to crash immediately, to loop forever, to destroy resources, etc.

The potential of chemical computing for evolution is indeed already being explored in related work on evolving Artificial Chemistries [82, 134] (see Section 5.2.3), Artificial Regulatory Networks [15, 16, 78], and our own research on evolving protocol implementations with the Fraglets chemical language [120, 130] (see sections 5.2.4 and 5.3.1).

In addition to the attractive traits mentioned above, the fraglets model presents other desirable features for distributed on-line evolution:

- Its syntax is simple and uniform: any string of symbols is a valid fraglet, therefore fraglets can be split at arbitrary places and merged with other fraglets to produce different code.
- The ability to express *code mobility* in a natural way: any fraglet can be regarded as both a set of packet *header tags* that can be processed by a header processing engine, and as a program fragment that is executed at a given node. This facilitates the *dynamic deployment* of newly evolved protocol logic.

We have indeed performed some genetic programming experiments to evolve simple fraglet protocols [130, 131]. We summarise these experiments Section 5.3.1 below, and then discuss the insights gained from them, and the on-going work to extend the fraglets model to support evolution of protocols and services in BIONETS.

5.3.1 Genetic Programming Experiments

We have performed some early experiments using Genetic Programming for the automated evolution of simple network protocols written in the fraglets [120] chemical programming language. The results of these experiments have been reported in [130, 131].

The experiments encode a protocol as a "genotype", which is a concatenation of the fraglets implementing the protocol. The genotype is transformed from one generation to the next via the genetic operators crossover and mutation.

Each protocol is encoded as a fraglet genotype made up of constituent modules or genes. The genotype is the concatenation of all the modules (and their constituent fraglets) that implement the protocol. Each module starts with an *"m"* marker followed by the module name.

Mutation is applied with a low probability, and corresponds to the simple replacement of a fraglet symbol at a random position by another symbol chosen at random from the set of symbols used in the original implementation.

The crossover operator is a simplified implementation of the genetic concept of *homologous recombination*. Homologous recombination states that the exchange of genetic material can only occur between functionally compatible DNA segments, and is only triggered when the two DNA strands are completely aligned. This form of recombination preserves gene functionality, promotes genetic stability, and increases the probability of producing viable offspring. It is implemented in our system by swapping compatible modules in different protocol implementations. Two modules are compatible if they have the same name and interface. They are therefore alternative implementations of the same protocol. Therefore the crossover performed in this way between two viable individuals (i.e. already well-functioning programs) produces other viable individuals.

The experiments were performed in a simulated environment with synchronous genetic programming scripts that manipulated fraglet source code files outside the execution environment. The resulting programs were then injected into the execution environment, to which an external application was coupled. This application evaluated the fitness of the protocols in a centralised way. The resulting fitness was provided as input to the external scripts that then selected higher fitness individuals and discarded low fitness ones. Genetic operators were then applied to produce new individuals that replaced those that had been discarded. Successive generations of protocols were created, executed and evaluated in this way.

Several variants of the CDP and RDP protocols described in Section 5.2.4 have been implemented to make up a reasonably sized initial population for the GP run. Figures 2 and 3 show examples of correct implementations. Other correct variants are also present in the experiments, as well as deliberately inefficient or buggy variants that do not retransmit packets correctly, introduce arbitrary delays, consume more memory, contain useless code segments, pollute the code pool with byproduct debris of reactions, and so on.

Three sets of experiments were performed. In all of them, an initial mixed population as described above is set to evolve by genetic programming, using mutation and homologous crossover, fitness evaluation and selection. We now provide a brief summary of these experiments, See [130] for detailed results.

In the first set of experiments we tested whether the genetic programming system was able to eliminate useless garbage code arbitrarily added to the programs. The experiments showed that the system was typically able to eliminate most of the garbage modules. However if a module happens to propagate to the whole population, it can not easily be eliminated, since homologous crossover results in identical modules being swapped, and mutation is currently too slow.

The second set of experiments aimed at measuring the adaptation capability of a given population of protocols to changes in the environment, in this case the presence of a lossy versus a non-lossy channel. The initial population is a mix of CDP and RDP-derived individuals, some of them correct, some of them buggy or inefficient. The populations are able to quickly adapt, as shown in Figure 7. In this figure, a high-score individual is defined as an individual that has achieved a score equivalent to at least 80% of the best score (optimum) individual. A low-score one scores less than 40% of the optimum. The optimum individuals are considered as the "perfect" implementations shown in Figures 2 (CDP for the non-lossy environment) and 3 (RDP for the lossy environment).

The third experiment evaluates the re-adaptation capacity of a population, namely to adjust to an environment different from the one where it had originally evolved. Two populations are taken from the end of the second experiment: the one which had evolved in a lossy environment is injected into a non-lossy one and vice-versa, and set to continue evolving by genetic programming as before. Here the results clearly show that the populations are *not* able to readapt so easily. All individuals from the population which had evolved in a non-lossy environment simply lost their retransmission modules which were not needed; since crossover cannot bring the module back, they are only left with mutation (of single symbols) which is too slow and random to produce a useful retransmission module in feasible time: the search space is simply too vast. The current genetic

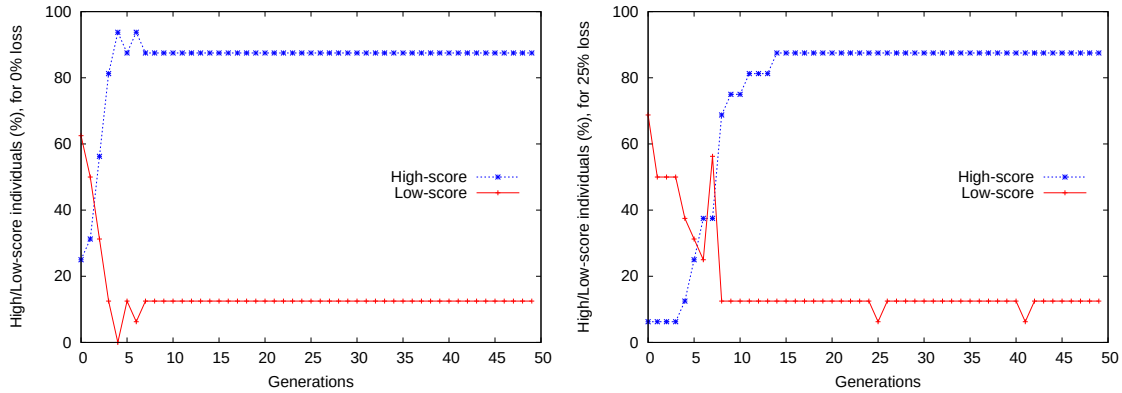


Figure 7: Percentage of high and low score individuals in a population adapting to a non-lossy environment (left) and to lossy one (right)

operators are insufficient to ensure evolution in this case, where genetic variability has been lost due to a previously stable environment. A similar problem may also occur in nature. Nevertheless, injecting a single adapted individual suffices for the missing functionality to spread very quickly through the population, so that it can then readapt.

5.3.2 Lessons Learned

The early experiments reported above showed that evolution of genetic programming of network protocols expressed in fraglets occurs and is feasible as long as a minimum variability of code instances is kept. It remains to be investigated whether the dynamic network islands resulting from disconnected operation and opportunistic communication in BIONETS provide the right balance between isolation and dynamics, so that this minimum genetic variability can be maintained. If not, the genetic inheritance mechanisms have to be refined to try and preserve variability, for instance by introducing "dormant" or "recessive" genes.

The experiments also provided insights on other obstacles that have to be overcome. The first one that decentralised fitness evaluation is a non-trivial issue in a real-world, distributed on-line competitive environment as the one envisaged in BIONETS. Centralised fitness evaluation effectively prevents the emergence of *cheat programs*, i.e. programs that "lie" about their actual behaviour, e.g. by issuing an acknowledgement without transmitting any data. And we have indeed observed that these cheat programs can easily emerge and spread through the population if the fitness evaluation function is not strict enough to catch and eliminate them. We are investigating the combined use of redundancy with trust and reputation mechanisms to to provide a safe and reliable way to evaluate the behaviour of protocols at run-time.

Another fundamental issue is to devise genetic operators beyond homologous crossover and point random mutation. The experiments have pointed out the weaknesses of both operators, namely the inability to create or to innovate. First of all, all mutations observed were harmful. Random, single-symbol mutation is unable to produce useful program varieties in a reasonable time: the search space is far too vast, and no new symbols are generated which are sometimes necessary judging by the hand-made programs. There is a trade-off between the predictability of a

small alphabet (to limit the size of the search space) and the creativity to search for new solutions. Second, the homologous crossover operator is too conservative, it is only able to combine existing modules, so innovation cannot come from there either.

We must somehow obtain new genetic operators able to evolve genuine new code for unforeseen situations, if possible with correctness preserving properties. We are indeed investigating deterministic code generation techniques combined with genetic operations, in order to improve program transformation. Although the CDP and RDP program variants described above have been essentially programmed by hand, a sparkle of deterministic automated program synthesis has been used as well. Indeed, we have implemented a short Perl script that is able to generate small portions of fraglet code automatically, when the input and output is very well defined. Essentially, the script is able to deal with the frequent case of replicating an input symbol from one fraglet into a set of others. The input symbol together with a template for the output fraglets is provided as an input, and the script produces the fraglet program that taken the input symbol, produce the desired fraglets containing the symbol. Although it is still rather primitive, if we consider the generated program as a module, then the script is able to regenerate modules out of their specification (input and desired output). This is a first step towards more sophisticated deterministic methods. However deterministic methods are not able to generate innovation either, since they follow hard-wired steps to produce a provably correct solution. We envisage that a hybrid between deterministic and more advanced genetic operators could be a good compromise.

We also need to elaborate on the inherently asynchronous nature of evolutionary computing in a distributed on-line setting, as pointed out in BIONETS and in related work [46]. In on-line distributed evolution, some individuals might be mutating, while others being evaluated, and so on, without any global management. While this is relatively straightforward in a real-world set-up, we still need a simulation environment where to test and evaluate ideas before deployment. One way to achieve this would be to integrate the genetic programming environment with a network simulator such as ns-2. Work is in progress towards this goal. We intend to integrate these developments with WP1.2 and WP3.2 related to the network architecture and the service architecture, respectively, in order to be able to evolve real protocols and services.

5.3.3 On-Going Work and Ideas

From the lessons learned above, we start by asking ourselves again, which problems reported above come from the programming model used, and which ones are inherent to distributed on-line evolution. First of all, the use of the fraglets language should now be questioned in the light of some BIONETS requirements, for instance to support high-level services. On the other hand, as discussed at the beginning of Section 5.3, chemical languages seem to be the only ones able to provide the necessary flexibility for code manipulation; and other chemical languages are either in pure theoretical stage with no known publicly available implementation, or provide no easy support for implementing distributed systems.

The Fraglets execution model has been originally inspired by the Gamma systems explained in Section 5.2.1. Although some similarities can still be observed, the model radically departs from Gamma in many ways. Like Gamma, it can express parallelism in an elegant and natural way; and

like the higher-order γ-calculus [19, 22], code and data are part of the multiset. Unlike Gamma, however, Fraglets possess a specific string structure which embeds some sequentiality and allow code and data to be easily and transparently shipped to other nodes through communication channels and continue processing at those nodes. This is a consequence of fraglets' original motivation to express communication protocols.

Fraglets' "header processing" feature turns out to be a double-edged sword: it offers a powerful mechanism to express protocols and communication functions in a very compact way, and it offers a simple linear representation that can be easily transformed into a "genotype" form that can be manipulated by genetic programming [130] as will be shown in 5.3.1. On the other hand, the resulting programs are hard to be manipulated by humans (which is a natural consequence of fraglets' original design aimed at the automatic synthesis of protocols), and the resulting "flat" structure poses scalability problems. Although fraglets were designed for automatic programming, no tools to generate code fully automatically are available so far.

Still, we are persuaded that this research path remains promising, due to the strong advantages of chemical computing with respect to conventional programming, and the potential of fraglets as the only known so far *practical* realisation of chemical computing to produce concrete protocol implementations.

Within BIONETS, we are now working to improve the fraglets model by trying to combine its strengths on distributed processing with the strengths of other models: Gamma on the natural, human-dimension programming model, Membrane Computing on the scalable dynamic hierarchical structure, and Artificial Chemistries on the potential to derive macro-level properties from micro-level chemical programs.

Although the gene expression task is only schedule to start at M13, we have already started to do some prospective work in that direction. We are studying the mechanisms of genetics and gene expression in biology, and putting together a service and protocol evolution model based on these mechanisms. The model comprises two parts: the genotype related part concerning the program "genome" and corresponding gene transformation mechanisms (how to improve mutation and crossover, other possible operators including deterministic code generation from specifications), and dissemination (by cloning or reproduction) of selected individuals with their genotype to regions of the network where they are needed; the second part concerns the phenotype, i.e. how to perform "code expression" by injecting segments from the genotype or "genes" into the executing multiset with a given multiplicity or concentration, how to establish fitness criteria to evaluate the resulting service performance, then communicate the fitness value as a feedback to the service, and select the best individuals according to their fitness, while discarding unsuitable ones. Some of these elements are reflected in the framework of Section 7 while others remain to be defined, such as the exact code expression mechanism.

In order to better understand the properties and potential of chemical computing, and get more hands-on experience, on-going effort is also in progress to develop fraglet interpreters in scripting languages and in Java, as well as to integrate an interpreter with a simulation package such as ns-2.

6 Macro-Evolution Building Blocks

Here we consider techniques that look at the system as a whole, in order to understand its global properties. Three main research lines have been identified:

- *Evolutionary Games*: In evolutionary game theory, the evolution of entire populations can be studied by modelling the interactions among individuals that receive a certain amount of "reward" related to their reproduction capability (survival of the fittest). Populations that receive higher reward will naturally grow, while others might disappear. Populations may also be affected by mutations, or invasions by other populations. The evolutionary games formalism is one of the basic mathematical tools designed for predicting population dynamics in this context. Section 6.1 provides a brief overview of evolutionary games, their relevance for BIONETS and the activities in this area that take place in BIONETS WP2.1.
- *Stochastic processes*: The behaviour of large-scale long-term distributed evolutionary processes can be studied using stochastic methods. This can be shown in Section 6.2, where conditions for the convergence of evolutionary processes are determined in the context of an opportunistic network. Section 6.2 contains a summary of research previously published in [32] plus a short report on on-going work.
- *Systems Biology*: The field of Systems Biology [72] makes heavy use of mathematical models to understand real biological systems and the complexity of the interactions between different biological cycles and processes. It builds heavily upon graph theory, control theory, and probability, and includes experimental, theoretical, and modelling techniques. We can learn from the techniques used in systems biology to study our artificial evolving systems in order to understand their global behaviour. Indeed, as discussed in Section 5.2, Artificial Chemistries [44] use graph representations which are very similar to those used in systems biology, therefore similar tools could be used to analyse them. In [81] chemical organisation theory is presented as a theoretical base for studying large-scale artificial chemistry systems, such that full, complex systems can be divided into smaller, manageable organisations. This theory has been applied to study how evolution might emerge spontaneously in chemical organisations [82], therefore this topic is highly relevant for BIONETS. Indeed, some prospective work has started in this direction within the project, and points in the direction of Self-Organised Criticality as a potential mechanism related to evolution. A summary of the insights gained so far is presented in Section 6.3.

6.1 Evolutionary Game Theory

In this section we give a brief overview of evolutionary game theory, and discuss its potential use in BIONETS. Note that evolutionary games are covered mainly in WP2.1, and are therefore outside the scope of this deliverable, strictly speaking. However we felt that we should link this important related topic to our general evolution framework, therefore we provide this present section as a kind contribution from WP2.1 partners. The section shows the current state and plans for the use of Evolutionary Games for network design to support the evolution of services in BIONETS.

6.1.1 Basic features of evolutionary games

In the biological context, the amount of reward for an individual in interactions with other individuals is related to its reproduction capability. A higher reward (which can represent more food or more chances to mate) to a behaviour (used by some population or a sub-group of a population) implies a higher growth rate of that population (or in other words, of individuals that adopt that behaviour [102]).

The evolutionary games formalism is one of the basic mathematical tools designed for predicting population dynamics in the context of *interactions between populations*. This formalism identifies and studies two concepts. The first is called ESS (for *Evolutionary Stable Strategy*), and the second is the *Replicator Dynamics*.

The equilibrium is characterised by a property of robustness against invaders, which, in the biological context are called mutations. The equilibrium has the following properties. First, if equilibrium is reached, then the proportions of each population do not change in time. Secondly, at equilibrium, the populations are immune from being invaded by other small populations.

The replicator dynamics describes the evolution of the interacting populations. This dynamics can lead to convergence to the ESS, but in general, even if an ESS exists, there is no guarantee for convergence to it. Even in the absence of convergence to ESS, one may be interested in identifying weaker forms of convergence: convergence to a limit trajectory (possibly to a periodic one) that may occur from any initial sizes of populations within some subset of states (this is known as asymptotically stable system). If the subset includes all initial states then the system is said to be globally asymptotic stable. For the definitions of these and other notions of convergence and stability, see [71]. Trajectories may of course diverge, so that any two initial states give rise to distinct trajectories whose distance from each other need not converge to zero. Yet, even then, some form of stability can be defined: one in which all trajectories remain in some bounded set.

In the biological context, the replicator dynamics is a property of the population(s) that biologist observe, where as in engineering, the dynamics can be engineered. More generally, in trying to learn from evolutionary games on design of services in BIONETS, we are interested in designing evolutionary capabilities so that the services we are interested in have desirable properties at equilibrium. To enhance the evolution of services, we are also interested in designing the adaptation capabilities of the network so as to allow new services that may be developed to invade an existing configuration (which is possibly in some quasi-stable equilibrium).

Note that there are aspects of evolution which are not related to evolutionary games (both in biology as well as in our framework). The adaptation capabilities through evolution which we seek could be used not only in interactions with other populations, but also to adapt to a varying environment. An example in biology is the adaptation to climate changes (which caused entire species to disappear). An example in BIONETS is the adaptation opportunities of services to faster processors and larger and quicker memory units.

6.1.2 Definition of evolutionary games

Consider a large population of players. We assume that each individual occasionally needs to take some action (such as power control decisions, or forwarding decision). We focus on some (arbitrary) tagged individual. Occasionally, the action of some (random number of) other individuals interact with the action of that individual (e.g. other neighbouring nodes transmit at the same time).

For simplicity, assume that each individual has only two available actions: 1 and 2.

We say that the whole population uses a mixed strategy q^* if a fraction q^* of the population playing one strategy and the remainder $\bar{q}^*$ playing the other (This can be realised for example if each individual randomises between the strategies.) We define by $J(p, q)$ the expected payoff for our tagged individual if it uses a mixed strategy p while the rest of the population (with which it interacts) uses the mixed strategy q^*.

Suppose that the population uses a mixed strategy q^* and that a small fraction (called "mutations") adopts another distribution p over the two strategies. If for all $p \neq q^*$,

$$J(q^*, q^*) > J(p, q^*) \tag{1}$$

then the mutations fraction in the population will tend to decrease (as it has a lower reward, meaning a lower growth rate). q^* is then immune to mutations.

If there are n pure strategies ($n = 2$ in our case) denoted by $s_1, \ldots, s_n$, then a sufficient condition for (1) is that

$$J(q^*, q^*) > J(s_i, q^*), \quad s = 1, \ldots, n. \tag{2}$$

In the special case that the following holds,

$$J(q^*, q^*) = J(p, q^*) \text{ and } J(q^*, p) > J(p, p) \ \ \forall p \neq q^*, \tag{3}$$

a population using q^* are "weakly" immune against a mutation using p since if the mutant's population grows, then we shall frequently have individuals with strategy $q*$ competing with mutants; in such cases, the condition $J(q^*, p) > J(p, p)$ ensures that the growth rate of the original population exceed that of the mutants. q^* that satisfies (1) or (3) is called a an evolutionary stable equilibria (ESS).

ESS has first been defined in 1972 by M. Smith strategy in [113]. In 1982, Maynard Smith's seminal text Evolution and the Theory of Games [114]). appeared, followed shortly thereafter by Axelrod's famous work [9]. Although ESS have been defined in the context of biological systems, it is highly relevant to engineering as well (see [124]). In particular, in the context of competition in the access to a common medium, we can expect that a technology that provides better performance will gain more market shares on the expense of less performant technologies.

In addition to identifying ESS, the Evolutionary Game theory community is often interested in the actual evolution dynamics, i.e. of the actual convergence to an ESS (when it exists). Various models called "replicator dynamics" have been proposed for that, see e.g. [27] and [58].

We can learn and adopt notions from biology not only through the concept of evolutionary game, but also in applications related to energy issues that have a central role both in biology

as well as in mobile networking. The long term animal survival is directly related to its energy strategies (competition over food etc), and a population of animals that have good strategies for avoiding starvation is more fit and is expected to survive [61, 83, 84]. By analogy, we may expect sensor networks whose components have efficient energy strategies to live longer and to have more chances to survive [86].

6.1.3 Architecting evolutionary games in BIONETS

In WP2.1 work is in progress towards identifying parameters of evolutionary games in a BIONETS context that have an impact on adaptation of services. We are not aware of existing work in that direction.

A key issue in this context is the dynamics. Where as biologists come with models describing observed evolution in nature (which is the Replicator Dynamics), in Engineering we can go beyond this and determine new dynamics, or choose new values of parameters that are used by new or by already known dynamics.

To that end, within WP2.1 there are plans to study a number of examples of convergence and of non convergence of dynamics to an equilibrium, and see what we can learn from them. Some of these examples are:

- [5] where conditions for non-convergence as well as conditions for convergence are identified for non-cooperative routing games.
- [37], where convergence is established for a flow control problem,
- [4], where convergence is established for a decentralised power control problem.

The idea is to study conditions for which services have good conditions to evolve in the presence of changes that may occur at various time scales, and to which these services may have to adapt.

6.2 Convergence Properties of Distributed Evolutionary Processes

In this section we focus on the case of atomic services exchanging parameters in a GA-like fashion: with crossover and mutation, selection comes in as a consequence of interactions. The results of this research have been published in [32].

The basis for our work was the idea that, in order to evolve, services need to show the ability to drift towards better performance, resembling what several biological entities do. The key mechanism is the exchange, through a mating procedure, of data/code/parameters with other users. In other words, we exploit nodes mobility to enable node cooperation, shaped as exchange of code and/or parameters, so that the overall effect is a *distributed evolution process*. The success of this evolution process is quantified through a standard metric, i.e., the *fitness*, which represents also the driving parameter of the mating process. From the user's point of view, the process of service evolution is completely transparent, since what it experiences, in reality, is the evolution of the degree of satisfaction to the actual service provided, i.e., what we call *fitness*. In other words, and in order to keep the analysis and simulation scenarios simple, we will model *not* the actual service

evolution, but, rather, the evolution of the fitness value (i.e., performance and/or satisfaction level) experienced. It should be clear that our approach is rather simplistic, in that it confines the service into a black box and just looks at its output in terms of fitness. On the other hand, the evolution of fitness as an outcome of the evolution of the service code is something which should be evaluated on a case-by-case basis, whereas we are interested in getting insight into a more general framework. This is expected to provide useful information for the design of service mating policies, i.e., the algorithms that will actually drive the service evolution. In the following, we will provide, as examples, three possible service mating policies, derive the associated fitness mating functions (i.e., the functions that defines the fitness evolution process) and study their convergence properties. While this is not meant to be omnicomprehensive, it enables us to individuate stable and optimal service mating policies and to gain insight into the various factors influencing the design of an effective service mating algorithm. In particular, we are interested in understanding how some factors (i.e., the number of nodes, the nodes speed, the mobility model) affect the evolution process. In terms of fitness, we expect that services with a higher degree of fitness will have a higher chance to survive, so that, in the mating process, their *genes* (e.g., routines, code parameters etc.) are likely to be inherited by the offspring.

We define by $I_i(t)$ the fitness level of service i at time t, taken to be in the interval $[0, 1]$. A fitness level of 1 denotes a perfect service. We consider as system metric $X(t) = \frac{1}{N} \sum_{i=1}^{N} I_i(t)$.

We then introduce the following definitions:

Definition 1 *A service mating policy is called stable if it leads to convergence of $X(t)$ with unitary probability.*

Definition 2 *A service mating policy is called optimal if it leads to convergence of $X(t)$ to 1 with unitary probability.*

Please note that the condition of optimality is, in general, not sufficient for a mating policy to be efficient. Indeed,we actually want a service that is able to converge fast to the optimal operating point, which concerns the features of the transient behaviour of the process $X(\cdot)$, while optimality here refers to a steady-state characteristic.

We assume that the service of user i can be represented as a binary vector $\underline{v}_i = [v_i(1), \ldots, v_i(T)]$, $v_i(l) \in \{0, 1\}$, $l = 1, \ldots, T$ [1]. The fitness is then taken to be:

$$I_i = \frac{d_H(v_i, v^*)}{T}, \tag{4}$$

where v^* represents the (unknown) optimal service setting and $d_H(\cdot, \cdot)$ is the usual Hamming distance.

We introduce the following service mating policies:

[1] The representation of the service as a binary string is fully general, in that it applies to any ICT service. This abstraction, while enabling a general tractation, results in a simplistic approach with respect to "real" services. This complies with the main focus of the paper, which is to gain a deep understanding of this distributed evolutionary process; the application of the proposed framework to actual services is not straightforward, and is left for future work.

Definition 3 (Clonation mating policy) *Let us consider two nodes with respective fitness level I_1 and I_2 that get into mutual communication range. Let us assume, without any loss of generality, $I_1 > I_2$ (if $I_1 = I_2$ no mating takes place). Then user 2 downloads (clones) user 1's service. User 1 keeps its service unchanged.*

Definition 4 (Clone-and-mutate mating policy) *Let us consider two nodes with respective fitness level I_1 and I_2 that get into mutual communication range. Let us assume, without any loss of generality, $I_1 \geq I_2$ (if $I_1 = I_2 = 1$ no mating takes place). Then user 2 downloads user 1's service. Mutation is then performed on the new vector $\underline{v}_2$, by changing each digit independently with a given probability p (called the mutation probability). User 1 keeps its service unchanged.*

Definition 5 (Combine-and-mutate mating policy) *Let us consider two nodes with respective fitness level I_1 and I_2 that get into mutual communication range. Let us assume, without any loss of generality, $I_1 \geq I_2$ (if $I_1 = I_2 = 1$ no mating takes place). User 2 downloads user 1's service, i.e., the vector $\underline{v}_1$. A number k is uniformly taken in the set $\{1, \ldots, T\}$. Then, a new vector $\underline{v'}_2 = [v_1(1), \ldots, v_1(k), v_2(k+1), \ldots, v_2(T)]$ is formed. Mutation is performed on this vector, by changing each digit independently with probability p (called the mutation probability). User 1 keeps its service unchanged.*

To illustrate the fitness evolution process associated with such policies, let us consider the situation when two nodes, running the same service (but with different parameters), presenting fitness levels I_1 and I_2, respectively, meet at time t_{k+1}. We assume, without any loss of generality, that $I_1(t_k) \geq I_2(t_k)$ and $I_2(t_k) < 1$. Both I_1 and I_2 are taken to be in the interval $[0, 1]$.
In general, we have $(I_1(t_{k+1}), I_2(t_{k+1})) = \phi\,[I_1(t_k), I_2(t_k)]$, where $\phi[\cdot]$ is what we call the *fitness mating function*, that maps $[0, 1] \times [0, 1]$ into itself. The mating function is, in general, taken to be a stochastic function, defined on $\{\Omega, \mathcal{F}, \mathbb{P}\}$. For the three examples considered above, the fitness mating function takes the following form:

$$\phi[x, y] = (x, x)\,, \quad \textit{clonation mating policy,} \tag{5}$$

$$\phi[x, y] = (x, x + \xi)\,, \quad \textit{clone-and-mutate mating policy,} \tag{6}$$

$$\phi[x, y] = (x, \psi \cdot x + (1 - \psi) \cdot y + \xi))\,, \quad \textit{combine-and-mutate mating policy,} \tag{7}$$

where ψ accounts for the combination operator and ξ is a random variable accounting for the mutation operator. From the structure outlined in the definition, and considering that we can take v^* to be uniformly distributed over the set of all possible service vectors (this can be seen as a consequence of the maximum entropy principle) it is clear that $\mathbb{E}[\xi] = 0$ and $\mathbb{E}[\psi] = \frac{1}{2}$.
We are interested is in studying the convergence properties of the aforementioned policies. We assume that the initial fitness values are independently taken from a continuous distribution $F_0(\cdot)$. We get the following results, whose proofs (based on a simple use of the classical submartingales convergence theorem) can be found in [33]:

Proposition 1 *The clonation/clone-and-mutate/combine-and-mutate mating policies are stable.*

Proposition 2 *The clonation mating policy is not optimal.*

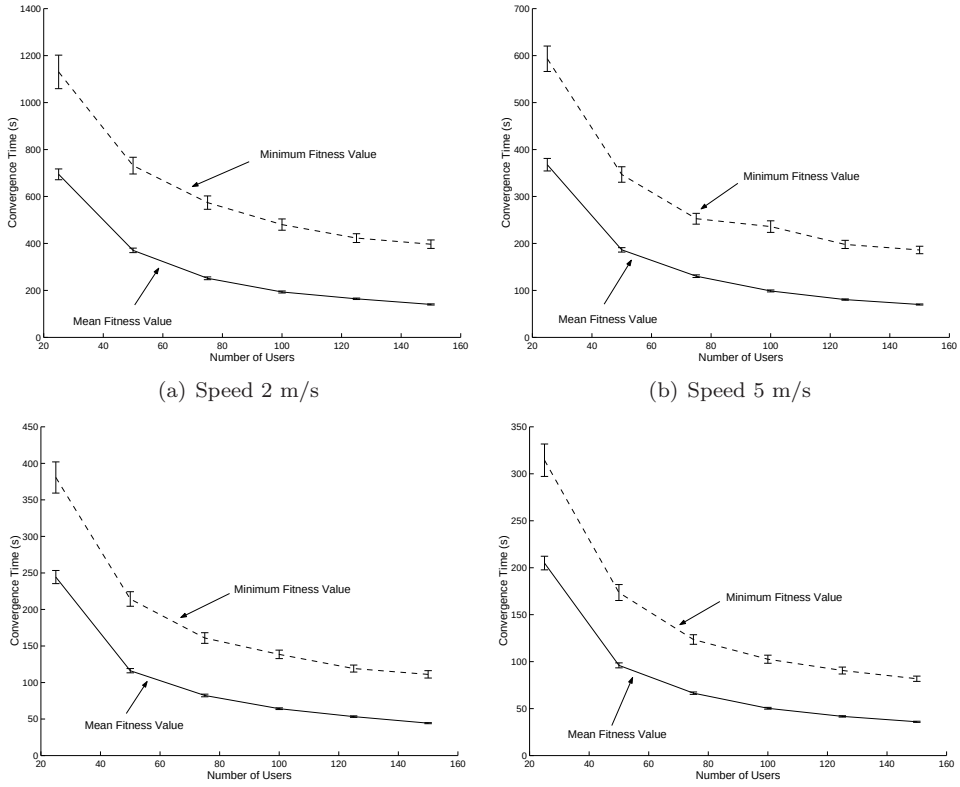
Proposition 3 *The clone-and-mutate mating policy is optimal.*

Proposition 4 *The combine-and-mutate mating policy is optimal.*

These results stress the importance of keeping some form of "openness" in the evolution environment. Such openness comes, in the proposed framework, by the introduction of the stochastic perturbation induced by the mutation operator. On the other hand, these results are useful in that they provide a first insight into design guidelines for efficient mating policies.

We have also developed a simulation platform for the proposed policies, using a freely available software tools [94]. We simulated, for the whole range of simulation parameters, the two optimal service mating policy outlined above, and took the mutation parameter p equal to 0.1. We denote by N the total number of nodes, and assume that they are constrained to move in a square of 2000×2000 m^2. Each node is assumed to have a transmission range of 50 m, and IEEE 802.11b-compliant PHY and MAC protocols are used [63]. The nodes are initially dropped according to a uniform distribution on the square, and then start moving according to either a Brownian Motion (BM) or a Random Waypoint Mobility (RWM) at constant speed v taken in the set $\{2, 5, 10, 15\}$ m/s. For the RWM model, the pause time has been set to zero; also since we are interested in the transient behaviour of the fitness level, the speed decay phenomenon does not play a significant role in our scenario [133]. Please note that our simulation of RWM is *not* a perfect simulation [77]. Indeed, we do not start the simulation with nodes distributed according to the steady-state distribution, but, rather, with a uniform one. This does indeed represent a pessimistic assumption, in that, as it may be easily understood, a uniform distribution of nodes over the area of interest is the distribution yielding the lowest probability of having nodes connected to one another (or, alternatively, the highest node isolation probability). Nonetheless, we believe that such an assumption leads to meaningful results in terms of comparison of the performance obtainable with BM and RWM. For the BM model, a billiard-like reflection was used when the mobile reached the edge of the domain. The initial fitness values are drawn from a set of i.i.d. random variables having uniform distribution in the interval $[0, 1]$. What we are interested in measuring is the *time to convergence*, that will be defined in two ways. First, fixing a threshold ξ (in the simulation, we will use $\xi = 0.95$), we want to measure the time it takes for the average fitness level to exceed ξ. Formally, $T_{conv}^{avg} = \min \left(t : \frac{\sum_{i=1}^{N} I_i(t)}{N} \geq \xi \right)$. Then, we are interested in the time it takes for the all the fitness values to exceed ξ, i.e., $T_{conv}^{min} = \min \left(t : \min_{i=1,\ldots,N} (I_i(t)) \geq \xi \right)$. Clearly, $T_{conv}^{min} \geq T_{conv}^{avg}$. Further, the smaller such convergence times, the more efficient the evolution process and the ability of the service to adapt to rapidly changing environmental conditions. Indeed, while the framework outlined in the previous section was able to answer our questions regarding the stability (i.e., the steady-state) of the distributed evolution process, we did not get any quantitative result on the convergence time, that is what in reality impacts the user's perception of the service quality.

The first issue we want to address is which of the two optimal mating policies described in the previous section is able to achieve the faster convergence. The results, in terms of 95% confidence



(a) Speed 2 m/s

(b) Speed 5 m/s

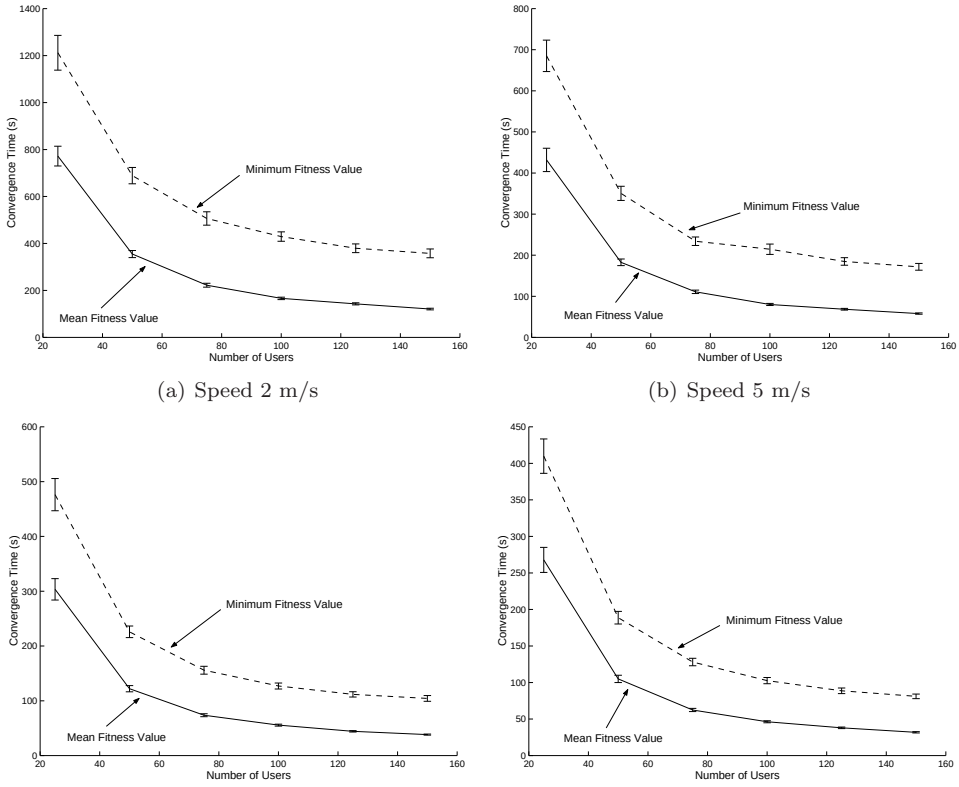
(c) Speed 10 m/s

(d) Speed 15 m/s

Figure 8: Convergence times for the clone-and-mutate mating policy, random waypoint mobility model.

interval for the convergence times, for the RWM mobility model are reported in Fig. 8 and Fig. 9 for the clone-and-mutate and combine-and-mutate policies, respectively. It can be seen that the clone-and-mutate has, in general, quite lower convergence times, showing thus higher ability to adapt to changing environmental conditions. On the other hand, the combine-and-mutate is able to achieve interesting performance figures when dealing with high-density high-mobility scenarios, the most interesting cases for the pervasive environments we are targeting. The combine-and-mutate policy shows then to represent an interesting choice, and we are currently investigating whether more complex extensions of such a scheme can be actually used to speed up the service evolution process.

We also tested extensively the case of BM mobility model, which resulted in worse performance with respect to the RWM case. In Fig. 10 and Fig. 11 we reported the convergence times for the clone-and-mutate and combine-and-mutate policies. The results are reported, in terms of 95% confidence interval, only for the cases of $N = 75, 100, 125, 150$ users moving at a speed $v = 10, 15$ m/s, because of the extremely long convergence times under such a mobility model. As it may be easily seen comparing these results with the ones in Fig. 8 and Fig. 9, the RWM model is able to achieve much better performance (almost one order of magnitude), in terms of convergence time. This phenomenon is worth some comments. In general, it reflects the fact that the inter-meeting times in the RWM are smaller than in the BM (see [54] for an extensive and in-depth discussion of



(a) Speed 2 m/s

(b) Speed 5 m/s

(c) Speed 10 m/s

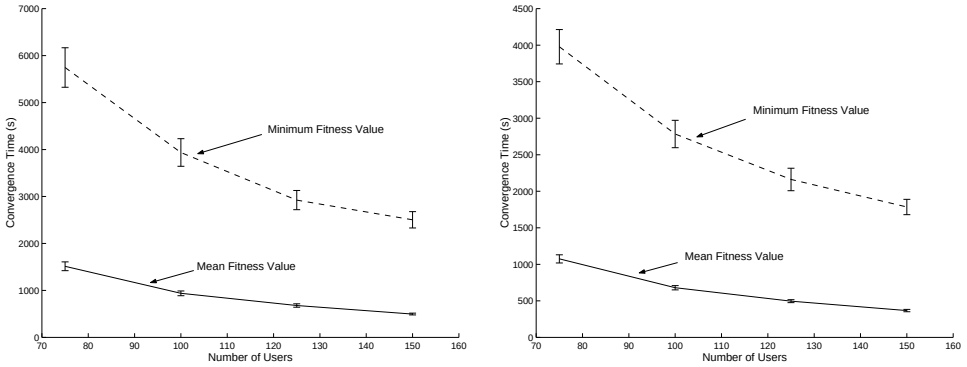
(d) Speed 15 m/s

Figure 9: Convergence times for the combine-and-mutate mating policy, random waypoint mobility model.

such phenomena). However, the difference in the inter-meeting times statistics is not sufficient to completely understand the difference between the two. Indeed, an extensive analysis of the trace files shows that the convergence (in particular for the minimum value) is driven by a few nodes which keep isolated for a very long time. This comes from the well-known tendency of the BM model to "move around", without getting far from the initial location. Hence, if a node is very far from all the others in the initial distribution, it will take him a very long time before getting in contact with the rest of the population. Further, nodes will tend to remain in closed clusters; the evolution process remains localised inside the cluster and hence becomes much lower, since such a "local" evolution takes place over a small population. On the other hand, in RWM, nodes tend to pass through the centre, and to meet more regularly with each other. In particular, a careful analysis of the trace files show that the time it takes for a node to get in touch with *all* other nodes is much lower in RWM than in the BM model. We might thus conclude that regularity in the traffic pattern helps in speeding up the convergence process.

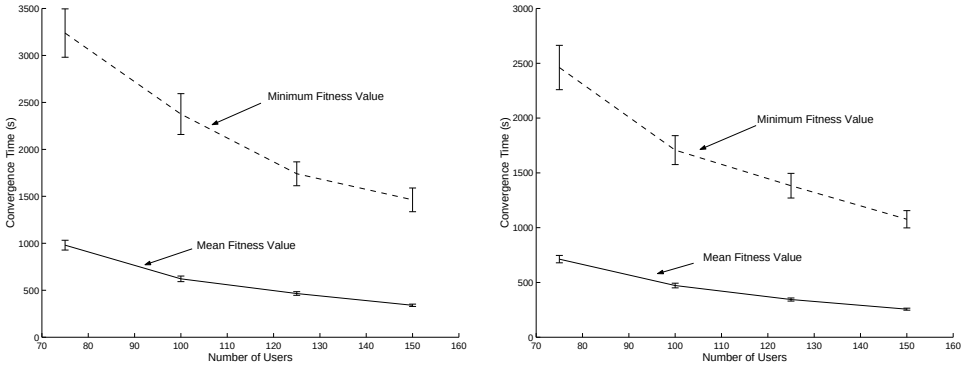
6.2.1 Current State and Next Steps

We are currently working on the dynamics of the processes described above, beyond just steady-state analysis as in the convergence analysis, by using dynamic programming and other tools. The goal is to achieve a definition of optimality of mating policies in terms of their dynamic behaviour.



(a) Speed 10 m/s (b) Speed 15 m/s

Figure 10: Convergence times for the clone-and-mutate mating policy, Brownian motion mobility model.



(a) Speed 10 m/s (b) Speed 15 m/s

Figure 11: Convergence times for the combine-and-mutate mating policy, Brownian motion mobility model.

In this sense, an optimal mating policy is one which achieves average fitness of one in the smallest possible time. We conjecture that the clone-and-mutate policy is optimal in this sense, though we have not been able to prove it formally yet. We also conjecture that a "weighted" version of the combine-and-mutate policy (in which the crossover point is chosen according to the fitness level of the two original services) is optimal in this sense as well.

Another interesting direction to pursue is the study of *adaptive* mutation policies. The idea is to mimic what happens in some real biological systems, in which, in the presence of a stress condition (which can be considered as equivalent to low fitness value), a high mutation rate can be observed. Coupled with natural selection, this ensures that in a relatively short time (usually of the order of ten generations) the organism gets back to a relatively stable state, characterised by a quite high fitness value and a low mutation probability. Since in our system the mutation probability is –in some sense– a "free parameter", we plan to extend our study to introduce more dynamical mutation policies.

6.3 Self-Organised Criticality and Evolution

In this section we aim at pointing out some models of real biological systems which play an important role in understanding and designing self-evolving services.

The first link which is apparent concerns the problem of describing interactions by means of graph-based models. This concept has quite a long successful history in systems biology, being driven by the need to understand the complex machinery underpinning the gene expression mechanism and interactions among molecules. A good survey of the application of graph-based models to such problems is reported in [23]. It is worth remarking that such models are extensively used in bio-informatics, and in particular in application of (stochastic) π-calculus, where they are used to model the interactions among molecules/genes.

An interesting property biological systems share with (some) computer networks is related to the degree distribution of such graphs of interactions. In particular, it has been extensively shown that a great number of such systems show a node degree which is distributed according to a power law [1]. Such models apply, in particular, to a rather wide class of evolving (or growing) networks, in which the attachment of new entities to existing ones follows a preferential model, i.e., newcomers attach more easily to well-connected nodes. Preferential attachment is, however, only one of the mechanisms which give rise to networks with power-law node degree distribution. The group of A.-L. Barabasi at University of Notre-Dame coined the term *scale-free* to describe such graphs. The characterisation in terms of node degree distribution alone is, nonetheless, not sufficient to explain some of the key properties of many naturally arising networks which possess such scale-free property. Indeed, such graphs can be characterised as *robust yet fragile*. This is due to the fact that such systems appear to be extremely robust in terms of fault-tolerance against random failures, but extremely fragile to targeted attacks. This comes from the fact that, in scale-free networks, there are "hubs" which tend to constitute a kind of backbone of the system, holding the system together. And these hubs represent the Achille's heel of such systems.

Another related property of most graphs arising in the study of biological and socio-economical processes is self-similarity (also referred to as scale-invariance). This refers to the fact that the system, observed at different scales, shows the same properties (i.e., shows a fractal structure) [115]. This involves the definition of a coarse-graining operation, which can be defined in terms of nodes pruning and collapsing. As an example, we could think of a graph of social contacts, where nodes corresponds to individuals and links correspond to some form of relationship. We could then, according to some well-defined rule, group such individuals in communities. We would therefore get another graph, where the nodes set consists of communities and links exist if individuals from two different communities are related in the original graph. The process can then be repeated to get the picture, at a different scale (or level of granularity) of the social network under consideration.

The fact that the nodes degree follow a power law is –in general– not sufficient to ensure that the network show a self-similar behaviour (and the term itself "scale-free" suggests inherently this self-similarity). A more detailed model can be given as follows [79]. Let us consider a network with

associated graph $\mathcal{G}$, and edge set $\mathcal{E}$. We denote by d_i the degree of node i. Then, we introduce

$$s(\mathcal{G}) = \sum_{(i,j)\in\mathcal{E}} d_i d_j.$$

Such a quantity takes a large value if nodes with high degree are connected to nodes with high degree. We define by s_{max} the maximum value of $s(\cdot)$, taken over the set of all graphs with an identical node degree distribution to $\mathcal{G}$. We then define:

$$S(\mathcal{G}) = \frac{s(\mathcal{G})}{s_{max}}.$$

Graphs with low $S(\mathcal{G})$ are said to be scale-rich, whereas a graph is said to be scale-free if $S(\mathcal{G})$ is close to 1. In this way, the definition of scale-free incorporates fractal-like behaviour.

In general, the term "self-similar" refers to the fact that some correlation functions show non-trivial power law behaviour. This is clearly an interesting property, from the point of view of a system engineer, in that it ensures that the resulting system is fully scalable. Interestingly, most biological systems are able to reach a steady state presenting self-similarity properties without the need of any external control. Such property is broadly referred to as Self-Organised Criticality (SOC), after the seminal work of [10]. While the suitability of SOC as a target goal for system engineers is currently debated in the complex systems community [79], it is understood that SOC is one of the mechanisms by which complexity may arise in nature [53].

System exhibiting SOC are, in general, not in equilibrium. They do present a steady state (in the broad sense that some average characteristics keep unchanged over time), but they do present variation in time. They are open dissipative systems, which require energy from the outside to offset the dissipation [40]. And it is worth remarking that BIONETS systems are intrinsically open dissipative systems, where the free energy needed to self-organise the services (and, consequently, the supporting network) comes from socio-economical processes.

It is also particularly appealing that some simple models of evolution in an ecosystem lead to a non-equilibrium steady state exhibiting SOC. One of these, on which we will focus due to its simplicity, is the Bak-Sneppen model of co-evolution of species. It presents all the key BIONETS "ingredients" (natural selection and interactions) and is therefore useful to gain insight into how SOC may appear in an evolutionary BIONETS system.

The Bak-Sneppen model considers N co-species, represented by points on a circle. Initially, each specie is assigned a fitness value uniformly taken in the set $[0, 1]$. The system dynamic works in rounds. At every round, the specie with the minimum fitness is eliminated from the system, together with its two adjacent species on the circle (this accounts for interactions among different species). These are replaced by new species, with fitness taken independently and uniformly in $[0, 1]$. This model can be shown to present SOC features [85].

7 A Framework for Self-Evolving Protocols and Services

After describing the evolution problem in Section 3, breaking it into micro- and macro-levels in Section 4, and selecting promising building blocks for a solution at each level in Sections 5 and 6 respectively, we now put all these puzzle pieces together in order to obtain a generic framework that can later be instantiated for service (in WP3.2) or protocol (WP1.2) on-line evolution.

The framework highlights two important aspects: The first is a link between micro- and macro-levels that allows us to study the impact of micro-level program transformations on the general behaviour of the system, reflected by its capacity to achieve the desired goals set by the user. The second is the communication aspect inherent to a pervasive network context: communication enables the interactions that drive the evolution process. To cover these two aspects, we identify essential elements for evolution, and propose an extension of evolutionary computing using these elements. The extension is based on an interaction graph model through which services can exchange information for evolutionary purposes.

We start by identifying the evolution elements in Section 7.1 and then in Section 7.6 we describe the graph model and the corresponding extension of the evolutionary computing steps in Section 5.1 to our distributed on-line context.

7.1 Elements of the Evolution Framework

According to the BIONETS paradigm proposed in [29, 31] (which was the seed for the BIONETS project), evolutionary services are modelled after *living organisms* which are *user-situated*. They are hosted on U-Nodes and move through the physical movement of the users through the network, which is the *habitat* of such moving organisms.

Each service is governed by a program and its related data that are organised into *chromosomes*, which constitute the *genetic information* that encodes their behaviour and goals. According to [29], "chromosomes are collections of genes that are the smallest service (related) data unit and inborn intelligence/instincts and thus represent all the information needed by the organism to function and by the service to be executed".

Note however that the service designer might decide not to encode all the service implementation information in a "chromosome" that is exported and submitted to evolution rules. The way chromosomes are encoded and exchanged determines which parts of the service are subject to evolution, and which parts are fixed. In the BIONETS plan, this relates to the four roadmap phases described in Section 3.4: in a first phase, only service parameters would be subject to evolution; and so on, up to a fourth phase where the service code could also evolve.

A *mating process* is defined as the exchange of genetic information among services, which results in *offspring* representing new service variants generated on user's devices. Through mating preferences, some services will have a higher chance to reproduce and therefore to spread. A *fitness* measure is available for this purpose. This mechanism provides a *selection pressure* similar to natural selection which pushes services to constantly optimise themselves, leading to evolution.

As in nature, it is possible to define a complete life-cycle of our "service organisms", starting

from the "birth" of an organism (service creation), then reproduction (with the purpose of evolution) and death (service deprecation). This life-cycle is now being defined in detail in WP3.2. Within the BIONETS project we are also extending the notion of evolving services proposed in [29, 31] to communication protocols in WP1.2.

The specifics of protocol and services with respect to generic software require that communication, i.e., interaction between evolving portions of the network, be placed at the forefront of the evolution framework. This communication happens in the context of a pervasive environment as described in Section 2, in which disconnected operation is often a rule rather than an exception. This is depicted in Figure 12 where "service chromosomes" are subject to genetic transformations such as crossover and mutation (indicated by dotted arrows on the figure), and are exchanged via the network among U-Nodes (represented by thick arrows between big red dots). T-Nodes are represented as small blue dots that do not participate in evolutionary interactions but which may be influenced by them, as the sensor data they provide might be treated differently as a result of new emerging services.

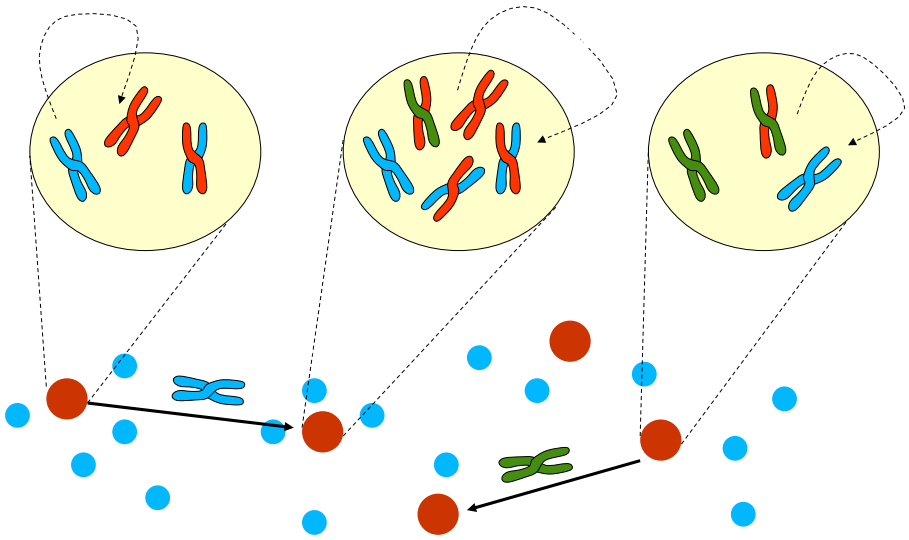


Figure 12: Distributed On-Line Evolution in the BIONETS Architecture

Evolution should happen such that it improves system operation while keeping it robust and safe. These requirements seem contradictory, since evolution implies change, and change involves risks. The only way to achieve such goal is to build resilience and self-healing features at the core of every service, such that the system is able to recover from eventual disruptions in a smooth and autonomic way.

We can therefore identify the following essential "pieces of the puzzle" without which the system is not able to evolve autonomically:

- **High-level goals**: User-level goals that the system should constantly seek to satisfy in spite of changes in the environment.

- **Monitoring**: Measure the system behaviour and performance, as well as external environment conditions.
- **Fitness evaluation**: Evaluate the *fitness* of a given protocol or service as a measure of its performance against the expected high-level goals.
- **Feedback**: Provide fitness feedback to the evaluated services; the latter ones can use this feedback to improve themselves.
- **Selection**: Select those services or protocols that are best suited for a given task according to their fitness, and replace outdated or misbehaving ones.
- **Transformation**: Operate changes on services and protocols with the aim of improving their performance. Common transformations include mutation and crossover, but could also include different operations. A transformation could involve simple changes in parameters, but could also imply code change, ultimately leading to a new version of a component, with potentially new functionality and behaviour.
- **Communication**: Exchange of data through the network, necessary to exchange fitness information, to perform transformations that involve two or more components (e.g. crossover), and to disseminate the best service and protocol variants to regions of the network where they are needed.

Some of the pieces above are *functions* that can be performed by some logical units: monitoring, fitness evaluation (using monitored information), selection (using fitness information), transformation, communications. Others are *signals* that are obtained by a logical unit via an interface: high-level goals, fitness feedback (after evaluation), monitoring information concerning service performance (input for fitness evaluation). We now describe some of these building blocks in more detail.

7.2 High-level goals

The user interface with an autonomic system is a crucial element to obtain high-level goals from the user, translate them into an initial corresponding service configuration, and use them later on to keep the service optimised.

Besides that, fitness feedback from the user is also a form of expressing high-level goals, and also of expressing changes in these goals. This means that a drop in fitness for a service whose performance remains unchanged may indicate a change in the desired high-level goal expressed by the user.

The simplest form of feedback is just switching applications or services: if the user is not satisfied with the results, he or she simply switches to a potentially better one, until the desired result is achieved. Currently this happens mostly in an ad hoc way. The underlying services running on the machine are totally unaware of the reasons why a user might opt of this or that application. They are therefore unable to use this feedback to try and improve their own performance.

In the future we envisage systems that are able to learn from feedback provided by the users. The article [108] contains several pieces of text and citations that illustrate this research goal: *"Can you think of some chore or duty that a person does that she doesn't do better the second time? Or can you think of some chore or duty that a computer does that it does do better the second time? [...] In people ... learning and adapting take place at many levels simultaneously and continually, and we in Machine Learning must find out how to build our models and software to do so too. [...] People learn because they* care *– that is, they have purposes or motivation. The essence of a purpose is choice and evaluation through some form of feedback [...] I really want my agent to be what I am going to want in the future, even as those wants change and evolve. [...] purpose structures in ambitious pieces of software would enable adaptation at many levels simultaneously, making it far easier for programmers to design that software and to understand and correct errors."*[2]

In the field of autonomic communication, several research groups are dealing with problem of obtaining and manipulating machine-readable semantic-rich knowledge [38, 69, 107] in order to facilitate the human-machine communication needed to express such user-level goals and other tasks. We assume that somehow the problem will be solved, and that feedback from the user is readily available at some machine-readable form that can be interpreted by the underlying fitness feedback system.

7.3 Transformation

A fundamental problem to be solved is how to obtain successful program transformations, starting with an initially correct program (one of our assumptions from 3.3).

First of all, the meaning of "successful" is highly application-dependent, and implies that the user interface with the autonomic system is crucial to obtain this information from the user, as explained in Section 7.2 above. On the other hand, transformations might also apply to services which are not directly connected to the user, but provide services to other services, in a cascade service composition way, as will be explained in the graph model of Section 7.6.

We assume that the fitness of the resulting transformations of a given service can be evaluated by their client(s) as explained in Sections 7.4 and 7.6.

Two problems then remain: how to proactively detect that a transformation is necessary or desirable, and (probably the hardest one) how exactly to operate on the program to make the transformation happen with the highest chance of success.

Autonomic program transformations are one of the most difficult challenges that we have to face. Even mere parameter changes may lead to service disruption when not carefully engineered. Code transformations are even harder: it has been shown that only a small fraction of assembly-like Turing-Complete programs evolved by genetic programming halts, and this fraction decreases sharply with the length of the program and with the number of jump instructions [76]. This is not the case with tree-based (Lisp-like) programs that are commonplace in genetic programming; these are in most cases not Turing-Complete, and the set of instructions is carefully chosen to

[2] excerpts from [108] which contains itself other citations, so please refer to [108] for more details on the exact sources of each sentence.

avoid loops. However, tree-based programs are not guaranteed to operate correctly either, so other forms of disruption may occur, which make them unfeasible for on-line evolution. As highlighted in Sections 5.2 and 5.3, we are currently investigating *chemical programming languages* in which instructions are represented by molecules and the execution proceeds as chemical reactions. Code transformations are a natural process in a chemical program, which makes them more robust to the transformations necessary to achieve evolution.

It is also essential that bad or unsuitable programs be detected and eliminated. One could argue that it would be even better if the system *never* generated bad or unsuitable program variants. For instance, one could argue that by using a completely deterministic and formal approach, it should be possible to derive provably correct programs, that will always satisfy the specifications. In reality however, this is not a feasible approach in practice, except for very simple problems. This is typically true in an on-line setting, and where unpredictable changes may occur, which is the scenario considered in BIONETS.

Therefore we argue in favour of a *resilient approach to evolution*, that considers failures as part of the operation, and that take action accordingly, in a self-healing way. This approach is more robust and autonomic, since it does not require provably correct transformations but rely on the selection system to promote good transformations and filter out bad ones. This also makes the system more generic, because if formal transformations methods are available then the system can benefit from them since they would lead to high-fitness programs.

7.4 Fitness Evaluation and Feedback

One of the critical issues for evolutionary processes to take place is the implementation of a natural selection process. Feedback in the form of fitness is the driver of such process. Recall from Section 3.3 that we assume that the starting point for evolution is a running service that needs to continuously self-optimise and adapt to changes in the environment. Evolution as a self-optimisation process can be modelled and understood in terms of closed-loop control systems, where the environment can be used to gather information for tuning and optimising the system behaviour in an automatic manner. This is depicted in Figure 13.

A fitness function has two components: (i) an objective component (user-independent), which reflects the properties of the running service in terms of performance, i.e., ability to attain the target goals (ii) a subjective component (user-dependent) which reflects the level of satisfaction of the end user. While part (i) can be implemented in a software system (it just requires a monitoring unit), part (ii) seems more challenging to obtain. Indeed, it implies a lot of work in terms of human-computer interface, for understanding, in a transparent automated way, what the user expects from the system.

Assuming that somehow this "fitness" level can be estimated, the next step is to determine how to react to it. In general, the fitness can be understood as a feedback signal, generated by the environment. The general picture is therefore that of a classical closed-loop control system, as in Figure 13, acknowledged to represent a general model for self-managing systems [41]. The system is constituted of the service unit which receives an input from a unit (called evolution engine)

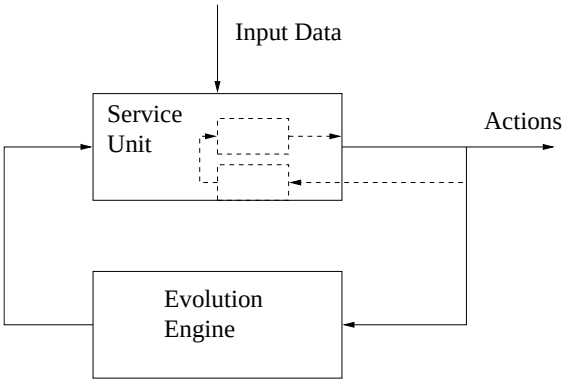


Figure 13: Representation of a self-evolving service as a feedback control system. The dashed inner-loop control represents an adaptive mechanism hardwired in the service code.

which, upon evaluation of the service fitness (by considering the actions taken, comparing them with the high-level goals and considering also user-dependent fitness components) determines the modification to be done to the service itself. The inner feedback loop in Fig. 13 accounts for the fact that a service can include a hardwired adaptive mechanism. Such inner loop is able to change only the behaviour, but not the functionalities of the service. The evolution engine comes into action when functionality changes are involved.

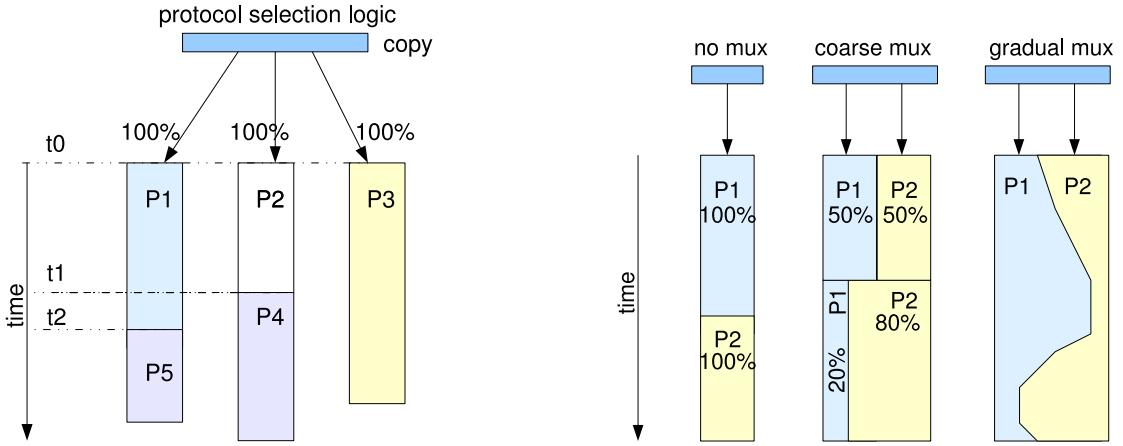
7.5 Selection

Once fitness information is available, how can a client decide which of the (potentially competing) service providers to choose? In an ideal world, choosing among two or more potential services could be based on full knowledge of the factors that determine the services' fitness for a given task and situation. This knowledge could be gained either through analytic work and/or through past experiences, and incorporated into an intelligent decision making system that would then be able to pick the right service for the right task. An algorithm to perform such choice is proposed in the context of autonomic computing [132], to automatically select among competing components based on their workload characteristics.

Our hypothesis is that a complete modelling will in general not be feasible in a dynamically changing environment. In an evolutionary framework, *"on-line experiments"* are a necessary addition to classical assessment methods and can be applied in all situations, even when no model is available.

We have proposed a framework for dynamic evaluation and selection of protocols, and have performed some initial feasibility experiments to show how an application could automatically select from a set of protocols given their fitness, evaluated over a given period that determines the duration of the experiment [104]. Two types of *on-line experiments* have been be envisaged: *Serial* and *parallel* experiments.

Figure 14 shows how these parallel and serial experiments can be performed. Figure 14(a) shows a typical parallel experiment. At time t_0 the experiment starts with three protocols in



(a) parallel experiments (b) serial experiments

Figure 14: Several cases of parallel (a) and serial (b) experiments online.

parallel, P_1, P_2, and P_3. Each protocol receives a full copy of the traffic, leading to redundancy which is used for fault tolerance: if one of the protocols misbehaves, the service is still delivered by one of the others. At t_1 the experimenter decides that it is better to replace P_2 by P_4, and at t_2 P_1 is replaced by P_5.

Figure 14(b) shows three possibilities for serial experiments: the first one (left side) is the simplest case, where only one protocol is used at a given time, and the experimenter switches protocols in response to observations from the monitors and the checker. In the second case (middle) a coarse multiplexing of protocols is performed, with part of the traffic sent to one protocol and part to the other. The shares of the different protocols vary in coarse chunks. The third case (right side) shows a fine-grain, gradual multiplexing scheme in which the percentage of traffic that goes to each protocol is adjusted dynamically.

The choice of the observation and switching period is crucial: too short observations do not provide sufficient statistical information, and too long observations provide slow feedback to switching decisions. A too short switching period does not leave sufficient time for new protocols to initialise and adapt to the new environment, does not capitalise on the transition time spent, and could lead to undesirable performance oscillations. A too long switching period would result in slow adaptation.

We have simulated the selections scheme for two cases: the selection of a reliable transport protocol for a simple file transfer application, and the selection of proper loss concealment mechanism [105] for a voice over IP application. See [104] for the detailed results.

Our results show that designing such a selection and evaluation functionality for an on-line setting is a challenging problem. First of all, a definition of “normal” service operation must be considered, in analogy to intrusion detection systems [49]. Although some works propose schemes to measure protocol normal behaviour [49], the functionality check of a service from a high level description is still mostly unsolved.

Fitness evaluation requires continuous tests, whose duration and resources consumed should be carefully assessed. Moreover, tests may affect network behaviour, making the comparison between protocols difficult.

The appropriate monitoring and "service/protocol switching logic" needs to be carefully studied in order to detect the correlation between actions and events, which is a classical problem in feedback systems needed for continuous evaluation. The work of [132] provides a good entry point for this, but more needs to be done to obtain flexible strategies valid for any number and type of protocols/services in parallel or in series.

The coordination (signalling) needed is also a matter of concern: first of all, switching protocols must happen in a coordinated way such that all end points use compatible protocols. This is one of the reasons why in Section 7.6 we make a simplification by starting with *atomic services* as opposed to fully distributed protocols. Secondly, fitness should be reported by the remote side in an accurate and trustworthy manner, in order to detect and eliminate unsuitable programs, like the *cheat* programs that could emerge by genetic programming, which have been discussed in Section 5.3.2.

Even if very challenging, we believe this is a promising research direction, ultimately leading to future networks in which protocols can be synthesised automatically, and which evolve to match new needs, and heal themselves against disruptions.

7.6 A Graph Model of Atomic Service Evolution

We are dealing with a distributed communication system, in which interactions (understood as the possibility of transmitting information between service entities) take place by means of communication. Therefore, the ability to communicate is a *necessary* (but not sufficient) condition for the ability to interact.

In this context, modelling all the elements listed in Section 7.1 is a complex task in general. Here the distinction between higher-level services and communication protocols comes into play: While some services may be distributed entities spanning several nodes, other services may run at a single node: we refer to the latter as *atomic services*. On the other hand, communication protocols *always* run on multiple nodes by definition: there is at least one source and one sink in a communication session. This means that evolving a protocol or a distributed service requires a distributed infrastructure comprising a trusted distributed fitness evaluation mechanism.

We have started by considering *atomic services*. Atomic services are composed of a single element whose fitness is evaluated at a single location. The resulting simplified model has been published in [88] and is summarised here. In this context, modelling a distributed service may be simplified by locally evaluating each of its constituent parts independently. This model is deficient since it loses sight of the global service behaviour. However it is useful as a starting point.

We model atomic service interactions using graphs, as follows: Given a system consisting of N nodes, each one representing a service, we can define an undirected graph $\mathcal{G}_{interactions}$ having vertex set $\mathcal{V} = \{1, \ldots, N\}$ and an edge between node i and node j if and only if atomic services i and j actually interact.

There are several types of interaction, including client-server, direct peer interaction (for information sharing between peers belonging to the same distributed service), and breeding (crossover) interactions for evolutionary purposes.

An interaction is considered over a given timescale T. Considering the mobility of users/devices/services in the network, we can also construct a connectivity graph $\mathcal{G}_{connectivity}$ over the timescale T, where an edge is drawn between node i and j if, over the considered timescale, they happen to belong to the same island of connected nodes, and there exists an underlying bidirectional communication path connecting i and j. If all the nodes were static, $\mathcal{G}_{connectivity}$ would represent, for any T, the overlay topology of connectivity among nodes hosting service interconnection.

To complete the picture, we also need to define two other graphs. The first one is called *evolution graph* and denoted by $\mathcal{G}_{evolution}$, where the existence of an edge (i, j) means that the interaction between services i and j corresponds to the exchange of information for evolutionary purposes (i.e., they can give rise to new services by crossover operations). Services i and j must be alternative implementations of the same service. It is easy to understand that

$$\mathcal{G}_{evolution} \subseteq \mathcal{G}_{interactions} \subseteq \mathcal{G}_{connectivity}.$$

The second graph is called the client-server graph and denoted by $\mathcal{G}_{clientserver}$. It is a *directed* graph, where the existence of a $i \rightarrow j$ link means that service i is a client of service j. A weight on link $i \rightarrow j$ indicates the fitness of service j as evaluated by its client i. A client-server relationship between two nodes is a form of interaction, therefore:

$$\mathcal{G}_{clientserver} \subseteq \mathcal{G}_{interactions} \subseteq \mathcal{G}_{connectivity}$$

Note that the first inclusion means that the set of directed edges of $\mathcal{G}_{clientserver}$ is a subset of the set of undirected edges of $\mathcal{G}_{interactions}$. This can be achieved either by ignoring the direction of the links in $\mathcal{G}_{clientserver}$, or by replacing every undirected link in $\mathcal{G}_{interactions}$ with two corresponding directed links. Further, it is worth stressing that $\mathcal{G}_{clientserver}$ and $\mathcal{G}_{evolution}$ have no direct relationship with each other, apart from the fact they are subgraphs of the same graph $\mathcal{G}_{interactions}$.

Since the interactions related to evolution rely on connectivity, the network dynamics is one of the drivers of the evolution process. However, as already stressed, this provides just a necessary condition for interactions to take place. We believe that socio-economic processes shall be accounted for in order to reach a complete picture of the interaction process between services in BIONETS. The field of Social Network Analysis [125] will help in the analysis of the interdependencies between the technical infrastructure, the business and service transactions, and the social networks of users.

While the concept of connectivity graph is well-understood in the case of static fixed networks (and a lot of papers within the complex systems community have revealed the implications arising from the properties of the Internet's topology [1]), there is no global theoretical framework for understanding connectivity in a mobile disconnected scenario. In this section we provide a first example on how to define such connectivity graph. For the sake of clarity, we will refer to (atomic) services as nodes of the graph; when we say that two services meet, we mean that the devices hosting them get within mutual communication range and are (physically) able to exchange information.

Consider, again, a set of N users, possibly mobile, and fix a time window T. We will focus on the time window $[t_0, t_0 + T)$, for an arbitrarily chosen time instant t_0. Consider the contact process $Z = (t, \sigma)$, where $t_0 \leq t < t_0 + T$ is the time instant at which a meeting takes place[3] and σ is a mark of the form $(i, j),\ i, j \in \{1, \ldots, N\}$ being the IDs of the nodes that become able to communicate at time t. When an island gets formed, all the possible pairs of nodes are considered. At t_0, the process registers all the paths in the initial configuration. We can build a connectivity graph in the following way. Consider an initially empty graph with vertex set $\{1, \ldots, N\}$. If the process Z admits a mark (i, j), we draw an edge between i and j, if not already present. In this way, we can construct the graph $\mathcal{G}_{connectivity}$.

Our aim is to extend the algorithmic steps of GAs and GP to this graph context, using a catalysis graph model. In order to do so, we first introduce a chemical model, introduced for studying the origin of life problem in [66]. Indeed, many researchers studying the origin of life problem have come up with hypotheses based on chemical reaction models, on how life would have emerged out of simple molecules. The problem is to understand under which conditions order (in the form of structure and complexity) may appear given an initial set of simple molecular species which interact in a pseudo-random fashion.

A chemical reaction takes a number of *substrates* or *reactants* and produces a number of *products*, sometimes with the help of a *catalyst*, a substance that accelerates or facilitates the reaction without being consumed in the process. A molecule may act as catalyst or inhibitor for different reactions. When the supply of substrate is abundant with respect to the amount of catalyst, the rate of product formation depends directly upon the amount of catalyst. The model in [66] relies on this simplifying assumption, and models molecules as catalysts or inhibitors for the production of other molecules. Moreover they consider the system evolution over time-scales long enough so that any molecule may interact with any other molecule in the system. The least fit molecules (in terms of relative population) are dropped at each round and are replaced by new molecules.

While perhaps too simplistic from a chemistry point of view, this model can be easily transposed to the service evolution case: services can be modelled as molecules that act as catalysts for other services, providing incentives for them to spread over the network, or may inhibit services that are not suitable.

In [66] catalytic and inhibitive features are modelled by means of a directed graph, where a positive (respectively: negative) weight associated to a link $i \rightarrow j$ means that molecule i is a catalyst (respectively: inhibitor) for molecule j. No self-loops are considered.[4] It is shown that such model experiences a phase transition, consisting in the sudden appearance of a set of molecules that take over the others, i.e. some form of "order" emerges. The arising of order can be well understood here by considering autocatalytic sets (ACSs). An autocatalytic set is defined as a set of molecular species which contains a catalyst for each of its member species. ACSs act as self-replicating structures, even if none of the constituent molecules can self-replicate in isolation. Two examples of ACSs are shown in Fig. 15. In this figure, the nodes s_i are catalysts according to

[3] Referring to [34] for a more formal definition of the meeting process, we define a meeting as the event that two nodes are able to transmit data.

[4] The absence of self-loops ensures that no self-replicating molecules can be present.

the simplified model in [66], and the links between nodes have a constant weight $c_{i,j}$ representing the strength of the catalytic (when $c_{i,j} > 0$) or inhibitory (when $c_{i,j} < 0$) effect (only a few sample node labels and weights are indicated, for the sake of simplicity). The simplest ACSs are cycles (also referred to as hypercycles [8, 48]), but not all ACSs are cycles; on the other hand, every ACS contains at least one cycle. It is possible also to show that (i) ACSs appear with high probability in a finite time, given any initial condition (ii) once an ACS appears, the phase transition takes place at exponential speed [67]. This example shows the importance of interactions (and cooperation) in evolution.

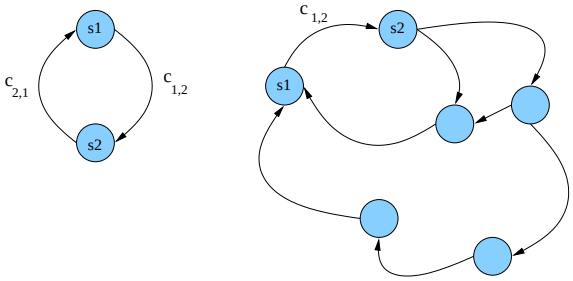


Figure 15: Two examples of autocatalytic sets.

It is worth adding a remark with respect to the absence of self-loops in the model. Self-loops represent a degenerate form of ACS, in that a self-loop with positive weight would indeed characterise, in the simplified chemical framework, a molecule able to self-replicate. The dynamics of such molecule is of little interest (the concentration would just grow exponentially over time). It is however not clear at the moment if self-loops can provide an interesting model for service evolution, i.e., whether services should be allowed to self-replicate (like worms do). It is interesting to note that self-loops correspond to a quasispecies model in systems biology, describing the dynamics of a system of self-replicating entities. This has proved to be useful in modelling the processes of self-replication of macromolecules (DNA, RNA). Self-replication can provide diversity by means of mutations (taking place as errors in the process of replication) and is used also as a model for asexual reproduction of bacteria and viruses. Its direct applicability to service evolution is an open issue, due to obvious security considerations. This is an example of the typical trade-off between security and flexibility (provided here by automated functionality extension).

Now we can provide a generalisation of GA/GP based on such model of chemical interactions.

Step 1. Representing Candidate Services

In GA/GP we generally have a set of candidate solutions for a single given problem, while in our case we have a full graph of distinct services, each solving a separate problem as requested by their clients. Furthermore, several instances of the same service may be available in the network, according the demand for that service. In order to model that, we take inspiration from the catalysis model introduced above.

Initially ($t = t_0$), a set of candidate services are available, represented as the nodes of $\mathcal{G}_{clientserver}(t_0)$.

Each node i has an associated "concentration" value k_i, representing the initial amount of catalyst i available in the system. Fig. 16 shows two examples of client-server interaction graphs, where s_i represents a service and a weight $c_{i,j}$ on a link represents the fitness of s_j as evaluated by s_i. Note that, in contrast with the autocatalytic sets mentioned above (compare with Fig. 15), we focus on acyclic graphs. We do not know yet whether cyclic graphs would emerge or make a sense in real-world client-server interactions. This remains to be verified in future experiments.

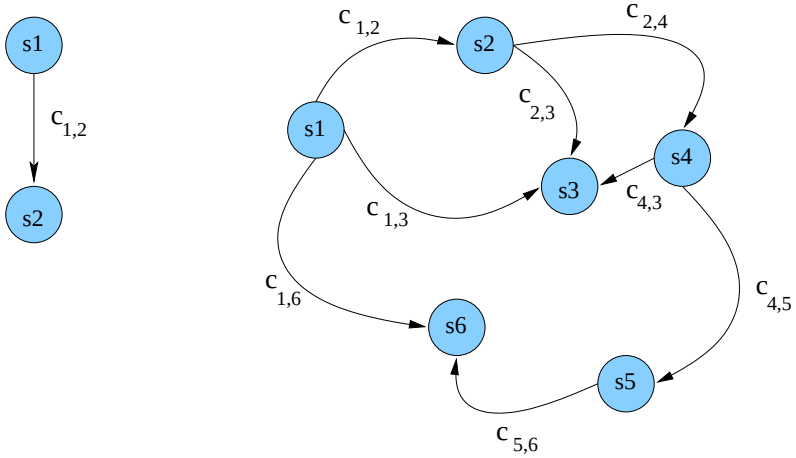


Figure 16: Two examples of acyclic client-server interactions

Assuming that the concentrations of reactants are large enough (as in [66]), the rate of product formation when catalyst i catalyses product j is:

$$\frac{\partial k_j}{\partial t} = \alpha_{i,j} k_i - \beta k_j \tag{8}$$

where $\alpha_{i,j}$ is a constant weight on link (i, j), and $\beta > 0$ represents a natural "death rate".

The rate of product formation can be controlled by the quantity of catalyst and the constant α, while the rate of destruction of the same product is determined by β.

Considering that one service may have several clients, (8) can be extended to a set of equations for the system dynamics:

$$\frac{\partial k_j}{\partial t} = \sum_i \alpha_{i,j} k_i, \quad j \in \mathcal{V}, \tag{9}$$

where $\alpha_{j,j} = -\beta$ by definition and $\alpha_{i,j} = 0$ if no (i, j) link is present in the client/server graph.

Since the physical location of services is not explicitly modelled here, a high concentration of a substrate means that more copies of the service are available in the network, no matter at which physical nodes these copies are located. In case they are all in the same machine, it actually means that the weight or popularity of this service on this machine is high (the physical process itself is not necessarily replicated, as in the case of a web server serving multiple clients). This model matches well with multisets in chemical computing (see Section 5.2), in which copies of the same instance are represented by a simple multiplicity counter.

Step 2. Distributed Fitness Evaluation

In contrast to GA/GP, it is now up to each client to evaluate the fitness of the services it uses. Each service may be used by a set of clients, and may thus receive different evaluation feedback from each one.

In order to model this distributed fitness feedback flow, we define $\alpha_{i,j}$ in Equations (8) and (9) as the fitness of service j as evaluated by its client i during period T. In this way, the amount of a service increases with its fitness and the amount of clients demanding that service. A constant death rate ensures that a service that is not very useful will not indefinitely consume resources in the system.

The range of fitness values must be the same over the whole system. If we adopt the convention of [66] (catalysis versus inhibition) we could establish a fitness range of $[-1; 1]$ with -1 meaning a totally harmful service, 0 a useless service (but not harmful) and $+1$ a totally fit (perfect) service.

Since the weights directly influence the rate of catalysis of the service according to (8) and (9), a negative weight (or fitness) contributes to decrease the service "concentration". When the concentration reaches zero the service is eliminated from the system. This provides a way to eliminate harmful services from the system. When a service i is eliminated, all the services that it used will stop receiving the positive feedback on the fitness, and will eventually also be eliminated.

Step 3. Selection

The selection mechanism operates according to a chemical reaction model based on multisets [14]: substances that are present at a higher concentration in a chemical solution are more likely to meet other compatible substances with which they can react, or which can catalyse new substances. When looking for a suitable service, a client may send a service discovery query to the neighbourhood to find suitable candidates. More popular services are more likely to be hit by such queries, and therefore bind to the potential clients, creating a client-server link in the interaction graph. The service discovery mechanism is orthogonal to the chemical model but influenced by it, in the sense that services occurring at higher concentrations have a higher likelihood to be discovered than those occurring at low concentrations.

Step 4. Generating new candidate services

The graph system is dynamic. The elimination, addition and movement of services, as well as interaction changes, are reflected by transformations on the graphs $\mathcal{G}_{clientserver}$ and $\mathcal{G}_{evolution}$ which occur every period T, with a direct impact on $\mathcal{G}_{interactions}$. Nodes and links may be added or deleted, and weights may be updated based on new fitness evaluation values.

Two genetic operators are modelled: mutation and crossover. A mutation of a given service s into its variant s' means that at the next instance of the graph ($t_{u+1} = t_u + T$) node s will be replaced by node s'. The potentially new behaviour of node s' due to the mutation might have an

impact on its fitness evaluation, therefore the weights on its incoming/outgoing links might change as a consequence.

A crossover between two services s and r at time t_u is represented by an edge (s, r) on $\mathcal{G}_{evolution}(t_u)$. At t_{u+1} the edge is removed and two new nodes s' and r' are inserted, which are the result of the crossover operation. The new nodes have the same set of links in the interactions graph as the original nodes, but their weights may change due to the new fitness evaluation. It is worth remarking that a crossover may lead to an increase in population, since the offspring do not necessarily replace their parents.

Step 5. Distributed continuous optimisation

In order to enable continuous evolution, the steps of fitness evaluation, transformation through genetic operators, and selection must be performed at regular intervals, such that new, potentially more suitable services can be generated, discovered and used instead of old ones. The system elements are then constantly being evaluated and modified to adapt to new situations. As opposed to traditional GAs and GP, there is no stop condition, as the system runs continuously.

8 REVISION: Updated Research Roadmap

This section is an addendum to D2.2.2 produced upon request from our reviewers during the First Project Review in Zurich, March 2007. Quoting from the review report:

> *BIONETS project encompasses evolution and adaptivity at both service as well as network levels. It is somehow vague at present (Section 7 in D222) how these two properties are realized at each level (for ex. How service/protocol evolution is realized in the BIONETS context? What are the critical parameters of a service/protocol that evolve and adapt?). You should provide clear roadmap that illustrates how these two properties are envisioned at both levels in BIONETS. Include your findings in a self-contained section in D222 or under Section 7 of D222.*

We must first of all recall that the scope of the present deliverable is restricted to *evolution*, as opposed to *adaptation*.

Evolution and *adaptation* are distinguished according to their biological counterpart: Evolution happens as a long-term consequence of small genome changes that are inherited from parents to offspring. *Species evolve* across generations, while *organisms adapt* during their lifetimes (see also Section 3.1).

In BIONETS, the "organisms" or "individuals" are protocol or service instances, while "species" are collections of protocols or services providing the same or similar functionality. Evolution in this context implies structural changes that are inherited from parent(s) to offspring. So we need a notion of "generation", which can be associated to forking a new instance of a service with parameters/code inherited from the previous instance. Short-term specific adaptation/learning that does not survive to the "next generation" would not qualify as evolution. Evolution is necessarily long-term, useful when the pre-established adaptive mechanisms are no longer sufficient.

Adaptive algorithms operate at short time scales, and are necessary for protocols and services to quickly react to changes in the environment or in network conditions. For instance TCP includes an adaptive algorithm for congestion control (recalling from Section 3.2). Currently such algorithms are designed by humans, and require considerable skills and effort. Evolving these algorithms automatically is certainly a challenge worth being investigated on its own. At first, however, BIONETS does not impose that all the evolved protocols and services must be adaptive. An interesting research challenge would be to verify whether adaptive algorithms emerge as an outcome of the evolution process. At the moment we are not able to demonstrate that, but during the course of the project we expect to gain more insights on it. Indeed, some related literature is available, for instance [39] on coupling evolution across generations with a lifetime learning process in order to produce individuals able to adapt to their environment. Recently [50], Genetic Programming (GP) was applied to evolve round-trip time estimators for TCP, which are shown to outperform the currently widespread estimator. These samples of related research could be used as seed for future work in evolving adaptive algorithms for BIONETS protocols and services.

Notwithstanding the relevance of adaptation, we must work first on the evolution problem alone, which is per se already significantly complex. The remainder of this section will therefore

focus on evolution only.

This discussion reveals one amendment or clarification concerning the service lifecycle described in BIONETS Deliverable D3.2.1 [99]. There, evolution including service transformation is part of the lifecycle, therefore one expects a service to evolve *during* its lifecycle. Instead, the idea was to incorporate mechanisms that create the *conditions* for evolution, like a service representation in the form of a genotype, together with the corresponding genetic operators such as crossover and mutation. So the lifecycle should clearly include some form of reproduction ability, but not necessarily evolution by itself. Clearly, it does not make sense to talk about incorporating evolution ability per se into service individuals, as evolution is an *emergent* property that can only be observed after several generations, thus several iterations of service transformations and selections. We plan to make this issue more clear and concrete in the upcoming Deliverable D3.2.2.

We divide evolution research in BIONETS into three topics: evolution of *configurations*, of *compositions*, and of *functionality*. These are associated with items *(i)*, *(ii)* and *(iv)* of the roadmap of Section 3.4, respectively. Note that item *(iii)* (evolution of *interfaces*) has been dropped from the list for the time being, since its priority and exact realization are still unclear. There are more than sufficient research issues to be explored even if we assume that the interfaces of protocols and services are not subject to evolution. The current state and potential contributions for each topic are outlined below.

8.1 Evolution of Configurations

One property of autonomics is *self-configuration*. Most protocols, even adaptive and self-organizing ones, typically have a list of parameters that must be tuned beforehand for a given target operation scenario. This is often done in an ad hoc way, due to the lack of guidelines on how to tune these parameters properly. No rules for tuning parameters are available, simply because, if they had been available, they would have been used to automatically tune them in software. Those parameters left open to manual configuration typically reveal areas where a clear procedure to obtain the optimum is unknown. An evolutionary approach to automatically find the best choice of parameters, and reconfigure the system dynamically would therefore represent an important contribution to the field of autonomics.

BIONETS has proposed to use epidemic forwarding to exchange *service genotypes* [31]. In its simplest form, a service genotype encodes a list of configuration parameters to be evolved. A selection mechanism operates at each node to select the best genotypes among those received via epidemic spread. This results in a form of online decentralized Genetic Algorithm. This idea was the seed for the BIONETS project, and its feasibility should be demonstrated during the project. An earlier investigation was conducted on the convergence properties of the resulting distributed evolutionary processes [32] (summarized in Section 6.2). More recently, we have applied this genotype dissemination approach to the evolution of parameters in the epidemic forwarding protocols themselves [2, 3].

The concept of evolving parameter configurations is generic and can be applied to protocols as well as to services. It suffices to specify the set of parameters in the genotype according to the

problem at hand. In practice however, the consortium has focused on SP1 protocols as case studies. This is because, on one hand, SP3 must focus on the evolution of compositions as described in Section 8.2, and on the other hand, back to SP1, evolving protocol implementations from scratch is difficult, so it is more realistic to start with their parameters only. Moreover, epidemic forwarding protocols, especially those in [2, 3], have the special property that not all the nodes in the network need to use exactly the same parameter values. A totally decentralized evolution process is feasible in this case. In a generic case, for either protocols or services, an evolution mechanism would have to consider the case where changing a parameter in one node (e.g. at the source node) would require changing a similar or complementary one in another node (e.g. the sink node). In case of highly distributed applications with multiple sources and sinks, the problem becomes extremely challenging. At the present stage we simplify the problem by focusing on cases where such interdependencies do not occur.

In the experiments in [2, 3] only two parameters were evolved: the forwarding probability and the maximum number of hops that a packet may traverse. Each node uses these parameters to configure a forwarding policy that decides whether a packet should be forwarded or not. It is shown that the system is able to automatically find optimum policies that achieve equivalent or better performance than when probabilistic forwarding is hand-tuned for a given scenario.

As a side outcome of [2, 3], a stable simulation package was developed for the dissemination of protocol genotypes. It runs on top of the Omnet++ simulator. The plans for the next stage are to integrate this package with the simulation tools currently being developed in WP1.3, and proceed with the experiments, improving on dynamic optimization aspects [68].

We also wish to explore the evolution of other protocols beyond epidemic dissemination. In the long run, we would like to move towards evolution of *strategies* rather than just parameters, applying upcoming results in evolutionary game theory expected from WP2.1.

8.2 Evolution of Compositions

This topic acknowledges the fact that not all system elements can be realistically evolved from scratch. Instead, one can compose "new" services out of elementary building blocks which are dynamically selected according to their fitness. Ideally, these building blocks should emerge from the evolutionary process, but in practice this is difficult to achieve (for instance, automatically defined functions and other modular approaches to GP have multiple limitations). One could then assume that at initialization time a number of atomic (i.e. indivisible, non-composite) building blocks or *service cells* are available to compose *service individuals* that contain the necessary functionality.

Of course it does not make sense to compose these individuals by choosing blocks randomly. In some early GP experiments with fraglets [130], we have used homologous recombination as a metaphor for exchanging only compatible modules. These modules were programmed by hand and annotated before being included in the initial population. Analogously, one assumes that a *service description* associated to each cell is available, as well as some composition logic to form new individuals. The composition logic could be initially based on simple type matching, and at

a later stage based on semantic information.

The task is then simply to compose the service according to such information, by selecting, for each needed functionality, one or more alternative implementations of the corresponding cells; each cell is then evaluated at runtime, and those that do not perform well enough should be discarded. Some initial protocol selection experiments were performed to validate this concept [104] (summarized in Section 7.5).

Note that the new service must be evaluated in *both functional* (Does it do what it is intended to do? e.g. does it deliver information properly? Does it calculate things properly? etc.) *and non-functional aspects* (e.g. performance: how well it performs its functions, how big the code is, etc.). Fitness evaluation relies on the *service description* to provide such functional and non-functional *expectations* on a service, which must then be contrasted with the actual observed behaviour and performance: the fitness is the difference between the expected and the observed values. For an unknown service (e.g. newly created) the expected values could be inherited from the parents' descriptions, or could be taken by comparison to similar services from the same species. The ability to perform such an evaluation in a reliable way is crucial to selecting a given service (or not) into the composition, and is subject of current research. This work is related to WP4, since security and trust issues are involved (How to make sure an evalution can be trusted? How to securely share previous evaluations of existing individuals, in order to minimize the need to constantly re-evaluate them?)

Although this task seems to represent no real challenge in terms of Evolutionary Computing (EC) itself (instead, it can be regarded as a degenerated form of EC) it does present some big challenges at the level of system interactions. First of all, it should be possible to switch modules in a smooth way, without disrupting system operation; or to benefit from multiple alternative modules in parallel, each contributing to provide part of the service; or to use them in a redundant way for fault tolerance purposes. Second, since individuals might be reused to compose other individuals, recursively, this generates a cascade fitness evaluation process, in which the fitness of an individual is determined by the fitness of its underlying cells, and perhaps also influenced by the fitness of neighboring individuals. The research challenge is then to provide robust dynamic composition mechanisms that can cope with potential side-effects of such cascade fitness evaluation process, leading to a feasible online composition mechanism. Starting from our initial formulation of this problem in [88] (summarized in Section 7.6), research is in progress in SP3 to define a graph representation and a robust fitness evaluation model that should make this composition approach possible.

Such composite systems could exhibit complex dynamics. A potential research direction here is to *co-evolution* [101] the branch of EC that studies the complex dynamics arising from the interplay among co-evolving populations. First of all, it is important to understand how different species (here represented by different services) evolve simultaneously, possibly influencing each other. The second step is then to devise optimum strategies that can make evolution more forward in a more efficient way in this context. Techniques for the analysis of co-evolutionary systems include evolutionary game theory and dynamical systems theory. An interaction with WP2.1 is envisaged to help making progress in this topic.

8.3 Evolution of Functionality

The aim of this topic is to optimize the internal *implementation* of a *service cell* as defined in Section 8.2. We assume that the modified cells keep the same interface and semantics of their parents, i.e., they can still be considered as (atomic) individuals that belongs to the same species, such that they can still be composed with others as described in the previous section. This is because we have dropped or at least postponed the evolution of interfaces issue (as pointed out at the beginning of Section 8) in order to make the problem tractable. This restricts what the system can really do in terms of truly creating new functionality, since an entirely new functionality would also require an entirely new interface. Still, this does not prevent a cell with fixed interface from evolving entirely new ways to achieve a given goal. This goal could change in time, if it is provided as input to the cell, and not as part of the interface (see also Section 7.2 for a related discussion).

If online evolution is a challenging topic, online *code* evolution is even more challenging, since the generated code must be executed in an operating environment, without the opportunity for isolated lab tests. Programs must therefore be resilient to harmful mutations and other perturbations of their code base, and must support full self-modification in order to evolve new functionality in an unbounded way. Living organisms are so far the best example of such intrinsic robustness, which artificial systems so far fail to match.

Note that there is a fundamental difference between robustness in living beings or in artificial life, and in software that implements a concrete service for a user. In the former case the fitness criterion is the survival of the individual, no matter whether it causes harm to others or not. While in the latter, services must remain faithful to their users even in the presence of disruptions; this means that functionality should be preserved with an acceptable level of performance. This is an extended fault tolerance property: the system should work even after parts of it modify themselves to achieve a next step in the evolution process.

We have identified artificial chemical computing as a promising programming model to achieve such online resilience, as multiple execution paths could co-exist and be regulated to reinforce the fittest ones dynamically. Our investigations started with our fraglets programming language, which was originally designed for the automated synthesis and evolution of network protocols [120]. During BIONETS we have performed some first experiments on a code regulation mechanism as an asynchronous selection mechanism for fraglet code [129]. These experiments involved very simple code still without any evolution mechanism.

The next two immediate steps currently in progress are: First of all, to enhance the fraglets language with generic computation properties such that it can be used to implement services in BIONETS, as well as to implement more complex protocols.

Second, to design and implement an evolution framework based on fraglets, such that fraglet programs presenting such robustness characteristics can be evolved. Although the language was originally designed with automatic programming in mind, in practice our experiments showed that it is not so easy. Fraglets have almost no syntactic constraints: any string is a fraglet, so recombining or mutating fraglets always results in a syntactically valid fraglet. Similar characteristics have been proposed in other languages targeted at GP [59, 116], with the same motivation: a syntax

that can be easily manipulated, always resulting in a viable program. It might appear tempting to use these languages instead of fraglets, however, their stack-based architecture does not easily lend them to a resilient execution model, since a stack is a natural single point of failure. They also lack the notion of light-weight mobile code fragments that can be easily transported between nodes; this notion present in fraglets provides a natural metaphor for a genotype exchange model that is close to biology.

In the case of fraglets, the molecular computing model implies that the output of a computation must be tagged such that it can be properly recognized as a substrate for the next reaction in the computing pathway towards a solution. Applying genetic operators to fraglets in a naive way simply breaks the execution path most of the time, resulting in malformed fraglets in terms of expected arguments (these are discarded when detected by the interpreter), or fraglets that are never used, leading to dead-ends in the reaction path, when no further computation takes place because of a lack of matching tags.

Execution path disruption is not an issue in other artificial chemistries such as [17]. In [17], instructions are drawn from a multiset and executed in random order. They show that evolution leads to highly reproducible output independent of execution order, a resilience property which we also aim at. The architecture is based on a register machine with a fixed number of registers. It is always possible to read input values from random registers, compute some function with them, and write the result to another random register. Therefore, execution path disruption can never occur. In fraglets the situation is different, since the amount of tags varies from one program to the other and crossover might result in new, removed or duplicated tags. Morever, the fraglet interpreter does not draw instructions in random order for execution, but chooses them among those which are either standalone transformations or reactions with matching tags.

Note that obtaining valid execution paths is not strictly required by the system, as the fitness evaluation and selection mechanism should be able to eliminate invalid individuals. Moreover, individuals could evolve parallel execution paths, such that disrupting one of them would still make execution possible. However, in order to make evolution move forward in reasonable time, the genetic operators must produce viable individuals with a high probability. It is well known in Evolutionary Computing that the genotype representation and the genetic operators applied to it play a key role in the performance of the evolutionary process. The genotype to phenotype mapping scheme is included in this representation problem, and it is well known in GP that indirect encodings like Cartesian GP [87] and Grammatical Evolution [95] can greatly help in obtaining viable individuals.

We are now working on an indirect encoding scheme for fraglets, based on Cartesian GP [87] in order to guarantee that each individual has at least one valid execution path. In Cartesian GP, genotypes can be represented as strings of integers which decode into a path through functional blocks analogous to logical gates in electronic circuits. This guarantees that the resulting program has a valid path from input to output. We are currently working on the representation in fraglets of a Cartesian GP genotype (basically a long string of integers represented as a fraglet) and the corresponding genetic operators also in fraglets, such that the fraglet code to be executed could be derived by fraglet rules that rewrite the genotype into the actual code.

We have also considered the use of Grammatical Evolution [95], another form of indirect encoding for GP, but have for the moment discarded it, since the main obstacle in obtaining valid fraglets by mutation and crossover is not the syntax (as there are hardly any syntax constraints in the language itself) but the semantics (production of the right tags at the right moment). Moreover it would probably not be straightforward to implement a full BNF grammar derivation engine (as required in Grammatical Evolution) in fraglets.

We start by evolving fraglets to solve a symbolic regression problem which is a typical benchmark in GP. Once this works, we plan to move to more realistic problems in the BIONETS scenario, such as evolving a controller given some sensor input, and later evolving distributed controllers for a sensor/actuator network. The current state of this research will be reported in more detail in D3.2.2.

8.4 Summary

Although the three topics outlined above are valid for protocols as well as for services, different BIONETS workpackages have focused on either one or the other, in order to make the problem tractable. Namely:

- **Evolution of Configurations** is now clearly linked to **SP1** and mapped to the evolution of **epidemic protocols**, currently work in progress with first results on [2, 3]. Main partners currently involved: CN and INRIA.
- **Evolution of Compositions** is a main theme within **SP3** and mapped to the evolution of **composite services**, where the main research issues are the representations for service cells and service individuals (currently a graph representation is being considered) and the impact of cascade fitness evaluations. Main partners currently involved: INRIA and TUB.
- **Evolution of Functionality** should backtrack to **SP2** for a year or so, where basic research on **fraglets** and the *evolution of chemical programs* should lead to a generic evolution framework where *self-evolving programs* take care of their own genetic operations in a resilient way. Afterwards, the plan is to show case studies in **SP3** on the evolution of distributed controllers for a sensor/actuator network, and in **SP1** on the epidemic spread of protocol code using a chemical diffusion model. Main partners currently involved: UBASEL and UNIHH.

We believe that these three orthogonal topics sufficiently cover the research landscape directly needed for the success of BIONETS. In a final stage, depending on the progress in each of these topics, an integration could be envisaged, but care should be taken not to lose focus in this effort, since the risk of considering too many topics simultaneously is that no concrete conclusions can be derived that help advancing the state of the art in online evolution.

9 Conclusion and Next Steps

Pervasive disconnected environments as those considered for BIONETS call for an autonomic computing and communication approach in order to obtain scalable self-managed systems. Self-evolution is an important property of these systems, without which they cannot be made fully autonomic.

Evolution in this context is an open-ended, long-term process in which the system is continuously undergoing self-optimisation, i.e., transforming itself to comply to significant changes in requirements or in the environment. Evolution goes beyond short-time reactive adaptations that are pre-programmed in the system and for which well-known techniques can be employed.

In an autonomic system, evolution is the result of a combined process involving transformation operations, fitness evaluation, selection with competitive pressure, and communication (for diffusion and deployment purposes). Contrary to traditional evolutionary computing, the distributed and on-line characteristics of self-evolving autonomic systems put communication at the centre of the evolution process: communication is essential to exchange information for transformation purposes (e.g. crossover of programs), which ensures the variability of the system; it is also essential to spread good service variants to where they are demanded; and it plays a key role in providing fitness evaluation feedback in a distributed way. This opens up new domains in evolutionary computing, which have remained so far almost unexplored.

In this document we have identified the main building blocks and research agenda of an evolutionary framework for the distributed on-line evolution of protocols and services in BIONETS. We divide the problem into two main parts: micro-evolution and macro-evolution. The micro-evolution part deals with the local small-scale evolutionary operations, while the macro-scale studies the global large-scale implications of the actions taken at micro-scale. The differences between the tools and methods for each part have been highlighted, and some first insights and results have been presented.

The next step is now to apply these insights to services in BIONETS WP3.2 and protocols in WP1.2. A performance evaluation of evolutionary protocols is intended within WP1.3. Another important issue is to elaborate on security, more specifically adaptive and evolutionary security mechanisms able to track an evolving platform. This is being done in WP4. Some of these activities are already in progress. For instance, the integration of evolution within the service lifecycle. Evolution operations are inherently part of the lifecycle of services in BIONETS: these services are constantly being evaluated and selected. Selection creates a competitive pressure that provides an incentive for the service to perform transformations that will improve its fitness, therefore its performance in given context. Selection also ensures robustness against failures or misbehaving services, as these would receive a low fitness value and be eliminated from the system. We believe this provides a new service paradigm that can change the way services are deployed and managed, and that can lead to truly autonomic services.

References

[1] R. Albert and A. L. Barabási. *"Statistical mechanics of complex networks"*. Reviews of Modern Physics, 74:47–97, Jan. 2002.

[2] S. Alouf, I. Carreras, D. Miorandi, and G. Neglia. *"Embedding Evolution in Epidemic-Style Forwarding"*, 2007. submitted to IEEE MASS 2007.

[3] S. Alouf, I. Carreras, D. Miorandi, and G. Neglia. *"Evolutionary Epidemic Routing"*. Technical Report 00130803, INRIA RR, 2007.

[4] T. Alpcan, T. Başar, R. Srikant, and E. Altman. *"CDMA uplink power control as a noncooperative game"*. Wireless Networks, pages 659–670, 2002.

[5] E. Altman, T. Basar, T. Jimenez, and N. Shimkin. *"Routing into two parallel links: Game-Theoretic Distributed Algorithms"*. Journal of Parallel and Distributed Computing, Special Issue on "Routing in Computer and Communication Networks", 61(9):1367–1381, September 2001.

[6] B. Andersson, P. Svensson, P. Nordin, and M. Nordahl. *"On-line Evolution of Control for a Four-Legged Robot Using Genetic Programming"*. In Real-World Applications of Evolutionary Computing – EvoWorkshops 2000, Springer LNCS 1803, pages 319–326, Edinburgh, Scotland, April 2000.

[7] S. G. Araújo, A. C. P. Pedroza, and A. C. Mesquita. *"Evolutionary Synthesis of Communication Protocols"*. 10th International Conference on Telecommunications (ICT 2003), 2:986–993, February-March 2003.

[8] C. S.-O. Attolini. *"From Molecular Systems to Simple Cells: A Study of the Genotype-Phenotype Map"*. PhD dissertation, University of Vienna, Austria, 2005.

[9] R. Axelrod. *"The Evolution of Cooperation"*. New York: Basic Books, 1984.

[10] P. Bak, C. Tang, and K. Wiesenfeld. *"Self-organized criticality: an explanation of the $\frac{1}{f}$ noise"*. Phys. Rev. Lett., 59:381–384, 1987.

[11] J.-P. Banâtre, P. Fradet, and Y. Radenac. *"A Generalized Higher-Order Chemical Computation Model with Infinite and Hybrid Multisets"*. In 1st International Workshop on New Developments in Computational Models (DCM'05), pages 5–14, 2005. To appear in ENTCS (Elsevier).

[12] J.-P. Banâtre, P. Fradet, and Y. Radenac. *"Generalized Multisets for Chemical Programming"*. Research Report RR-5743, INRIA, November 2005.

[13] J.-P. Banâtre, P. Fradet, and Y. Radenac. *"Towards Grid Chemical Coordination"*. In Proceedings of Symposium on Applied Computing (SAC), 2006. (short paper).

[14] J.-P. Banâtre and D. L. Métayer. *"Programming by Multiset Transformation"*. CACM, 1993.

[15] W. Banzhaf. *"Artificial Regulatory Networks and Genetic Programming"*. In Genetic Programming - Theory and Applications, R. Riolo, B. Worzel (Eds.), chapter 4, pages 43–61. Kluwer Academic, Boston, MA, 2003.

[16] W. Banzhaf. *"On Evolutionary Design, Embodiment and Artificial Regulatory Networks"*. In Embodied Artificial Intelligence, F. Iida, R. Pfeifer, L. Steels and Y. Kuniyoshi (Eds.), pages 284–292. Springer, Berlin, LNAI 3139, 2004.

[17] W. Banzhaf and C. Lasarczyk. *"Genetic Programming of an Algorithmic Chemistry"*. In Genetic Programming Theory and Practice II, O'Reilly et al. (Eds.), volume 8, chapter 11, pages 175–190. Kluwer/Springer, 2004.

[18] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *"Genetic Programming, An Introduction"*. ISBN 155860510X. Morgan Kaufmann Publishers, Inc., 1998.

[19] J.-P. Banâtre, P. Fradet, and Y. Radenac. *"Principles of Chemical Programming"*, June 2004. Fifth International Workshop on Rule-Based Programming (RULE'04).

[20] J.-P. Banâtre and D. L. Métayer. *"A new computational model and its discipline of programming"*, September 1986. Technical Report RR0566, INRIA.

[21] J.-P. Banâtre and D. L. Métayer. *"Gamma and the Chemical Reaction Model"*. Internal Publication PI-984, INRIA, February 1996.

[22] J.-P. Banâtre, Y. Radenac, and P. Fradet. *"Chemical Specification of Autonomic Systems"*. In Proc 13th International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE'04), pages 72–79, July 2004.

[23] A.-L. Barabasi and Z. N. Oltvai. *"Network Biology: Understanding the Cell's Functional Organization"*. Nature Rev., pages 101–113, Feb. 2004.

[24] T. C. Belding. *"The Distributed Genetic Algorithm Revisited"*. In Proc. Sixth International Conference on Genetic Algorithms, pages 114–121, San Francisco, CA, USA, 1995. Morgan Kaufmann.

[25] H. G. Beyer and H. P. Schwefel. *"Evolution Strategies: A Comprehensive Introduction"*. Journal of Natural Computing, 1(1):3–52, 2002.

[26] *"Bio-Net: Bio-Networking Architecture Project"*. http://netresearch.ics.uci.edu/bionet/.

[27] A. Cabrales. *"Stochastic Replicator Dynamics"*. International Economic Review, Department of Economics, University of Pennsylvania and Osaka University Institute of Social and Economic Research Association, 41(2):451–481, May 2000.

[28] C. S. Calude and G. Paun. *"Computing with Cells and Atoms: An Introduction to Quantum, DNA and Membrane Computing"*. Taylor & Francis, 2001.

[29] I. Carreras, I. Chlamtac, C. K. Francesco De Pellegrini and, D. Miorandi, and H. Woesner. *"A Biological Approach to Autonomic Communication Systems"*. In Proc. Converging Science, Trento, Italy, 2004.

[30] I. Carreras, I. Chlamtac, F. D. Pellegrini, and D. Miorandi. *"BIONETS: Bio-Inspired Networking for Pervasive Communication Environments"*. IEEE Transactions on Vehicular Technology, 2006. in press.

[31] I. Carreras, I. Chlamtac, H. Woesner, and C. Kiraly. *"BIONETS: BIO-inspired Next generaTion networkS"*. In Proc. 1st Workshop on Autonomic Communication (WAC), Springer LNCS 3457, pages 245–252, Berlin, Germany, October 2004.

[32] I. Carreras, F. De Pellegrini, D. Miorandi, and H. Woesner. *"Service Evolution in a Nomadic Wireless Environment"*. In Proc. 2nd Workshop on Autonomic Communication (WAC), pages 29–40, Athens, Greece, October 2005.

[33] I. Carreras, F. De Pellegrini, D. Miorandi, and H. Woesner. *"Service evolution in a nomadic wireless environment"*. Technical Report, CREATE-NET, 2005.

[34] I. Carreras, D. Miorandi, and I. Chlamtac. *"A Framework for Opportunistic Forwarding in Disconnected Networks"*. In Proc. of MOBIQUITOUS, San Jose, CA, 2006.

[35] A. Chaintreau, P. Jui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. *"Impact of Human Mobility on the Design of Opportunistic Forwarding Algorithms"*. In Proc. of IEEE INFOCOM, Barcelona, Spain, 2006.

[36] V. Chickarmane, A. Nadim, A. Ray, and H. M. Sauro. *"A p53 Oscillator Model of DNA Break Repair Control"*, June 2006. Quantitative Biology.

[37] W. K. Ching. *"A note on the convergence of asynchronous greedy algorithm with relaxation in multiclass queuing environment"*. IEEE Communication Letters, 3:34–36, 1999.

[38] J. Coutaz, J. Crowley, S. Dobson, and D. Garlan. *"Context is Key"*. Communications of the ACM, 48(3), March 2005.

[39] T. Damoulas, I. Cos-Aguilera, G. Hayes, and T. Taylor. *"Valency for Adapative Homeostatic Agents: Relating Evolution and Learning"*. In Proceedings of the 8th European Conference on Artificial Life (ECAL 2005), Cantebury, UK, September 2005.

[40] D. Dhar. *"Studying Self-organized criticality with exactly solved models"*. 1999.

[41] Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, G. E. Kaiser, and D. Phung. *"A Control Theory Foundation for Self-Managing Computing Systems"*. IEEE Journal on Selected Areas in Communications (JSAC), 23(12):2213–2222, Dec. 2005.

[42] *"Digital Business Ecosystem (DBE) Project"*. http://www.digital-ecosystem.org.

[43] P. Dittrich. *"Chemical Computing"*. In Unconventional Programming Paradigms (UPP 2004), Springer LNCS 3566, pages 19–32, 2005.

[44] P. Dittrich, J. Ziegler, and W. Banzhaf. *"Artificial Chemistries – A Review"*. Artificial Life, 7(3):225–275, 2001.

[45] O. Dousse, M. Franceschetti, and P. Thiran. *"On the throughput scaling of wireless relay networks"*. IEEE Trans. Inf. Th. - IEEE/ACM Trans. on Netw., joint issue, 2006. In press.

[46] A. Eiben. *"Evolutionary Computing and Autonomic Computing: Shared Problems, Shared Solutions?"*. In International Workshop on Self-Star Properties in Complex Information Systems, LNCS 3460 (2005), pages 36–48, Bertinoro (Forli), Italy, May-June 2004.

[47] A. Eiben and J. Smith. *"Introduction to Evolutionary Computing"*. Springer, 2003.

[48] M. Eigen and P. Schuster. *"The Hypercycle: A Principle of Natural Self-Organization"*. Springer, Berlin, 1979.

[49] J. M. Estévez-Tapiador, P. García-Teodoro, and J. E. Díaz-Verdejo. *"Measuring normality in HTTP traffic for anomaly-based intrusion detection"*. Computer Networks, 45(2):175–193, 2004.

[50] C. Fillon and A. Bartoli. *"Multi-Objective Genetic Programming for Improving the Performance of TCP"*. In Ebner et al., editor, Proceedings of the 10th European Conference on Genetic Programming (EuroGP 2007), volume 4445 of LNCS, pages 170–180, Valencia, Spain, April 2007.

[51] L. J. Fogel. *"Artificial Intelligence through simulated evolution"*. John Wiley & Sons Inc, 1966.

[52] *"Fraglets Home Page"*. http://www.fraglets.net/.

[53] R. Frigg. *"Self-Organized Criticality – What It is and what it isn't"*. Technical Report CPNSS-19/02, London School of Economics, 2002.

[54] R. Groenevelt. *"Stochastic Models in Mobile Ad Hoc Networks"*. PhD thesis, INRIA, 2005.

[55] M. Grossglauser and D. Tse. *"Mobility Increases the Capacity of Ad Hoc Wireless Networks"*. IEEE/ACM Trans. on Netw., 10(4):477–486, Aug. 2002.

[56] P. Gupta and P. R. Kumar. *"The Capacity of Wireless Networks"*. IEEE Trans. on Inf. Th., 46(2):388–404, Mar. 2000.

[57] A. Hjelmfelt, E. Weinberger, and J. Ross. *"Chemical Implementation of Neural Networks and Turing Machines"*. Proceedings of the National Academy of Sciences of the USA, 88:10983–10987, 1991.

[58] J. Hofbauer and K. Sigmund. *"Evolutionary game dynamics"*. American Mathematical Society, 40(4):479–519, 2003.

[59] K. Holladay, K. Robbins, and J. von Ronne. *"FIFTH: A Stack Based GP Language for Vector Processing"*. In Ebner et al., editor, Proceedings of the 10th European Conference on Genetic Programming (EuroGP 2007), volume 4445 of LNCS, pages 102–113, Valencia, Spain, April 2007.

[60] J. Holland. *"Adaptation in Natural and Artificial Systems"*. MIT Press, 1992. First Edition 1975.

[61] A. I. Houston and J. M. McNamara. *"Evolutionarily stable strategies in the repeated hawk-dove game"*. Behavioral Ecology, pages 219–227, 1991.

[62] P. K. I. Stamatopoulou, M. Gheorghe. *"Modelling of dynamic configuration of biology-inspired multi-agent systems with communicating X-machines and population P systems"*. In Fifth Workshop on Membrane Computing (WMC5), Milan, Italy, 2004.

[63] *"IEEE 802.11b PHY and MAC: Supplement to 802.11-1999, Wireless LAN MAC and PHY specifications: Higher Speed Physical Layer (PHY) extension in the 2.4 GHz band"*, Sep 1999.

[64] V. Jacobson and M. J. Karels. *"Congestion Avoidance and Control"*. In Proc. of ACM SIGCOMM, Stanford, CA, 1988.

[65] S. Jain, K. Fall, and R. Patra. *"Routing in a delay tolerant network"*. In Proc. of ACM SIGCOMM, pages 145–158, Portland, OR, USA, 2004. ACM Press.

[66] S. Jain and S. Krishna. *"A Model for the Emergence of Cooperation, Interdependence and Structure in Evolving Networks"*. Proc. of Nat. Acad. of Scien. of Unit. Stat. of Amer. (PNAS), 98:543–547, Jan. 2001.

[67] S. Jain and S. Krishna. *"Graph Theory and the Evolution of Autocatalytic Networks"*. In S. Bornholdt and H. G. Schuster, editors, Handbook of Graphs and Networks, pages 355–395. J. Wiley and Sons, New York, NY, 2003.

[68] Y. Jin and J. Branke. *"Evolutionary Optimization in Uncertain Environments - A Survey"*. IEEE Transactions on Evolutionary Computation, 9(3):303–317, June 2005.

[69] J. Keeney, K. Carey, D. Lewis, D. O'Sullivan, and V. Wade. *"Ontology-based Semantics for Composable of Autonomic Elements"*. In Proc. of Workshop on AI in Autonomic Communications at 19th International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, July-August 2005.

[70] J. O. Kephart and D. M. Chess. *"The Vision of Autonomic Computing"*. IEEE Computer Magazine, 36(1):41–50, January 2003.

[71] H. K. Khalil. *"Nonlinear Systems"*. Prentice-Hall, Upper Saddle River, NJ, 2002.

[72] H. Kitano (Ed.). *"Foundations of Systems Biology"*. MIT Press, 2001.

[73] J. Koza. *"Genetic Programming: On the Programming of Computers by Means of Natural Selection"*. MIT Press, 1992.

[74] J. Koza, M. Keane, M. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *"Genetic Programming IV : Routine Human-Competitive Machine Intelligence"*. Springer, July 2003.

[75] W. B. Langdon and R. Poli. *"Foundations of Genetic Programming"*. Springer, 2002.

[76] W. B. Langdon and R. Poli. *"The Halting Probability in von Neumann Architectures"*. In P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt, editors, Proceedings of the 9th European Conference on Genetic Programming, Springer LNCS 3905, pages 225–237, Budapest, Hungary, April 2006.

[77] J.-Y. Le Boudec and M. Vojnović. *"Perfect simulation and stationarity of a class of mobility models"*. In Proc. of IEEE INFOCOM, Miami, FL, 2005.

[78] A. Leier, P. D. Kuo, W. Banzhaf, and K. Burrage. *"Evolving Noisy Oscillatory Dynamics in Genetic Regulatory Networks"*. In Proc. 9th European Conference on Genetic Programming, P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt (Eds.) Springer LNCS 3905, pages 290–299, Budapest, Hungary, April 2006.

[79] L. Li, D. Alderson, R. Tanaka, J. C. Doyle, and W. Willinger. *"Towards a Theory of Scale-Free Graphs: Definition, Properties, and Implications (Extended Version)"*. 2005.

[80] Z. Manna and R. Waldinger. *"Fundamentals of Deductive Program Synthesis"*. IEEE Transactions on Software Engineering, 18(8):674 – 704, August 1992.

[81] N. Matsumaru, F. Centler, P. S. di Fenizio, and P. Dittrich. *"Chemical Organization Theory as a Theoretical Base for Chemical Computing"*. International Journal of Unconventional Computing (in print), 2006. earlier version in: Workshop on Unconventional Computing, p. 71-82, Luniver Press, Beckington.

[82] N. Matsumaru, P. S. di Fenizio, F. Centler, and P. Dittrich. *"On the Evolution of Chemical Organizations"*. In Proc. 7th German Workshop on Artificial Life, pages 135–146, 2006.

[83] J. M. McNamara. *"The policy which maximizes long-term survival of an animal faced with the risks of starvation and predation"*. Advances of Applied Probability, 22:295–308, 1990.

[84] J. M. McNamara, S. Merad, and E. J. Collins. *"The hawk-dove game as an average cost problem"*. Advances of Applied Probability, 23:667–682, 1991.

[85] R. Meester and C. Quant. *"Connections Between 'Self-Organized' and 'Classical' Criticality"*. Markov Processes Relat. Fields, 11:355–370, 2005.

[86] V. Mhatre and C. Rosenberg. *"Energy and cost optimizations in wireless sensor networks: A survey"*. In in the 25th Anniversary of GERAD. Kluwer Academic Publishers, January 2004.

[87] J. F. Miller and P. Thomson. *"Cartesian Genetic Programming"*. In R. P. et al., editor, Genetic Programming, Proceedings of EuroGP'2000, volume 1802 of LNCS, pages 121–132, Edinburgh, April 2000.

[88] D. Miorandi, L. Yamamoto, and P. Dini. *"Service Evolution in Bio-Inspired Communication Systems"*. International Conference on Self-Organization and Autonomous Systems in Computing and Communications (SOAS 2006), ITSSA journal (International Transactions on Systems Science and Applications), 2(1):51–60, September 2006.

[89] D. Miorandi (editor). *"Requirements and Architectural Principles: Application scenario analysis, network architecture requirements and high-level specification"*. BIONETS (IST-2004-2.3.4 FP6-027748) Deliverable (D1.1.1), June 2006.

[90] N. Mori and K. Matsumoto. *"Adaptation to a Dynamical Environment by Means of the Environment Identifying Genetic Algorithm"*. In Congress on Evolutionary Computation (CEC '03), volume 3, pages 1626–1631, December 2003.

[91] T. Nakano and T. Suda. *"Adaptive and Evolvable Network Services"*. In Proc. Genetic and Evolutionary Computation Conference (GECCO-2004), Springer LNCS 3102, pages 151–162, 2004.

[92] T. Nakano and T. Suda. *"Self-Organizing Network Services with Evolutionary Adaptation"*. IEEE Transactions on Neural Networks, 16(5):1269–1278, September 2005.

[93] S. Nolfi and D. Floreano. *"Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines"*. MIT Press, 2000.

[94] *"OMNeT++ Simulator"*. `http://www.omnetpp.org`.

[95] M. O'Neill and C. Ryan. *"Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language"*. Kluwer Academic Publishers, 2003.

[96] G. Paun. *"Computing with Membranes"*. Journal of Computer and System Sciences, 61(1):108–143, 2000.

[97] F. D. Pellegrini (editor). *"Disappearing Network Infrastructure and Design: Functionality and Challenges"*. BIONETS (IST-2004-2.3.4 FP6-027748) Deliverable (D1.2.1), June 2006.

[98] M. D. Penrose. *"The longest edge of the random minimal spanning tree"*. The Annals of Appl. Prob., 7(2):340–361, 1997.

[99] H. Pfeffer, D. Linner, S. Steglich, and I. R. (editors). *"Specification of Service Life-Cycle"*. BIONETS (IST-2004-2.3.4 FP6-027748) Deliverable (D3.2.1), February 2007.

[100] R. Poli. *"Parallel Distributed Genetic Programming"*. In D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization, Advanced Topics in Computer Science, chapter 27, pages 403–431. McGraw-Hill, Maidenhead, Berkshire, England, 1999.

[101] E. Popovici and K. de Jong. *"The dynamics of the besst individuals in co-evolution"*. Natural Computing, 5:229–255, 2006.

[102] K. Prestwich. *"Game Theory"*. available at `www.holycross.edu/departments/biology/kprestwi/behavior/ESS/`.

[103] R. L. Probert and K. Saleh. *"Synthesis of Communication Protocols: Survey and Assessment"*. IEEE Transactions on Computers, 40(4):468 – 476, April 1991.

[104] J. J. Ramos-Munoz, L. Yamamoto, and C. Tschudin. *"Serial Experiments Online"*. Technical Report, Computer Science Department, University of Basel, Switzerland, 2006.

[105] J. J. Ramos-Muñoz and J. M. Lopez-Soler. *"Low Delay Multiflow Block Interleavers for Real-Time Audio Streaming"*. Lecture Notes in Computer Science, 3420:909 – 916, January 2005.

[106] I. Rechenberg. *"Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution"*. PhD thesis, Stuttgart: Fromman-Holzboog, 1973.

[107] J. Robinson, I. Wakeman, and T. Owen. *"Scooby: Middleware for Service Composition in Pervasive Computing"*. In Proceedings of the 2nd workshop on Middleware for Pervasive and Ad-hoc Computing, ACM International Conference Proceeding Series; Vol. 77, pages 161–166, Toronto, Ontario, Canada, 2004.

[108] O. G. Selfridge. *"Learning and Education: A Continuing Frontier for AI"*. IEEE Intelligent Systems, 21(3), May-June 2006.

[109] T. Senivongse and I. Utting. *"A model for evolution of services in distributed systems"*. In Spaniol Schill, Mittasch and Popien, editors, Distributed Platforms, pages 373–385. Chapman and Hall, January 1996.

[110] N. Sharples. *"Evolutionary Approaches to Adaptive Protocol Design"*. PhD dissertation, University of Sussex, UK, August 2001.

[111] N. Sharples and I. Wakeman. *"Protocol construction using genetic search techniques"*. In Real-World Applications of Evolutionary Computing – EvoWorkshops 2000, Springer LNCS 1803, Edinburgh, Scotland, April 2000.

[112] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Perez-Uribe, and A. Stauffer. *"A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems"*. IEEE Transactions on Evolutionary Computation, 1(1), April 1997.

[113] M. Smith. *"Game Theory and the Evolution of Fighting"*. In John Maynard Smith, On Evolution (Edinburgh: Edinburgh University Press), pages 8–28, May 1972.

[114] M. Smith. *"Evolution and the Theory of Games"*. Cambridge University Press, Cambridge, UK, 1982.

[115] C. Song, S. Havlin, and H. A. Makse. *"Self-similarity of complex networks"*. Nature, (7024):392–395, Jan. 2005.

[116] L. Spector and K. Stoffel. *"Automatic Generation of Adaptive Programs"*. In P. Maes, M. J. Mataric, J.-A. Meyer, J. Pollack, and S. W. Wilson, editors, Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior: From animals to animats 4, pages 476–483, Cape Code, USA, 9-13 September 1996. MIT Press.

[117] L. Steels. *"Emergent functionality in robotic agents through on-line evolution"*. In Proceedings of AlifeIV, Cambridge, MIT Press, 1994.

[118] J. Suzuki and T. Suda. *"A Middleware Platform for a Biologically Inspired Network Architecture Supporting Autonomous and Adaptive Applications"*. IEEE Journal on Selected Areas in Communications (JSAC), 23(2):249–260, February 2005.

[119] A. Syropoulos. *"On P systems and distributed computing"*. In Fifth Workshop on Membrane Computing (WMC5), Milan, Italy, 2004.

[120] C. Tschudin. *"Fraglets – A Metabolistic Execution Model for Communication Protocols"*. In Proc. 2nd Annual Symposium on Autonomous Intelligent Networks and Systems (AINS), Menlo Park, USA, July 2003.

[121] C. Tschudin and L. Yamamoto. *"A Metabolic Approach to Protocol Resilience"*. In Proc. 1st Workshop on Autonomic Communication (WAC), Springer LNCS 3457, pages 190–205, Berlin, Germany, October 2004.

[122] C. Tschudin and L. Yamamoto. *"Harnessing Self-modifying Code for Resilient Software"*. In Second NASA GSFC/IEEE Workshop on Radical Agent Concepts (WRAC), Greenbelt, MD, USA, September 2005. To appear in LNCS/LNAI 3825.

[123] C. Tschudin and L. Yamamoto. *"Self-Evolving Network Software"*. Praxis der Informationsverarbeitung und Kommunikation (PIK Magazine), 28(4):206–210, December 2005.

[124] T. L. Vincent and T. L. S. Vincent. *"Evolution and control system design"*. IEEE Control Systems Magazine, 20(5):20–35, October 2000.

[125] S. Wasserman and K. Faust. *"Social Network Analysis: Methods and Applications"*. Cambridge University Press, 1994.

[126] M. Weiser. *"The computer for the 21st century"*. ACM Mob. Comput. Commun. Rev., 3(3):3–11, 1999.

[127] M. A. Wermelinger. *"Specification of Software Architecture Reconfiguration"*. PhD dissertation, Universidade Nova de Lisboa, Lisbon, Portugal, September 1999.

[128] F. Xue and P. R. Kumar. *"Scaling Laws for Ad Hoc Wireless Networks: an information theoretic approach"*. NOW Publishers, 2006.

[129] L. Yamamoto. *"Code Regulation in Open Ended Evolution"*. In Ebner et al., editor, Proceedings of the 10th European Conference on Genetic Programming (EuroGP 2007), volume 4445 of LNCS, pages 271–280, Valencia, Spain, April 2007. poster presentation.

[130] L. Yamamoto and C. Tschudin. *"Experiments on the Automatic Evolution of Protocols using Genetic Programming"*. In Proc. 2nd Workshop on Autonomic Communication (WAC), pages 13–28, Athens, Greece, October 2005.

[131] L. Yamamoto and C. Tschudin. *"Genetic Evolution of Protocol Implementations and Configurations"*. In IFIP/IEEE International Workshop on Self-Managed Systems and Services (SelfMan 2005), Nice, France, May 2005.

[132] D. M. Yellin. *"Competitive algorithms for the dynamic selection of component implementations"*. IBM Systems Journal, 42(1):85 – 97, January 2003.

[133] J. Yoon, M. Liu, and B. Noble. *"Random Waypoint Considered Harmful"*. In Proc. of IEEE INFOCOM, San Francisco, CA, 2003.

[134] J. Ziegler and W. Banzhaf. *"Evolving Control Metabolisms for a Robot"*. Artificial Life, 7(2):171–190, 2001.