DOI: http://dx.doi.org/10.7551/978-0-262-33027-5-ch108

A Metamodel for the Evolution of Evolution

Paul Andrews and Susan Stepney

Department of Computer Science, University of York, UK
susan.stepney@york.ac.uk

Abstract

The ability of evolution to influence its own course in micro-organisms such as bacteria is a desirable property for computational systems. As a step towards exploiting this, we define a metamodel for the Evolution of Evolution. The metamodel is based on the concepts of structure and process, which are embodied together as the Machine. By describing different types Machines and structures we can capture a flexible metamodel to form the underpinnings for a new generation of evolutionary algorithms.

Introduction

The concept of *Evolution of Evolution* (EvoEvo) stems from the idea that evolution within organisms is able to influence its own course, and is derived from the observation that the molecular systems involved in evolutionary processes have themselves resulted from past evolution (Beslon et al., 2014). This has resulted in phenomena such as the evolution of robustness, mutation operators, mutation rates, and evolvability. Given the timescale of evolution, EvoEvo phenomena are most evident in bacteria and viruses, which can adapt rapidly and efficiently to changing environments by accelerating their evolution.

Inspired by the evolutionary dynamics observed in bacteria and viruses, the EU funded FP7 project, EvoEvo[1], aims to harness EvoEvo phenomena to create evolvable software systems. Specifically, the aim is to develop new algorithmic approaches to address dynamically changing *open-ended* engineering problems in which solutions adapt to previously unknown conditions and potentially evolve new types of solutions. To capture the relevant EvoEvo phenomena, the evolutionary dynamics of experimental bacterial and viral systems are being studied to inform computational models and simulations (based on Knibbe et al. (2007); Crombach and Hogeweg (2007)). In turn, these models form the inspiration for a *computational framework* that captures EvoEvo processes and can be instantiated as evolutionary algorithms for a given engineering problem. It is the design of this framework that is the basis for this paper.

[1] http://evoevo.liris.cnrs.fr/

We present here a *metamodel* for EvoEvo that encapsulates and abstracts the relevant analogies of biological EvoEvo processes, specifically those observed in bacteria. Whilst a model of a system provides an abstract language for relevant concepts, a metamodel provides the language for writing a model by specifying the *kinds of things* that might be present in the model (Kleppe et al., 2003, Chapter 8). In Andrews et al. (2011) we discuss in depth the concept of metamodels in the context of modelling complex systems and developing bio-inspired engineering systems. A useful example of a metamodel is given in that paper: a metamodel of agent-based modelling might include concepts of Agent, Rule and Emergent. Based on that metamodel, a model of ant pheromone trails would then include an instance of Agent, the Ant, and an instance of Emergent, the Trail.

So, different, separate models that explore EvoEvo concepts can be instances of the same metamodel, and this EvoEvo metamodel can establish the core components of the aforementioned EvoEvo computational framework. For this framework to be successfully used by third parties to instantiate their own EvoEvo-based algorithms, having the engineering rigour of an explicitly defined metamodel is a must.

It is worth noting that there are any number of different EvoEvo metamodels that could defined. The approach taken here is based on the requirement that the metamodel not only provides a language that can capture EvoEvo concepts, but has a natural analogy to computation so that it can be easily translated into the eventual computational framework. Before introducing the full EvoEvo metamodel, we first describe a more general Machine metamodel that fulfils this computational analogy requirement, and therefore forms the building blocks of the EvoEvo metamodel itself.

Machine Metamodel

At the most abstract level, any model of a biological system or concept such as EvoEvo can be considered in terms of structures (e.g. DNA) and processes (e.g. evolution via natural selection) that describe how these structures change through time and space. Further, we introduce the notion

Paul Andrews, Susan Stepney (2015) A Metamodel for the Evolution of Evolution. Proceedings of the European Conference on Artificial Life 2015, pp. 621-628

of a structure that implements a process (e.g. an enzyme). Here, we call this reified processes a *Machine*.

The Machine abstraction, informally conceptualised in Figure 1, provides us with a flexible language that is used to define our EvoEvo metamodel. A Machine is structure that implements a process that receives as input structures and energy, and returns as output (potentially) modified structures and (abstract) energy. Figure 1 also shows how Machines can form networks where inputs to one Machine are outputs from another. This allows us to compose a model of a system in which structures are subject to continual change via any number of processes that are driven by energy.

More formally, we can capture the Machine concepts using a class diagram, shown in Figure 2. This describes a Machine in terms of its relationship to Structure, Process, Symbol, and Energy.

Structure: composed from an ordering of Symbols. Does not itself posses internal behaviour or state (other than its own existence).

Symbol: a member of a given Alphabet that forms the building block of a Structure.

Alphabet: a set of possible Symbols. All Symbols that make up a Structure will be from the same Alphabet.

Process: acts upon Structures, potentially transforming them. Driven by Energy.

Energy: a quantity that drives the operation of Processes.

Machine: inherits from Structure and Process and so is an instance of both; a concrete Structure that reifies and implements a Process. As a Process, it can act upon other instances of Structure. It can store Energy and contain state.

There are three components to the Machine. First, as we have seen, the Machine extends the concept of Process, which means it exhibits some kind of behaviour. This behaviour can modify Structures that are the inputs to (and subsequent outputs of) the Machine. Second, a Machine can have state, which provides a memory to the Machine. One consequence of this is the ability to store parameters that shape the dynamics of the Machine's behaviour. Lastly, the Machine can (but is not required to) store energy that is used to drive the Machine's behaviour. In the absence of stored energy, the Machine's energy input must be sufficient to drive its behaviour. The Machine is not permitted to modify its own behaviour (Machines do not self-modify), but they can update their state.

The Machine also extends the concept of Structure. Hence a Machine (as Structure) can be passed to another Machine (as Process), allowing for its modification. But what does it mean for a machine to be data? Take for example an enzyme. The enzyme has a behaviour (catalyses a reaction) that can modify metabolites (structures). The enzyme itself, however, could be considered a Structure that was the output of a chemical reaction (Machine) that created it. Hence the same enzyme is either a Structure or Process *depending on the context*.

We can form an aggregate machine if the Structure/s out-

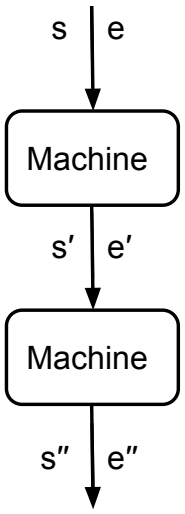

Figure 1: Machine takes structure, **s**, and energy, **e**, as inputs and returns potentially modified structures, **s**′, and energy, **e**′, as output. Networks of machines can form where outputs from one Machine are the inputs to the next.

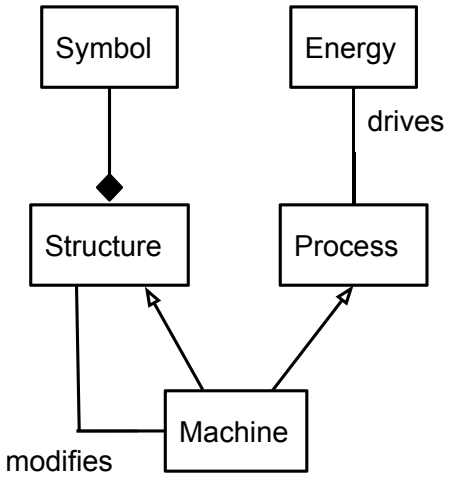

Figure 2: Class diagram showing the relationship between Machine, Structure, Process, Symbol, and Energy. Machine inherits from both Process and Structure, and modifies instances of Structure. Process is driven by Energy. Structures are composed of Symbols from an Alphabet (not shown).

Paul Andrews, Susan Stepney (2015) A Metamodel for the Evolution of Evolution. Proceedings of the European Conference on Artificial Life 2015, pp. 621-628

put from one machine match the input expected for another (and the energy outputs and inputs are also satisfied). Figure 3 demonstrates the case for two machines M1 and M2 that can also be viewed as a single machine M3. Depending on the model that implements the Machine metamodel concepts, the ability to aggregate at some level of abstraction could be useful. For example a metabolic pathway could be viewed both as a series of enzyme machines or a single pathway machine. How this aggregation is handled will be specific to the implementing model, however it is not inconceivable that some kind of observer process (machine) would be involved.

Energy, Space and Time

Energy in the Machine metamodel draws heavily from that given by Hoverd and Stepney (2011). Energy is a combined generalised resource flux (e.g. think nutrients and sunlight). Importantly it is a limited resource, and the flux can be used, stored or simply ignored in which case it dissipates and it not used. There are three parts to the energy metatmodel:

Flux: represents a flow of energy from outside the modelled system. It can have a particular temporal pattern e.g. high level during day, but lower level during night.

Store: represents the component's ability to store energy.

Demand: represents a demand for an amount of energy from a model component.

A Machine can be both a Store and Demand for energy.

The metamodel also contains the concept of Space. Spaces are containers within which Machines and other Structures are located. Importantly, a Space provides a locality for these Machines and Structures, which defines the Machines' connectivity. For example the Space might define distances to other Machines/Structures or filters that describe what other Machines/Structures can be seen and interacted with. In addition, Time defines how this ordering changes. Thus together Space and Time determine how machines interact (when and with what), which, along with the machine behaviour, gives the machine network dynamic

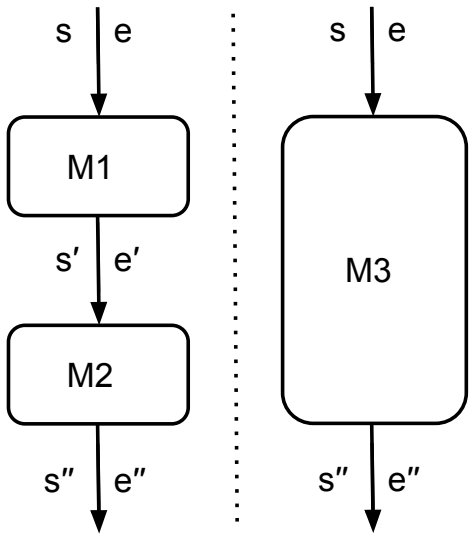

Figure 3: Machine M3 can be viewed as an aggregate of M1 and M2.

Creating Structures and Machines

We can extend the metamodel summarised above to include the ability to create and change Structures and Machines by introducing a MachineTemplate (MT) and Location Structure and three specific classes of Machine: Copier, Locator, and Constructor. These are shown in Figure 4 and summarised as:

MachineTemplate: a Structure containing the instructions for building a particular Machine, specifically: its initial state (e.g. the default parameter settings); components of the behaviour including what input structures it can receive, and what outputs structures are generated; energy storage ability. Importantly, MTs exist separately from machines and a Machine does not store its describing MT; they are different entities in the metamodel.

Copier: creates a copy of an input Structure, leaving the original Structure unchanged. The copied Structure could be exact (same Symbols), erroneous (potentially different Symbols from same Alphabet), or a translated copy (different Symbols from different Alphabet) based on the Copier's behaviour.

Locator: locates a sub-Structure of a Structure given Location identifiers that denote the beginning and end of the sub-Structure.

Location: a Structure that acts as an identifier on another Structure. Two specific Locations are required by the Locator, the Begin (cf. DNA promotor, start codon) and the End (cf. DNA terminator, stop codon).

Constructor: constructs a Machine from a MachineTemplate. It utilises a Copier and Locator Machine to construct the Machine's Structure representation from the MachineTemplate (cf. creating a polypeptide from RNA). The Machine's Structure is then given the property of Process, however that is defined in the model (cf. folding a polypeptide into its functional protein form).

It is interesting to note that different ConstructorMachines could create different Machines from the same MachineTemplates as they interpret the Symbols differently. Additionally, as the ConstructorMachine itself can have an associated MT, the Copier could potentially change the MTs

for new ConstructorMachines, thus the system itself could dynamically modify how future Machines are interpreted.

With both Machine and MachineTemplate we have a system in which we can potentially change only instances of Machines – a Machine can modify an instance of another Machine – or all future copies of a given type of Machine by modifying its associated MachineTemplate.

The machine metamodel should be general yet expressive enough to model many different systems, not just the intended EvoEvo metamodel described in the rest of this paper. For example it can capture a very general model of computation: a Machine is an executable computing process (compiled or interpreted); MachineTemplate is the source code for a Machine; the ConstructorMachine is an interpreter/compiler; Space and Time are a process scheduler.

Requirements for EvoEvo

As discussed in the Introduction, EvoEvo is based on the idea that evolution is able to shape its own path given that the molecular systems involved in the evolutionary process have resulted from past evolution. Specifically, there are four characteristics of the genotype-to-phenotype mapping that can enable EvoEvo Beslon et al. (2014):

Variability: the ability to generate new phenotypes via changes in mutation rates and operators

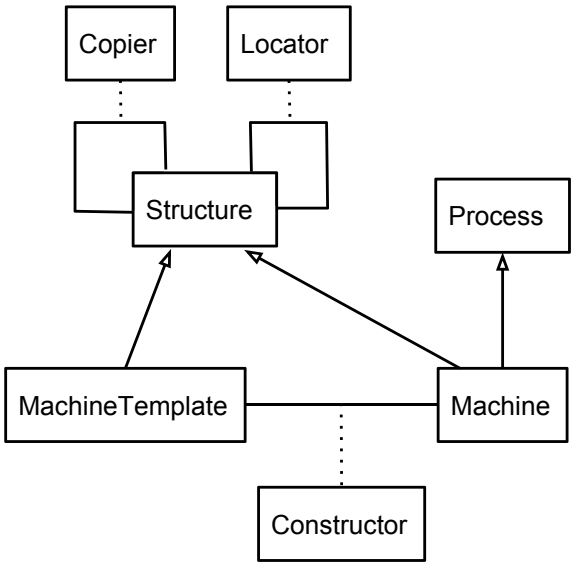

Figure 4: Class diagram showing the relationship between Structure, Machine, Copier, Locator, MachineTemplate, and Constructor. Copier and Locator inherit from Machine (not shown) and operate on Structures. A MachineTemplate is a Structure. A Constructor inherits from Machine (not shown) and creates a Machine from MachineTemplate.

Robustness: the ability to cope with mutational events without negatively impacting fitness

Evolvability: the ability to increase the proportion of mutational events that are favourable

Open-endedness: the ability to create new evolutionary avenues and targets.

These four characteristics emerge from the underlying processes and structures and the continual evolution of these processes and structures.

EvoEvo is concerned with allowing the genotype-to-phenotype mapping and the fitness landscape to evolve over time via indirect selection, leading to properties that enable evolution in dynamic environments Beslon et al. (2014). Targets for indirect selection are focussed on the organism's:

Genetic structures: numbers of genes, position of genes on genome, operons, non-coding sequences.

Networks: gene regulatory, metabolic and "social" (interactions between individuals)

These targets will become the focus for the EvoEvo metamodel that follows.

EvoEvo Metamodel

Here we extend the Machine metamodel to create a metamodel for EvoEvo that captures the desired EvoEvo concepts just described. This EvoEvo metamodel expresses the kinds of things we would expect to see in an EvoEvo model. This metamodel is based on the processes observed in bacteria, and has been inspired by studying the EvoEvo models of our project partners Knibbe et al. (2007); Crombach and Hogeweg (2007), as well as requirements for evolutionary computing. In brief, it introduces two types of Space, Individual and Environment, and a number of specialised Structures and Machines. Importantly we make a distinction between different types of machine that operate on different aspects of the Individual. First we describe the EvoEvo Spaces and Structures, followed by the EvoEvo Machines.

Spaces and Structures

We use the Space component from the Machine metamodel to represent both an Individual – such as a bacterium – and the concept of an Environment in which Individuals exists. Individuals and Environments contain various specialised Structures and Machines.

Individual A Space that represent an organism within (or potential "solution" to) an Environment. As a Space, the Individual contains any number of Machine instances, which interact with each other to form networks of the four types of Machine (see 'Machine Types' below). An Individual has a Genome consisting of at least one MachineRepository,

which encodes MachineTemplates. A Phenotype for the Individual results from the combined action of its Machines with reference to its Environment. Based on the Phenotype, the Individual is the unit of Selection – the thing that is selected for within the Environment – and undergoes reproduction. The same individual will have different Phenotypes for different Environments. We further explore Phenotypes, Fitness and Selection later in the paper.

Environment A Space containing one or more Individuals as well as other Machines and Structures. As a Space, the Environment provides an ordering for its contents and thus describes the connectivity between Individual instances. The embodiment of an Individual in the Environment defines the "problem" that is being addressed by the Individual. For example, in a model of a bacterium this would be its ability to survive and reproduce, whereas for an evolutionary algorithm this would be a potential solution to an optimisation function. The Environment can be dynamic (Machines, Structures and Individuals can change over time), therefore this "problem" can also be dynamic. The Individual receives inputs from Environment in the form of Structures, such as Metabolites and Machines. The Environment can therefore be a source of epigenetic effects on the Individual, with these inputs interacting with the Machines within the Individual – such as those that operate on the Genome – which can in turn influence Machine expression.

Genome and MachineRepository The Genome is the source of heredity between an Individual and its offspring and is copied, and potentially, mutated during reproduction. The Genome consists of one or more MachineRepository Structures (cf. a chromosome). A MachineRepository is the source of MachineTemplates that encode the Individual's Machines (cf a gene). A MachineRepository is constructed from any number of Symbols from a single alphabet, and therefore stores any number of related MachineTemplates. Not all Symbols have to form part of a MachineTemplate (non-coding regions). Different MachineRepositories could be constructed from different Symbol alphabets for MachineTemplates for different types of Machine (see 'Machine Types' below). MachineRepositories act as a persistent store for MachineTemplates during an Individual's existence as well as allowing new MachineTemplates to evolve and for organisational pattern of MachineTemplates to occur between generations of Individuals. The presence of Begin and End Locations on a MachineRepository define location of a TranscriptionUnit. A set of Genome Machines (described below) operate on the MachineRepositories to carry out copying, mutations and transcriptions.

TranscriptionUnit A Structure that can be transcribed (copied) from a MachineRepository, and may contain any number of MachineTemplates. The section of MachineRepository that is subject to the copy is defined by the position of Locations (see below) and is the biological analogy of an operon, whilst the actual TranscriptionUnit Structure is analogous to mRNA. The TranscriptionUnit provides a mechanism to group related MachineTemplates so that the subsequent Machines are constructed together. It is noted that many evolutionary algorithms omit the the TranscriptionUnit concept, translating genes (MachineTemplates) straight from the Genome. Begin and End Locations on TranscriptionUnit define location of a MachineTemplate.

Metabolite A Structure that forms the basis for operations of the Metabolite Machines (see below). These might be the building blocks for an artifical chemistry or other symbols for processing in an evolutionary algorithm.

Machine Types

The targets for indirect selection identified above in 'Requirements for EvoEvo' – genetic structures and the gene regulatory, metabolic and social networks – provide us with a useful categorisation for machines in the EvoEvo metamodel. These machine types are:

Genome Machines: operate on the Genome, responsible for constructing Machines and reproduction.

Metabolism Machines: provide a potential "solution" to the "problem" that is being addressed by the Individual (e.g. staying alive in order to reproduce in the case of a bacterium).

Regulation Machines: help control the dynamics of Machine construction from the Genome by repressing or inducing the action of other genome machines.

Boundary Machines: control the transport of Structures to and from Individuals and its external Environment.

Instances of each machine type interact (via Structures) to form machine networks. Interactions also occur between these networks to express the Individual's Phenotype. There is a clear analogy to the work of Lones et al. (2013) who come to a similar categorisation of biochemical networks.

Genome Machines

The Genome Machines exist within an Individual and operate on its Genome and TranscriptionUnit Structures. Essentially, these are the Machines that both decode the information stored on these Structures creating new Machine instances, and the Machines that create copies of these Structures.There are 5 specific types of Genome Machine that would be relevant to all EvoEvo models: Transcriber, Translator, Expresser, Cloner and Reproducer.

Transcriber This Machine is responsible for producing TranscriptionUnits from a MachineRepository. It receives as input a Structure (such as a MachineRepository) and two

Locations, Begin and End, which define the beginning and end locations of the DataStruture to be copied (i.e. the location of the operon). As output, the Transcriber returns the original Structure unchanged and the copy that has been made. Figure 5 demonstrates the Transcriber in action. We can define the process implemented by the Transcriber in terms of the Locate and Copy machines described in the machine metamodel. Locate is used twice to find the Begin and End Locations and the Copy is used once. In the case of a real biological system, the Copy will slightly change Symbols from the DNA bases (C, G, A and T) to RNA bases (C, G, A and U).

Translator This Machine is responsible for producing Machines from TranscriptionUnits. Similar to the Transcriber, it receives as input a Structure (the TranscriptionUnit) and two Locations, Begin and End, which signal the beginning and end locations of MachineTemplates within the TranscriptionUnit. As output, the Translator returns the original TranscriptionUnit and a Machine for every MachineTemplate located. Figure 5 demonstrates the Translator in action. We can define the process implemented by the Translator in terms of the Locate, Copy and Constructor machines described in the machine metamodel. For each MachineTemplate, Locate is used twice to find the Begin and End Locations, the Copy is used once to build the Structure representation of the Machine, and Constructor is used to turn this Structure into a Process (and hence an instance of Machine).

Expresser This Machine operates on a MachineRepository controlling the expression of TranscriptionUnits. It operates by providing the Transcriber with the Locations and

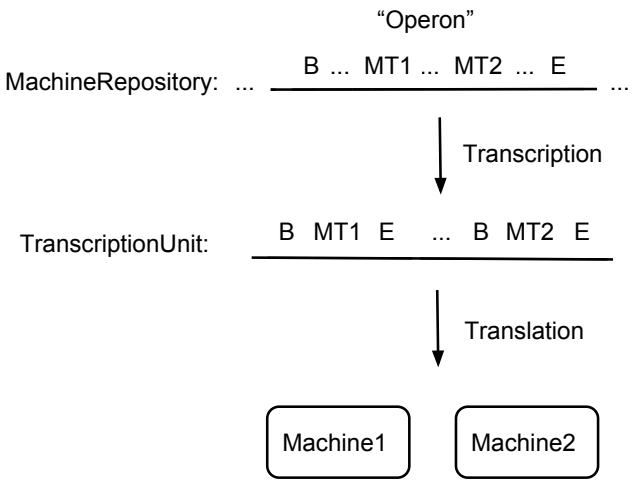

Figure 5: The interaction between the MachineRepositor, TranscriptionUnit, MachineTemplates, Machines, Transcription and Translation. B denotes Begin, E denotes End.

MachineRepository at the operon chosen for transcription. By working in conjunction with regulation machines (Regulators) this allows for different expression rates for TranscriptionUnits.

Cloner The Cloner is a Copier Machine that performs an inaccurate copy of a MachineRepository. This provides a mechanism to introduce mutations and recombinations to a copy of the MachineRepository. It takes as input a MachineRepository and returns as output the original MachineRepository (unchanged) and inaccurate copy of MachineRepository. How the Cloner changes the copy is problem specific – it will contain rules on which data symbols can change and how they change. Different Cloners will introduce different types of mutation and recombination.

Reproducer Responsible for creating a new Individual based on the current Individual. It requires each MachineRepository to be copied by the Cloner, with the child Individual receiving the cloned MachineRepositories, potentially generating novelty between generations of Individual. The child Individual will also inherit a share of the Structures and Machines present in the parent. Depending on the model that instantiates the Reproduce, one option for the parent Individual is that it dies (see 'Structure Degradation and Death') as a consequence of the Reproducer.

Metabolism Machines

Metabolism is the set of processes within an Individual that maintain its *viability* as an Individial within an Environment. For an organism such as a bacterium, this would be the staying alive in order to reproduce, whilst for an evolutionary algorithm this would be equivalent to the provision of a fitness function. Essentially, metabolism provides a potential "solution" the "problem" that is being addressed by the Individual (e.g. in the case of a bacterium).

Metabolisers are machines that perform the metabolism of the Individual, which together form the metabolism network. The behaviours implemented by the Metabolisers will be very much specific to the model that instantiates the metamodel. For example a model of biology might have an set of enzyme-related reactions, whilst for an evolutionary algorithm the Metabolisers they will collectively form the fitness function. As the definition of metabolism is specific to the model that instantiates the metamodel, there is little more we can say about specific Metabolisers.

The metabolic network will interact with the other machine networks, which together gives the Individual its Phenotype. The role fulfilled by the Metabolisers is to determine the viability of the individual in the current Environment. This viability is a key component in establishing fitness for the selection dynamics (see 'Phenotype, Fitness and Selection'). How selection is performed will be model specific.

Regulation Machines

Regulation within an Individual is performed by Regulator Machines. Specifically, these are a network of Machines that provide a set of extra dynamics on top of the genome machine network, controlling the action of the Transcriber and Translator. Both the MachineRepository and TranscriptionUnit are open to the action of the Regulators, cf. transcription factors and regulatory RNA.

Regulators identify their units of regulation (e.g. operon) by locating an associated Begin Location and BindingSite Location that is unique to the Regulator. The Regulator will then interact with the Machine responsible for coping that unit of regulation (e.g. Transcriber or Translator) to either repress or induce its action, which will either down- or up-regulate its expression.

The combined action of the Regulators forms networks such as gene-regulatory networks. Interactions can also occur with metabolism and boundary machines providing feedback control so that Machine expression can adapt to environmental pressures and signals. Additionally, the Regulators are encoded on the MachineRepository so the regulatory networks can themselves evolve over the generations.

Boundary Machines

Boundary Machines control the interface between the Individual and the Environment controlling transport across that boundary, determining what gets in and what goes out. If the Boundary Machines are present on the MachineRepository, they will be able to evolve the dynamics of this transport. In controlling the boundary, these Machines provide the environmental/epigenetic context for the Individual.

Any Structure is capable of being transported in and out of the Individual. Three specific types would be Metabolites, Metaboliser Machines and MachineRepositories. The first two are provide a mechanism for resources to flow in and out of the Individual, whilst the later enables processes such as horizontal gene transfer. In bacteria, horizontal gene transfer is takes place via plasmids, which are akin to MachineRepositories in this metamodel.

Structure Degradation and Death

The degradation of Structures (and by definition, Machines) drives change as new copies of Machine will be needed to replace those that degrade. Degradation is captured by the Entropy Machine/s, which are defined at the level of the Environment. Different Entropy Machines may be required for the different types of Machine. Importantly, Entropy Machines should never itself be subject to evolutionary change as they are essentially defining the *physics* of the system in which Individuals are evolving. For example, an Individual should not be able to cheat death by redefining the 2nd law of thermodynamics.

Entropy Machines degrade Structures into their constituent Symbols, which are then available to the containing space (Individual or Environment). Rates of degradation can be different for different Structures (e.g. DNA is more stable than RNA). The degradation behaviour within an Individual can lead to death – insufficient Machines and Structures to remain viable. A dead Individual can be release its contents into the Environment.

Phenotype, Fitness and Selection

We established above that the Indvidual's Metabolisers determine its *viability*. The reproduction of an Individual occurs as a result of this viability and the current conditions in the Environment, which is determined by the Selection machine. Selection does not just select the most viable (the fittest) Individuals to reproduce, but takes into consideration conditions in the environment. So, the Selection Machine will reproduce an individual based on a function of the Individual, e.g. does it have enough resource, and the Environment, e.g. is there enough space or is there a suitable mate (if the model incorporates sexula reproduction). Like the Entropy Machine, the Selection Machine is defined as a property of the system does not change behaviour over the shorter timescales of any simulation that implements the metamodel.

The timing of the Selection Machine is defined as part of the model that applies the metamodel. Selection could apply to all Individuals at the same time (cf evolutionary computing) or as and when certain conditions are met (cf bacteria when it has aquired sufficient resources and space).

Discussion

Embodiment and Machine Origins

Other than the requirement for fixed Entropy and Selection Machines, the metamodel says little about where Machines and MachineTemplates arise. It will be a modelling/design decision as to which MachineTemplates are located on the MachineRepository (and so the associated Machines are expressed by the Individual's genome machinery), or are statically defined ("hard-coded") into the system. These statically defined Machines constitute the physics of the system – those parts that cannot change. In the EvoEvo framework, the computational problem will dictate which machines are fixed and which are able to evolve. For each problem, a suitable set of Machines will need to be designed.

MachineTemplates on the MachineRepository are obviously evolvable within the system. This *embodiment* of MachineTemplates allows the instructions on how to make the functional parts of the system (the Machines) part of the system itself. This makes them accessible to change, and potentially would allow the system can change its own encoding in an *open-ended* way.

Representations

The flexibility of the Machines upon which the EvoEvo metamodel is based has a number of consequences for logic

and data representations. The Machines essentially apply the semantics to the syntax given by the Structures. Likewise meaning is only given to the Symbols on a Structure, such as MachineTemplate or MachineRepository, when it is processed by a Machine. It is the Machines present in the system that define the overall behaviour, and the expression of these Machines can be controlled by the system itself.

The same MachineTemplate can be interpreted in different ways by different Machines to, in principle, construct entirely different Machines. Similarly meaning is only given to other data representations such as orderings on a Structure (e.g. operons and the positions of different MachineTemplates) when they are processed by other Machines.

Different Begin Locations on the same TranscriptionUnit can be used as targets for different Translator machines. For example one Begin Location could be target for the Translator of Metabolite Machines, whilst a different Begin Location can be used for a Translator of Genome Machines. This would allow the two different Translators to build different Machines types from the same MachineRepository

Within the genomes of organisms seen on Earth there are different levels of structure and organisation such as: base, codon, gene, operon. The ability to define the behaviour of Machines that operate on a MachineRepository, and to allow these Machines to evolve, could allow different representations of the MachineRepository to evolve. This would give us insight about evolution *as it could be* in an artificial world.

With regard to computational evolutionary systems, being able to move between different representations of the same MachineRepository would help improve performance and exploit the best search space dynamics (exploration versus exploitation). For example some Copiers might mutate a level equivalent to bases (A,C,G,T or binary strings), whilst others could mutate at the level of MachineTemplates (e.g. moving, duplications). Which Copiers are currently used would be defined by the system itself if their MachineTemplates are evolvable.

Conclusions and Future Work

We have outlined and discussed above a metamodel for EvoEvo, which aims to abstract and interpret observed biological concepts in a form suitable for in silico implementation. In particular, the inspiring biological concepts arise from the evolutionary dynamics observed in bacteria and viruses. The metamodel is the first step towards building suitable computational analogues of the EvoEvo mechanisms that will form the basis of a computational framework enabling the development of novel evolutionary engineered systems.

The Machine metamodel gives us the language of process and structure, and the Machine which embodies process within a structure. The Machine metamodel also provides descriptions of Machines that enable us to create Machines encoded within MachineTemplate structures. The generalised Machine metamodel then provides the language to specify the EvoEvo concepts we obseve in the inspiring bacterial systems. We noted in the Introduction that there are any number of different EvoEvo metamodels that could defined. The concepts presented here in this metamodel have been specifically selected to enable the next stage of our research: to implement these concepts *in silico*.

Having established the EvoEvo metamodel, we plan to show how it can capture the models developed by our project partners, including Knibbe et al. (2007); Crombach and Hogeweg (2007), as well as being a model for evolutionary algorithms. Given this, we can address our ultimate goal of to implement the EvoEvo computational framework that will allows the user to create problem-solving algorithms based on the EvoEvo concepts in the metamodel.

As highlighted above, it is no coincidence that the Machine metamodel has many natural analogies to computational systems as it was a requirement of the metamodel to provide the basis for the implementation of the computational framework. One desirable analogy is the natural link between Machines and self-contained objects, which provides a natural link to a number of concurrency techniques. Our hope is implement a powerful EvoEvo framework, both in term of dynamic and computational power.

Acknowledgements

This work was funded by the EU FP7 project EvoEvo, grant number 610427. Thanks to Simon Hickinbotham and Chris Timperley for their useful discussions.

References

Andrews, P. S., Stepney, S., Hoverd, T., Polack, F. A. C., Sampson, A. T., and Timmis, J. (2011). CoSMoS process, models, and metamodels. In Stepney, S., Welch, P., Andrews, P. S., and Ritson, C. G., editors, *Proceedings of the 2011 Workshop on Complex Systems Modelling and Simulation, Paris, France, August 2011*, pages 1–13. Luniver Press.

Beslon, G., Elena, S. F., Hogeweg, P., Schneider, D., and Stepney, S. (2014). Evolution of evolution: Description of work. http://evoevo.liris.cnrs.fr/description-of-the-evoevo-project/.

Crombach, A. and Hogeweg, P. (2007). Chromosome rearrangements and the evolution of genome structuring and adaptability. *Molecular biology and evolution*, 24(5):1130–1139.

Hoverd, T. and Stepney, S. (2011). Energy as a driver of diversity in open-ended evolution. In *ECAL 2011, Paris, France, August 2011*, pages 356–363. MIT Press.

Kleppe, A., Warmer, J., and Bast, W. (2003). *MDA Explained: the Model Driven Architecture: practice and promise.* Addison-Wesley.

Knibbe, C., Coulon, A., Mazet, O., Fayard, J.-M., and Beslon, G. (2007). A long-term evolutionary pressure on the amount of noncoding DNA. *Mol. Biol. Evol.*, 24(10):2344–2353.

Lones, M. A., Turner, A. P., Fuente, L. A., Stepney, S., Caves, L. S. D., and Tyrrell, A. M. (2013). Biochemical connectionism. *Natural Computing*, 12:453–472.

Paul Andrews, Susan Stepney (2015) A Metamodel for the Evolution of Evolution. Proceedings of the European Conference on Artificial Life 2015, pp. 621-628