EvoEvo Deliverable 4.1

Paul Andrews, Susan Stepney

EvoEvo Deliverable 4.1

Computational Metamodel Definition

Due date:	M18 (April 2015)
Person in charge:	Susan Stepney
Partner in charge:	University of York (UoY)
Workpackage:	WP4 (A computational EvoEvo framework)
Deliverable description:	Metamodel that captures the EvoEvo concepts that we plan to embed within computational framework to generate EvoEvo inspired algorithms

Revisions:

Revision no.	Revision description	Date	Person in charge
1.0	Initial version of deliverable (internal version)	26/04/15	P. Andrews (UoY)
1.1	Review, and minor modifications	28/04/15	S. Stepney (UoY)
1.2	Typos corrected from review comments	3/07/15	S. Stepney (UoY)

Table of Contents

1. Introduction

The role of work-package (WP) 4 is to build suitable computational analogues of the biologically-oriented EvoEvo mechanisms identified by WP2 and WP3. These computational analogues will form the basis of the EvoEvo computational framework that will enable the development of novel evolutionary engineered systems, demonstrated in WP5. We have previously identified the essential role of domain modelling and principled software engineering for developing coherent bio-inspired computational systems [Stepney et al., 2005; Andrews et al., 2011], which highlights the need to establish a ***metamodel*** that captures the core EvoEvo concepts identified by WP2 and WP3. Such a metamodel captures the types of thing present across a suite of models (in this case the outputs from WP2 and WP3), and provides a language for constructing related models. It should expose the underlying principles and abstract away from irrelevant details.

The objectives of WP4 are:

1. to define a computational metamodel of EvoEvo, by abstracting and interpreting the biological model in a form suitable for in silico implementation;
2. to instantiate the metamodel into a computational model suitable for specifying demonstrator applications of EvoEvo;
3. to implement the computational model as an executable computational platform, suitable for developing demonstrator applications of EvoEvo.

This deliverable, D4.1, addresses the first objective, presenting a metamodel that is a suitable abstraction of the biological EvoEvo concepts, in particular those concepts presented in the integrated EvoEvo model in deliverable D2.7. This metamodel will form the basis for the computational model (D4.2), upon which the EvoEvo framework (D4.3) will be built.

To develop the metamodel, we have used the approaches for developing conceptual models of bio-inspired algorithms [Stepney et al., 2005] as refined by the CoSMoS metamodelling approach [Andrews et al., 2011; Stepney et al., 2015]. This method is outlined in section 2, showing how WP2 and WP3 outputs have been studied in order to identify the metamodel. In section 3, "*The Machine Metamodel*" we describe a general metamodel. This forms the basis for the EvoEvo metamodel itself, presented in section 4. In section 5, "*Capturing the Integrated EvoEvo Model*" we demonstrate how the integrated EvoEvo model described in D2.7 can be seen as an instance of our EvoEvo metamodel. (In D4.2 we show how a generic evolutionary algorithm is also an instance of our EvoEvo metamodel.) We conclude by summarising how the EvoEvo metamodel will be used to achieve objectives 2 and 3 of WP4. The Machine metamodel and EvoEvo metamodel appear in essentially this form in [Andrews and Stepney, 2015].

2. CoSMoS Methodology

To develop the EvoEvo metamodel, we have been guided by our previous methodological approaches [Stepney et al., 2005; Andrews et al., 2011; Stepney et al., 2015], which examine the relationship between conceptual models, metamodels, and bio-inspired algorithms. These approaches are collectively referred to as the CoSMoS approach. Here we describe this approach and show how we have used it to study the biological concepts that underpin the biological models developed by our collaborators in WP2 and WP3. It is upon these biological concepts that the EvoEvo metamodel (described in the subsequent sections) is based.

The CoSMoS approach provides guidance on how to engineer, and subsequently use, computer simulators for scientific investigations. At the heart of CoSMoS is a collection of core components, shown in Figure 1. These components are used as devices to capture, communicate and reason about different aspects of the construction and use of the simulation platform, and how this relates to the system under study.

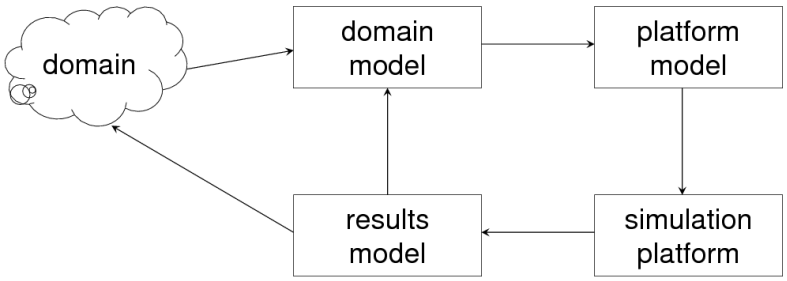

Figure 1. The core CoSMoS components.

The core components can be summarised as follows:

- **Domain**: represents a particular view or perspective of the real world system (or part of a system) that is being investigated
- **Domain Model**: a model that explicitly captures aspects of the domain, identifying and describing the structures, behaviours and interactions, and both the micro and macro (emergent) levels. It may also capture a model of the experiments (or experimental system) that are performed on the domain concepts in the real world system. It is a model based on the science and is free from simulation implementation details.
- **Platform Model**: a model derived from the domain model that details how the concepts captured in the domain model will be implemented by the simulation platform. The model is shaped by engineering design decisions, which might dictate that some concepts in the domain model are abstracted or simplified to allow for efficient implementation. The platform model adds interfaces (to allow observation/visualisation and user interaction), and instrumentation (to allow simulation data collection and analysis via statistical probes). It removes any emergent properties specified in the domain model that are required to emerge from (rather than be explicitly coded into) the simulation.
- **Simulation Platform**: encodes the platform model into a software and hardware system upon which simulation experiments are run. The simulation platform defines a set of parameters (variables) that allow the encoded model to be manipulated, and results data to be generated and interpreted.
- **Results Model**: a model describing the behaviour of the simulation platform based on the output of simulation experiments, providing the basis for interpretation of what the simulation results show. It is constructed by experimentation and observation of simulations, and might record observations, screen-shots, dynamic sequences, raw output data, result statistics, as well as qualitative or subjective observations. The contents of the results model are compared to the domain model to establish whether the simulation platform provides a suitable representation of the real-world domain being investigated.

The relationship between the three models within the CoSMoS core components (the domain model, platform model and results model) can be expressed in terms of a common metamodel. Whilst a CoSMoS model provides the abstract language of the relevant concepts, the metamodel provides the analogous language for writing a model: it defines the kinds of things that can occur in the model (it is the model of the model) [Kleppe et al., 2003]. For example, a metamodel might contain GeneticStructure which could be instantiated as DNA or RNA in the domain, platform and results models.

Figure 2 shows how each of the three CoSMoS models can be considered an instance of the same unifying CoSMoS metamodel, as they capture the same core concepts but for different purposes (e.g. recording the science or specifying an implementation). This does not mean that the models are identical, only that they are cast in the same language. For example, a domain model has instances of metamodel concepts that capture specific domain concepts, whereas the results model has instances of simulation analogues of those domain concepts. It is important to state that not all the concepts in the metamodel need be instantiated in a model. Although a platform model shares many metamodel concepts with a domain and results model, it will not include any hypothesised emergent properties (we do not wish to hard-code the answer into the simulator), but it will include interface and instrumentation concepts not present in the domain or results models.

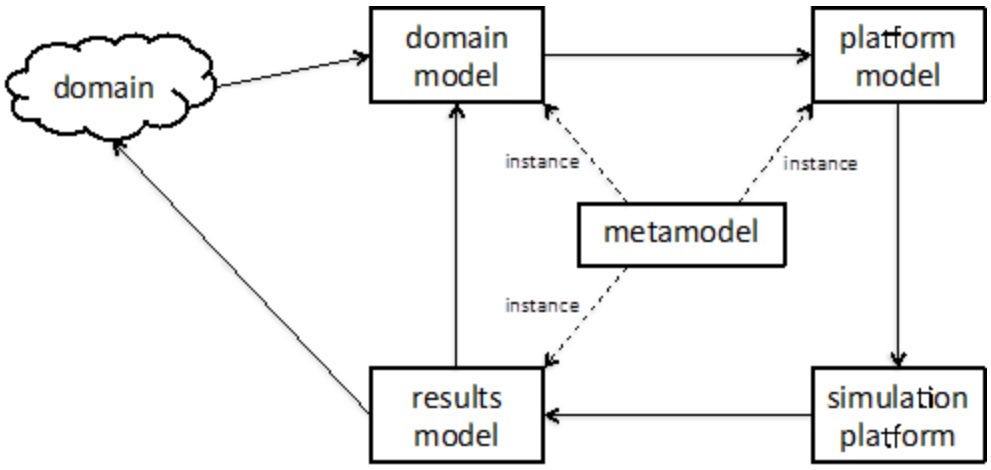

Figure 2. Relationship between the core CoSMoS components and their metamodel.

There are any number of different EvoEvo metamodels that could defined; there is no single correct metamodel. The particular EvoEvo metamodel described in the rest of this document has been chosen as it both provides a language that can capture EvoEvo concepts, and has a natural analogy to computation so that it can be readily translated into the eventual computational framework.

We use the CoSMoS approach to analyse the model outputs of WP2 and WP3 (Aevol, PoaS, and the integrated model from D2.7) to help us establish the core elements of EvoEvo that are of interest within the project. Subsequently we abstract and generalise these core elements into a suitable metamodel for EvoEvo.

Our first step is to map the WP2 and WP3 resources onto the CoSMoS components represented in Figure 1 so that we can infer the metamodel via the relationship shown in Fig 2. These resources include computer simulation code and research literature (including [Parsons 2011; Batut et al. 2013; Knibbe et al. 2007; Crombach and Hogeweg 2007; Crombach and Hogeweg 2008]) that contain descriptions of the background biology, along with model implementation details and simulation results. Figure 3 demonstrates how we map these resources. In CoSMoS terminology, our domain is the evolutionary biology of bacteria and viruses. Simulator source code, such as Aevol (available at http://www.aevol.fr/), maps to the CoSMoS Simulation Platform from which we can extract a Platform Model. Using the Platform Model and the associated research literature, we can identify the components of the domain and results models (we describe this is in more detail for the domain model of Aevol in [Andrews and Stepney, 2014]).

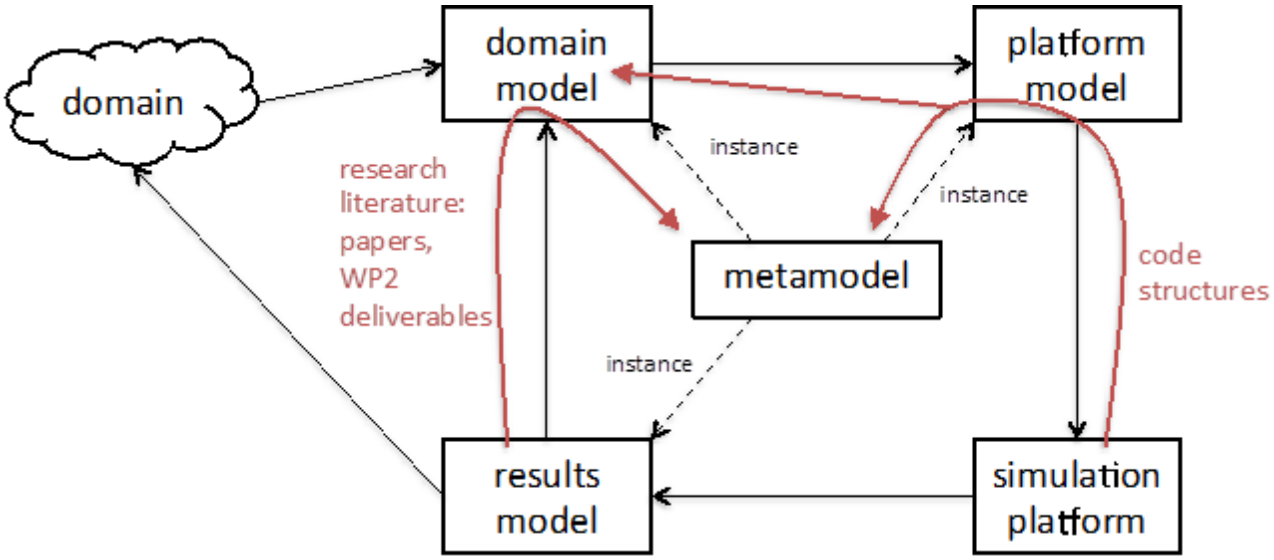

Figure 3. Process employed for extracting the EvoEvo metamodel using the CoSMoS approach

Once we have established the EvoEvo metamodel, we use it to start developing the EvoEvo framework. Figure 4 depicts the relationship between the EvoEvo metamodel and an EvoEvo inspired evolutionary algorithm. This is explored in D4.2.

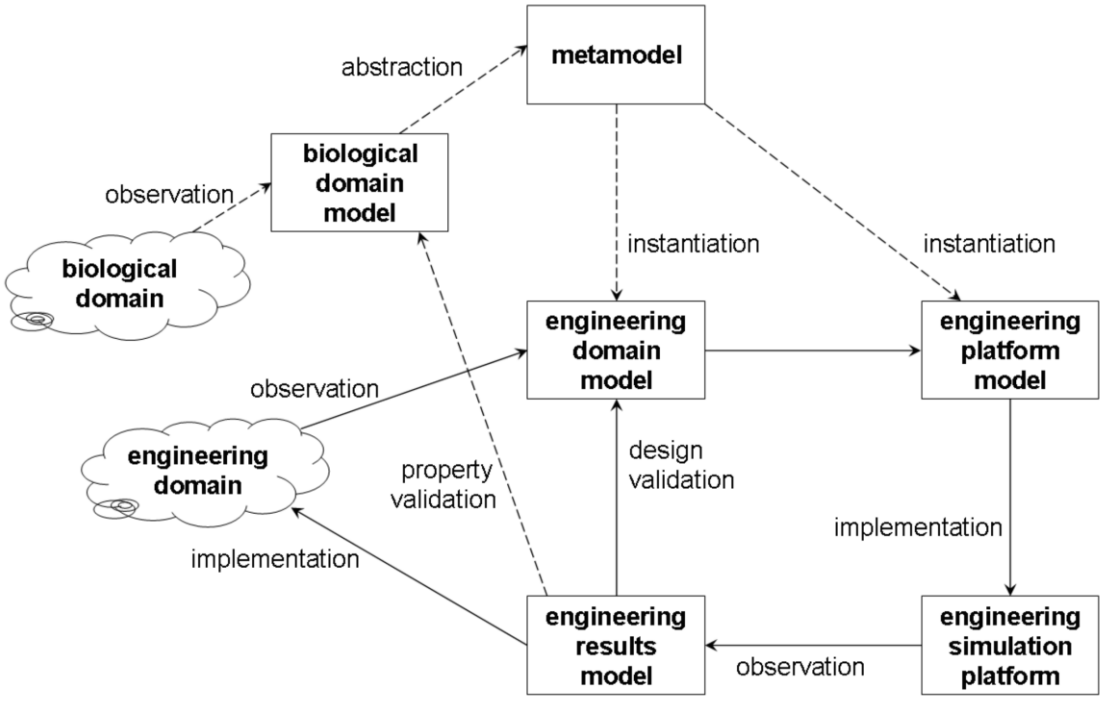

Figure 4. Relationship between metamodel and evolutionary algorithm (reproduced from [Andrews et al., 2011]).

3. The Machine Metamodel

At the most abstract level, we model a biological system or concept such as EvoEvo 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. Here, we introduce the notion of a structure that implements a process (e.g. an enzyme); we call this abstraction of a reified processes a ***Machine***.

The Machine abstraction, informally conceptualised in Figure 5, provides us with a flexible language that we use to define our EvoEvo metamodel. A Machine is structure that implements a process that receives as input structures and (abstract) energy, and returns as output (potentially) modified structures and energy. Figure 5 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 a model of energy.

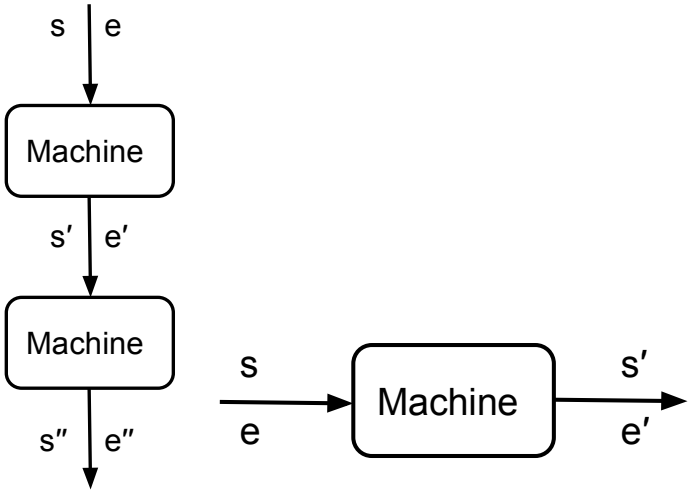

Figure 5. Machine takes structure, **s**, and energy, **e**, as inputs and returns potentially modified structures, **s′**, and energy, **e′**, as output. Networks of machines can form where outputs from one Machine are the inputs to the next.

More formally, we can capture the Machine concepts using a class diagram, shown in Figure 6. 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 possess 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; different Structures may be made from different Alphabets
- **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.

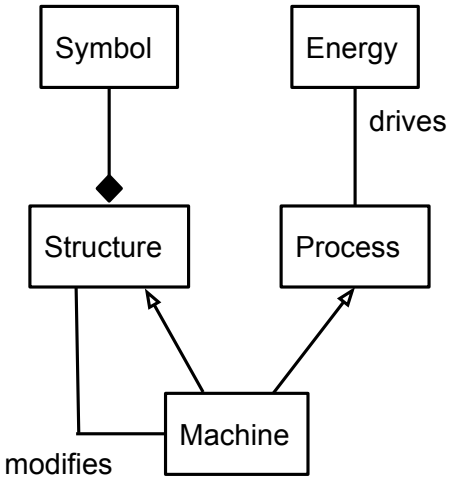

Fig 6. Class diagram showing the relationship between Machine, Structure, Process, Symbol, and Energy. Machine inherits from both Process and Structure, and modifies instances of Structure. Process is driven by Energy. Structure instances are composed of Symbols from an Alphabet (not shown).

There are several aspects to the Machine as Process.

1. 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.
2. 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.
3. A 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.
4. 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 Structure? 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) output from one machine match the input expected for another (and the energy outputs and inputs are also satisfied). Figure 7 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.

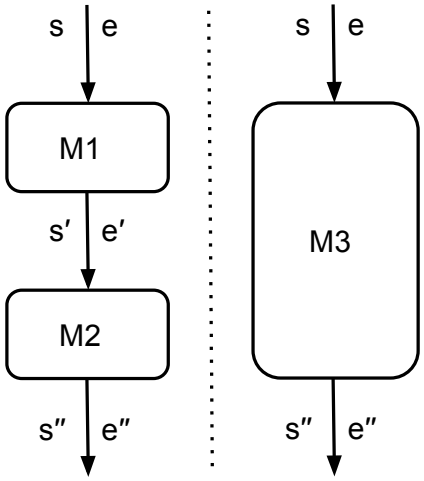

Figure 7. Machine M3 can be viewed as an aggregate of M1 and M2.

3.1. Energy, Space and Time

Energy in the Machine metamodel draws heavily from that in [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 metamodel:

- **Flux**: a flow of energy from outside the modelled system. It can have a particular temporal pattern e.g. high level during daytime, lower level during night time.
- **Store**: represents the ability of a component 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. 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

3.2. Creating Structures and Machines

We can extend the metamodel summarised above to explicitly include the ability to create and change Structures and Machines by introducing a MachineTemplate (MT), Location Structure, and three specific classes of Machine: Copier, Locator, and Constructor. These are shown in Figure 8 and are:

- **MachineTemplate**: a Structure that contains 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. MTs exist separately from machines and a Machine does not store its describing MT; they are different entities in the metamodel. (cf. protein as Machine; corresponding RNA as MachineTemplate)
- **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 *translated* (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 uses 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).

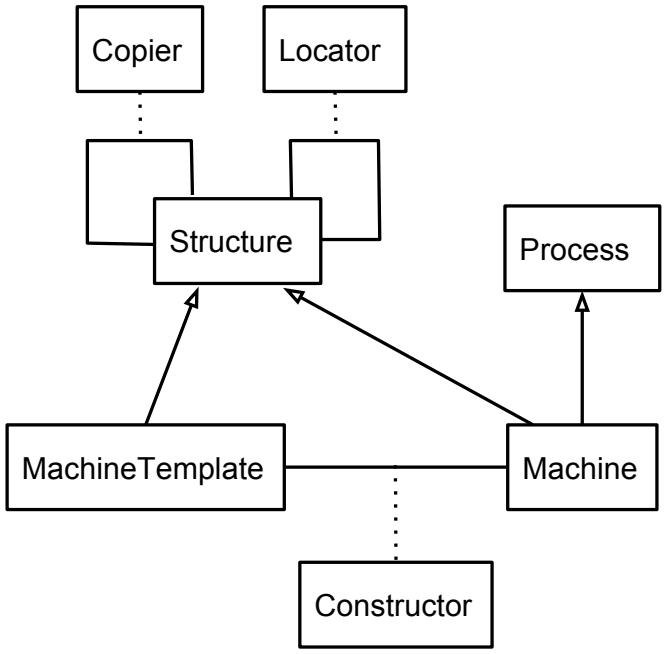

Fig 8. Class diagram showing the relationship between Structure, Machine, Copier, Locator, MachineTemplate, and Constructor. Copier and Locator inherit from Machine (not shown) and operate on Structures. A MachineTemplate is a Structure. A Constructor inherits from Machine (not shown) and creates a Machine from MachineTemplate.

Different ConstructorMachines could create different Machines from the same MachineTemplates if 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 the way 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 described in this section should be general yet expressive enough to model many different systems, not just the intended EvoEvo metamodel described in the rest of

this document. 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.

4. EvoEvo Metamodel

4.1. Requirements

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 the evolution of evolution [Beslon *et al.* 2014]:

- **Variability**: the ability to generate new phenotypes via changes in mutation rates and operators
- **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 this 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 are the focus for the EvoEvo metamodel that follows.

4.2. Overview

The EvoEvo metamodel starts from the Machine metamodel, and further introduces two types of Space – Individual and Environment – and a number of specialised Structures and Machines. We make a distinction between different types of machine that operate on different aspects of the Individual. Figure 9 presents an informal summary of concepts in the metamodel. First we describe the EvoEvo Spaces and Structures, followed by the EvoEvo Machines.

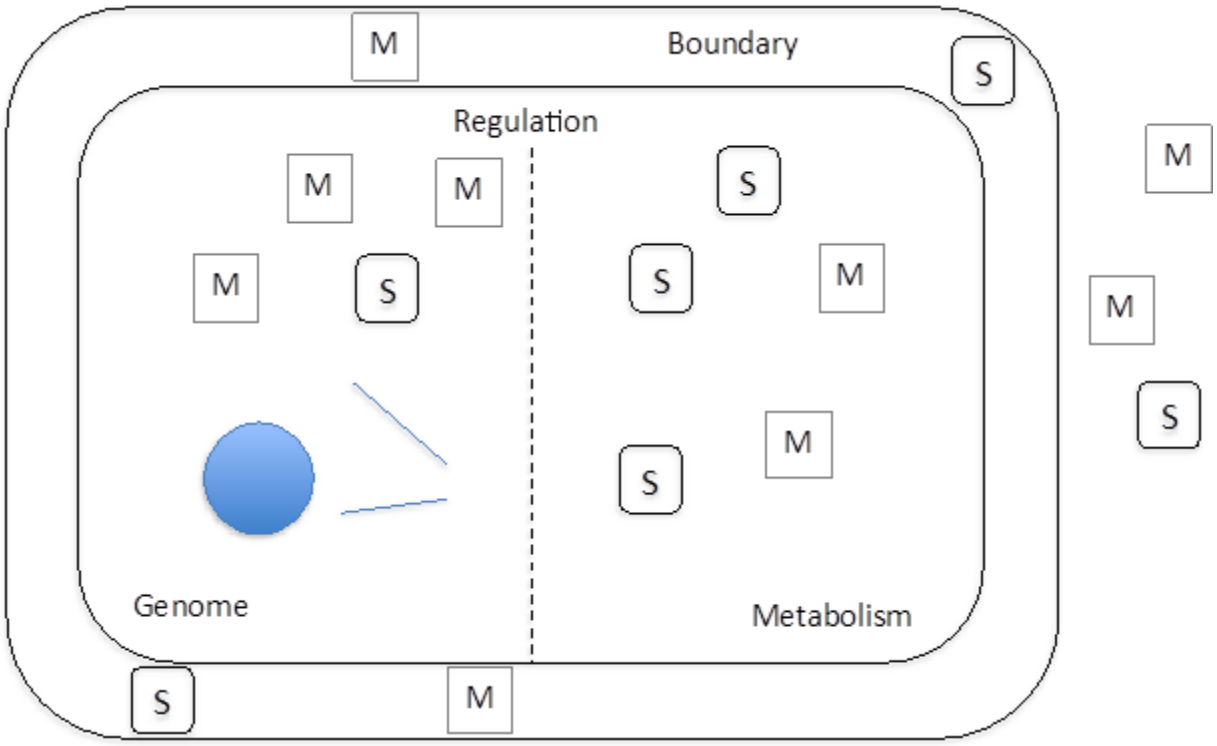

Fig 9. Depiction of main metamodel concepts. A single Individual is shown with spaces for genome, metabolism and boundary concepts. M represents Machines, S represents Structures such as Metabolites. The Genome contains a (circular) MachineRepository and two (straight) TranscriptionUnits that can be used to create Machines. Machine and Structure also exist outside of the Individual in the Environment. Boundary Machines support the transport of Machines and Structures between the Environment and inside the Individual. Regulation occurs within the Genome area, but can be affected by Metabolism Machines (cf. co-enzymes in biology).

4.3. 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 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 document.

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 it ability to survive and reproduce, whereas for an evolutionary algorithm this would be a potential solution to an optimisation problem. 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 Machine-Repositories could be constructed from different Symbol alphabets for MachineTemplates for different types of Machine (no direct biological analogue, but this allows for computational language differences between Machines).

MachineRepositoriess act as a persistent store for MachineTemplates during an Individual's existence, as well as allowing new MachineTemplates to evolve and for organisational patterns of MachineTemplates to occur between generations of Individuals.

The presence of Begin and End Locations on a MachineRepository define the location of a TranscriptionUnit.

A set of Genome Machines (below) operate on the MachineRepositoriess to carry out copying, mutations and transcriptions.

TranscriptionUnit

A TranscriptionUnit is 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 (below) and is the biological analogue 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. Many evolutionary algorithms omit the the TranscriptionUnit concept, translating genes (MachineTemplates) straight from the Genome. Begin and End Locations on TranscriptionUnit define the location of a MachineTemplate.

Metabolite

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

4.4. Machine Types

The targets for indirect selection identified above in section 4.1 – 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" the "problem" that is being addressed by the Individual (e.g. in the case of a bacterium, staying alive in order to reproduce).
- **Regulation Machines**: help control the dynamics of Machine construction from the Genome by repressing or enhancing the action of other genome machines.
- **Boundary Machines**: control the transport of Structures from Individuals to its external Environment and vice versa.

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. Figure 10 summarises some of the interactions between these four networks.

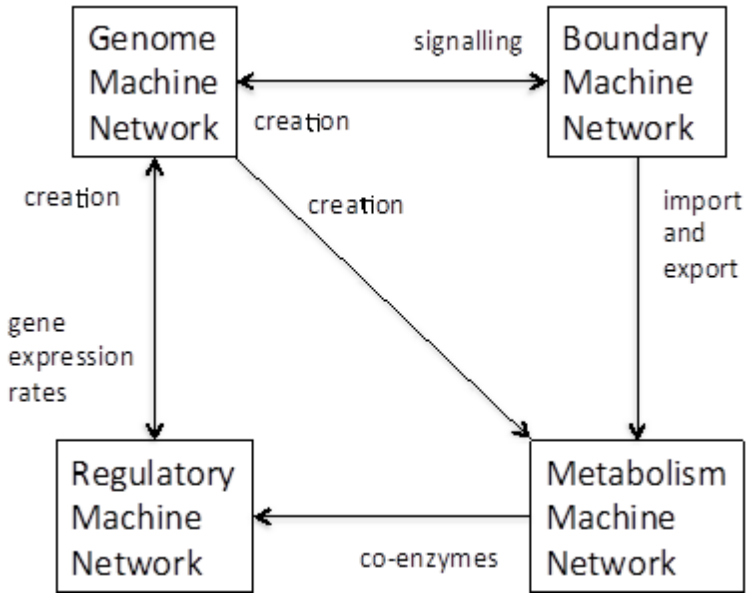

Fig 10. Interactions between machine types. The Genome Machine network is responsible for creating the Machines that make up the other three networks. The Regulatory Machine network controls gene expressions in the Genome network. Metabolism network can affect Regulatory network in the form of co-enzymes. The Boundary Network imports and exports Metabolism machines (and Metabolite Structures) and can provide signalling to the Genome Network by providing Structures from the Environment.

4.5. 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 are the Machines that create copies of these Structures. There are five specific types of Genome Machine relevant to all EvoEvo models: Transcriber, Translator, Expresser, Cloner and Reproducer.

If Genome Machines are encoded on the MachineRepository, the various genomic processes can evolve over the generations.

Transcriber

A Transcriber 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 (cf. the location of the operon). As output, the Transcriber returns the original Structure unchanged and the copy that has been made. Figure 11 demonstrates the Transcriber in action.

We 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 Copy is used once. (In a real biological system, Copy slightly changes Symbols from the DNA bases C, G, A and T to the RNA bases C, G, A and U.)

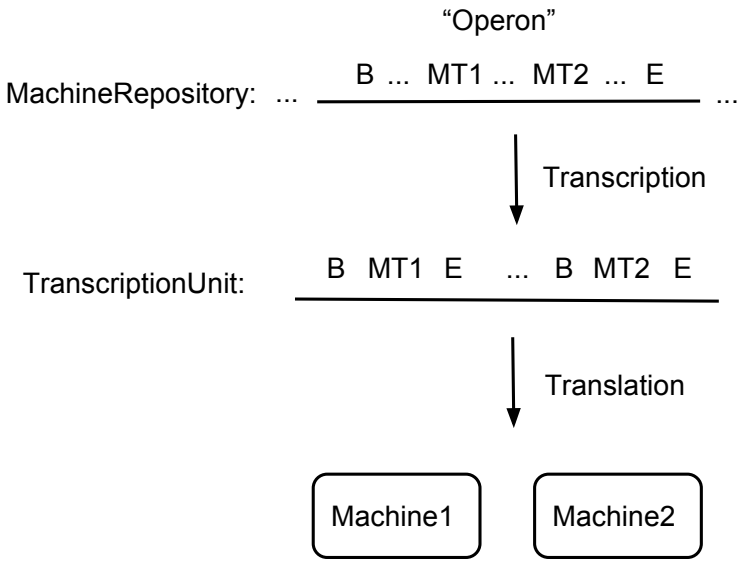

Figure 11. The interaction between the MachineRepository, TranscriptionUnit, MachineTemplates, Machines, Transcription and Translation. B denotes Begin, E denotes End.

Translator

A Translator 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 Transcription-Unit. As output, the Translator returns the original TranscriptionUnit and a Machine for every MachineTemplate located. Figure 11 also demonstrates the Translator in action.

We 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; Copy is used once to build the Structure representation of the Machine; Constructor is used to turn this Structure into a Process (and hence an instance of Machine).

Expresser

An Expresser Machine operates on a MachineRepository controlling the expression of TranscriptionUnits. It operates by providing the Transcriber with the Locations and Machine-Repository at the operon chosen for transcription. By working in conjunction with regulation machines (Regulators) this allows for different expression rates for TranscriptionUnits.

Cloner

A 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 Machine-Repository. It takes as input a MachineRepository and returns as output the original Machine-Repository (unchanged) and an 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

A Reproducer Machine is 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, thus potentially generating novelty between generations of Individual. The child Individual will also inherit a share of the Structures and Machines present in the parent Individual. Depending on the model that instantiates the Reproduce, one option for the parent Individual is that it dies (see section 4.9) as a consequence of the Reproducer.

4.6. Metabolism Machines

Metabolism is the set of processes within an Individual that maintain its *viability* as an Individual 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.

The Machines that perform the metabolism of the Individual are the Metabolisers, 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 a set of enzyme-related reactions, whilst for an evolutionary algorithm the Metabolisers 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 here 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 section 4.10). How selection is performed will be model specific.

If Metabolism Machines are encoded on the MachineRepository, the various metabolic processes can evolve over the generations.

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

The combined action of the Regulators form networks such as gene-regulatory networks. Interactions can also occur with Metabolism and Boundary Machines to provide feedback control, so that Machine expression can adapt to environmental pressures and signals.

If Regulators are encoded on the MachineRepository, the regulatory networks can evolve over the generations.

4.8. 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. 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 plassmids, which are akin to MachineRepositories in this metamodel.

If Boundary Machines are encoded on the MachineRepository, they dynamics of this transport can evolve.

4.9. Structure Degradation and Death

The degradation of Structures (and by definition, Machines) drives change as new copies of Machine are 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, the Entropy Machine should never itself be subject to evolutionary change: it is a fixed “physical” rather than an evolvable “biological” process.

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

4.10. Phenotype, Fitness and Selection

We established above that the Metabolisers of the Individual determine its *viability*. The reproduction of an Individual occurs as a result of both this viability and the current conditions in the Environment, which is determined by the Selection machine. Selection does not merely select the most viable (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). Like the Entropy Machine, the Selection Machine does not evolve and is a property of the system as a whole.

The timing of the Selection Machine is defined as part of the model. For example, 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 acquired sufficient resources and space).

5. Capturing the Integrated EvoEvo Model

As previously stated, the EvoEvo metamodel has been designed to capture the concepts common to WP2 and WP3, and to form the basis for the computational framework that will implement EvoEvo concepts within evolutionary algorithms. By way of example, we show in this section how the concepts within the integrated EvoEvo model (as presented in D2.7) are instances of the metamodel concepts described here. In D4.2 we show how this same metamodel can be instantiated for evolutionary algorithms.

The following table summarises how the integrated EvoEvo model implements the metamodel:

Metamodel Concept	Integrated EvoEvo Model (D2.7) Component
Individual	A single type of individual called a Cell. This contains the relevant genome, metabolites and machines.
Environment	Instantiated as environment. Contains the cell individuals, entropy machines (diffusion and degradation) and structures (metabolites).
Genome	Contains one MachineRepository
MachineRepository	A single ordered circular string of symbols (“pearls”)
MachineRepository Symbol Alphabet	There are 5 different symbols in the alphabet that can be used to construct a MachineRepository: E, TF, BS, P, NC. These symbols refer to the concepts they encode: enzyme, transcription factor, binding site, promoter and non-coding “pseudogene”. (Although these symbols contain substructure in the form of attributes they are viewed as atomic symbols by the Genome Machines that act upon the MachineRepository, therefore they are considered the symbols of the alphabet.)
MachineTemplate	Two types of MachineTemplate are encoded on the Machine-Repository: enzymes and transcription factors. The Machine-Templates share the symbol alphabet of the MachineRepository, and in this case, the symbols are the whole MachineTemplates.

Location	The binding site (BS) and promoter (P) symbols also act as Locations on the MachineRepository, acting as identifiers for the Transcriber and Translator. The NC symbol acts as a Stop for the Translator.
TranscriptionUnit	Not explicitly instantiated (see Transcriber and Translator)
Metabolite	Unit operated on by artificial chemistry (see Metabolism Machines). Metabolites consist of an identifier (type) and concentration.
Transcriber	As we have no explicit TranscriptionUnit, we have no explicit Transcriber. The process of creating Machines from the Machine-Repository (Figure 11) is performed by a single machine, in this case the Translator.
Translator	Creates Machines directly from the MachineRepository. It creates enzymes (metabolism machines) and transcription factors (regulation machines). It uses the binding site and promoter Locations and the NC symbol acts as a Stop for the Translator. Works in collaboration with the Expressor.
Expressor	Works in conjunction with the Regulator and Translator machines. Working with the binding site and promoter locations, it creates a higher level representation of enhancer sites and operator sites to determine transcription dynamics.
Cloner	Performs the various mutation and recombination operators.
Reproducer	Works with the Selector machine to perform fitness proportional replication in local neighbourhood of the Environment.
Metabolism Machines	Metabolism machines are implemented as enzymes, which interact as an artificial chemistry to form the metabolism machine network. Action of the metabolism machine network defines the cell's fitness. Metabolism machines transform metabolites.
Regulator Machines	Regulators are implemented as transcription factors, which interact with each other, the metabolism machines (via co-enzymes) and the genome machines to form a gene-regulatory network. Regulators work to calculate a transcription rate for each MachineTemplate (enzyme or transcription factor) on the MachineRepository.
Boundary Machines	Boundary machines are implemented as pumps on the cell that release and take-up metabolites from the environment.
Entropy Machines	Within the Environment, Entropy machines implement the metabolite degradation and diffusion, along with cell death.
Selector	Perform fitness proportional replication in local neighbourhood of the Environment. Fitness is determined by the Metabolism machines that implement the artificial chemistry.

6. Discussion

6.1. 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 is a modelling/design decision as to which MachineTemplates are located on the MachineRepository (so with associated Machines 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 evolve. 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 evolvable within the system. This *embodiment* of MachineTemplates is what allows the instructions on how to make the functional parts of the system (the Machines) be 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.

6.2. 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. Meaning is given to the Symbols in a Structure, such as a MachineTemplate or MachineRepository, only when the Structure is processed by a Machine. The Machines present in the system 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, in principle allowing them to construct entirely different Machines from the same MachineTemplate. Similarly, meaning is only given to other data representations, such as ordering on 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 we see 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 and 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.

6.3. Next Steps

This deliverable D4.1 defines the basic Machine metamodel, and an enrichment of it, the EvoEvo metamodel, and shows how the biological models of EvoEvo can be expressed as biological instances of the metamodel. The next steps are:

- to define a computational instantiation of the EvoEvo metamodel, suitable for using as a framework for EvoEvo-style evolutionary algorithms (D4.2)
- to implement such an evolutionary framework (D4.3)
- to apply the framework to suitable applications (WP5)

7. References

[Andrews *et al.,* 2011] P. S. Andrews, S. Stepney, T. Hoverd, F. A. C. Polack, A. T. Sampson, J. Timmis, 2011, CoSMoS process, models, and metamodels. *CoSMoS 2011*, Luniver Press, pp.1-13.

[Andrews and Stepney, 2014] P. Andrews and S. Stepney, Using CoSMoS to Reverse Engineer a Domain Model for Aevol, *CoSMoS 2014*, Luniver Press, 2014

[Andrews and Stepney, 2015] P. Andrews and S. Stepney, A Metamodel for the Evolution of Evolution, *ECAL 2015*, MIT Press, 2015 (accepted)

[Batut et al. 2013] B. Batut, D. Parsons, S. Fischer, G. Beslon, and C. Knibbe, “In silico experimental evolution: a tool to test evolutionary scenarios,” *BMC Bioinformatics*, vol. 14, no. Suppl 15, p. S11, 2013.

[Beslon et al. 2014] 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 and Hogeweg 2007] A. Crombach and P. Hogeweg, “Chromosome rearrangements and the evolution of genome structuring and adaptability,” *Molecular biology and evolution*, vol. 24, no. 5, pp. 1130–1139, 2007.

[Crombach and Hogeweg 2008] A. Crombach and P. Hogeweg, “Evolution of evolvability in gene regulatory networks,” *PLoS Computational Biology*, vol. 4, no. 7, 2008.

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

[Kleppe *et al.*, 2003] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: the Model Driven Architecture: practice and promise*. Addison-Wesley, 2003.

[Knibbe et al. 2007] C. Knibbe, A. Coulon, O. Mazet, J.-M. Fayard, and G. Beslon, “A longterm evolutionary pressure on the amount of noncoding DNA,” *Mol. Biol. Evol.*, vol. 24, no. 10, pp. 2344–2353, 2007.

[Lones et al 2013] 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.

[Parsons 2011] D. Parsons, *Indirect Selection in Darwinian Evolution: Mechanisms and Implications*. PhD thesis, L'Institut National des Sciences Appliquee de Lyon, 2011.

[Stepney *et al.,* 2005] S. Stepney, R. E. Smith, J. Timmis, A. M. Tyrrell, M. J. Neal, A. N. W. Hone, 2005, Conceptual Frameworks for Artificial Immune Systems, *International Journal of Unconventional Computing*, 1(3):315-338.

[Stepney *et al.,* 2015] S. Stepney, K. Alden, P. S. Andrews, J. L. Bown, A. Droop, T. Ghetiu, T. Hoverd, F. A. C. Polack, M. Read, C. G. Ritson, A. T. Sampson, J. Timmis, P. H. Welch, A. F. T. Winfield, 2013, *Engineering Simulations as Scientific Instruments*, Springer, 2015 (in preparation)