Digital Business Ecosystem

Contract n^{o} 507953

Workpackage: 6
Self-Organisation

Deliverable: 6.6
High-Level Design Specification of the Distributed Intelligence System

Information Society and Media

Project funded by the European Community under the "Information Society Technology" Programme

Contract Number: 507953
Project Acronym: DBE
Title: Digital Business Ecosystem

Deliverable No: D6.6
Due date: 31/12/2005
Delivery date: 31/12/2005

Short Description:
This report assimilates the *distributed intelligence* research into a suitable format for the engineering of the Distributed Intelligence System (DIS). As the DIS is augmentative to the Evolutionary Environment (EvE), the DIS architecture requirements are supplementary to the EvE architecture requirements, as defined in the 'Evolutionary Environment Architecture Requirements' internal project document [2]. So, this report also includes the EvE architecture requirements, which recently have been updated to include some modifications and improvements from our extended simulations and the current implementation of the EvE by Salzburg Technical University (STU), assisted by Sun Microsystems (SUN), Intel Ireland (INTEL), and others. This report will supplement our existing informal communication of the DIS architecture requirements to INTEL for its implementation, ensuring that the *distributed intelligence* capabilities, studied by us at Heriot-Watt University (HWU), are integrated into the architecture of the DIS.

Author: HWU (Gerard Briscoe, Philippe De Wilde)
Partners contributing: HWU
Made available to: DBE Consortium and European Commission

Versioning		
Version	**Date**	**Author, organisation**
1	31/12/05	Gerard Briscoe, HWU
1	31/12/05	Philippe De Wilde, HWU

Quality Check:
1st Internal Reviewer: Paolo Dini (LSE)
2nd Internal Reviewer: John Kennedy (INTEL)

Distributed Intelligence System Architecture Requirements

Gerard Briscoe
Philippe De Wilde

Intelligent Systems Lab
Department of Computer Science
School of Mathematical and Computer Sciences
Heriot Watt University

December 2005

Acronyms

ABM	Agent-Based Model
BPEL	Business Process Execution Language
BML	Business Modelling Language
DBE	Digital Business Ecosystem
DEC	Distributed Evolutionary Computing
DIS	Distributed Intelligence System
DNA	Deoxyribose Nucleic Acid
DSS	Distributed Storage System
ECJ	Evolutionary Computing in Java (library)
EvE	Evolutionary Environment
ExE	Execution Environment
FFF	Fitness Function Framework
HWU	Heriot-Watt University
ICL	Imperial College London
INTEL	Intel Ireland
ISUFI	Istituto Superiore Universitario di Formazione Interdisciplinare
JDJ	Java Developer's Journal
LSE	London School of Economics
MAS	Multi-Agent System
MUH	Migration and Usage History
NN	Neural Network
SBVR	Semantics of Business Vocabulary and Business Rules
SME	Small & Medium Enterprise
SOLUTA	SOLUTA.NET
STU	Salzburg Technical University
SUN	Sun Microsystems
SVM	Support Vector Machines
UBHAM	University of Birmingham

Contents

Executive Summary

This report assimilates the *distributed intelligence* research into a suitable format for the engineering of the DIS. As the DIS is augmentative to the EvE, the DIS architecture requirements are supplementary to the EvE architecture requirements, as defined in the 'Evolutionary Environment Architecture Requirements' internal project document [2]. So, this report also includes the EvE architecture requirements, which recently have been updated to include some modifications and improvements from our extended simulations and the current implementation of the EvE by STU, assisted by SUN, INTEL, and others. This report will supplement our existing informal communication of the DIS architecture requirements to INTEL for its implementation, ensuring that the *distributed intelligence* capabilities, studied by us at HWU, are integrated into the architecture of the DIS.

Regarding the objective of 'the research, design and implementation of a Distributed Intelligence System (DIS) based on intelligence based on self-organisation to optimise and enhance the Evolutionary Environment (EvE)' [8], we have so far partially completed the research, which is continuing in task S5, sufficiently for the architectural specification of the DIS in task C42, to support the implementation process in C53. We will continue with the task C42, in supporting task C53 in achieving the objective of an 'implementation of a DIS which optimises the EvE' [8].

Regarding the objectives of 'synchronisation with the science and computing streams to ensure that the scientific outputs are channelled appropriately into the existing high-level design of the EvE' [8] to 'ensure that the EvE high-level design is integrated optimally and faithfully into the DBE Core Architecture, upon which the DIS is dependent' [8], this is shown by the implementation of the EvE, led by STU with the assistance of SUN and INTEL, remaining faithful to the Digital Ecosystem model.

Regarding the objectives of 'the dissemination effort to distribute and explain our results to the computation work package contributors in C53 via C42' [8], the 'creation of a high-level design specification for the Distributed Intelligent System which augments the EvE, for use in the implementation of the Distributed Intelligence System' [8], 'finalisation of the EvE Architecture specification' [8], and 'a design of the DIS which is complementary to the EvE' [8], this deliverable is explicitly provided to meet these objectives and to complement our dialogue with the partners involved.

Regarding the research objectives of whether 'intelligence can optimise the evolutionary process' [8], 'how does this distributed intelligence interact with the ecosystem dynamics' [8], and whether 'software components that are part of genetic selection can be intelligent in themselves' [8], is partially answered by the now finalised *distributed intelligence* model and the preliminary results presented in the Appendix. The *distributed intelligence* model optimises the Agent-Pool at each habitat to support the evolving populations created to respond to user requests, thereby creating a distributed optimisation to support the local evolutionary optimisation. The objectives are only partially answered, because although the theoretical justification exists, the results presented in the Appendix are only preliminary. So we will be continuing with task S5, further investigating the *distributed intelligence* model, to meet fully these objectives. This will include investigating Support Vector Machines (SVM) for the DIS, to address the limitation of only a single technique being considered for the intelligence component of the DIS.

The *distributed intelligence* model has been finalised with the adoption of the *targeted-migration* approach, for which the preliminary simulation results are promising. The *distributed intelligence* model has been adapted to an architectural specification for the DIS, which has been integrated with the architecture of the EvE, which itself has been finalised. So, the engineering process of the DIS can begin and the *distributed intelligence* research will continue.

1 Introduction

This report is aimed at defining the architectural requirements for the DIS, from the *distributed intelligence* research. An updated EvE architecture has been included because of the high integration between the EvE and the DIS. It is not possible to define the DIS without the EvE, and the migration of the agents (pointers to service descriptions) within the EvE will require modifications to support the DIS. The EvE architecture has been updated to contain fewer assumptions, made possible by its initial implementation by STU. Although the DIS will create new architectural requirements for the EvE, it will also finalise the core architecture of the EvE.

Section 2 presents the Digital Ecosystem model of the EvE, integrated with the *distributed intelligence* model designed to optimise it. The *distributed intelligence* model, presented in deliverable 6.4, 'Intelligence, Learning and Neural Networks in Distributed Agent Systems' [5], has been expanded upon from improved understanding of the Execution Environment (ExE) gained from the recent EvE/DIS Integration meeting in November 2005 at INTEL, which will make it more effective in optimising the EvE.

The following sections will expand upon the sections from the Evolutionary Environment (EvE) Architecture Requirements document, jointly presenting the architectural requirements of components from the EvE and the DIS. **Section 3** defines the functionality of the 'EveService and Intelligence', describing the updated EveService and the new functionality provided by the *intelligence components* to be embedded within the EveServices to create the DIS. **Section 4** defines the functionality of the 'EveService-Pool and Agent Interaction', which has thus far been seen simply as set in the model, and as a vector in the architecture. **Section 5** defines the functionality of the 'Evolving Population' component, which architecturally remains unaffected by the DIS, but there are minor updates based upon the current implementation of the EvE. **Section 6** defines the functionality of the 'Habitats and Targeted-Migration', in which the operational effects of the DIS will be most prevalent.

Section 7 specifies the architectural requirements for the EvE/DIS derived from the previous sections, starting with the structural requirements. The behavioural requirements are then presented as sequence diagrams. Finally, the conclusions are presented and discussed in **Section 8**, including the value of the contributions to the DBE. Consideration is also given to the relation of this work to the other activities within the project.

It was felt unwise to produce an architecture requirements specification for the DIS without at least some experimental evidence of its effectiveness. This proved to be a significant task as simulating the DIS required most of the existing EvE simulation to be rewritten, because it had to be changed from a single Habitat simulation to a multiple Habitat simulation. The preliminary results are very promising, and are summarised in the **Appendix A**. The classes from our simulations for the Neural Network (NN)-based *individual intelligence* are provided in **Appendix B**, as requested during the internal review process. **Appendix C** contains our NN pre-precessing algorithm for real Semantics of Business Vocabulary and Business Rules (SBVR), which we provided as part of our continuing efforts to support the implementation process.

2 The EvE/DIS Model

The architecture of the DIS is very different to how it was seen at the start of the project. It was originally seen as a subsystem integrated with the Distributed Storage System (DSS), but separate to what is now called the EvE, providing fast responses to user requests. It was expected that the DIS would be more effective than the EvE in the short term, but less effective in the long term. The DIS developed differently for two main reasons. Firstly, the quick progress INTEL made with the DSS before the start of the DIS research by HWU (whose researchers were originally based at Imperial College London (ICL) at the beginning of the project). Secondly, the direction of the EvE research, specifically the creation of the EvE model, which showed that the 'ecosystem' concept would be fundamental in meeting the very significant and diverse goals to which the project aspires. Also, from our research orientated point of view the EvE is conceptually fundamental, and without it there is no Digital Ecosystem, or DBE. Deliverable 'D2.3 Software Roadmap'[1] states this succinctly, *'The concept (and the following implementation) of the Evolutionary Environment, is what makes the difference between the DBE project and any other project.'*.

Although the architecture of the DIS has radically changed from its original inception, it will still help to provide better solutions to users in the shorter term, than the EvE could alone. The *targeted-migration* approach chosen for the DIS will directly achieve this by targeting migration based on Migration and Usage Historys (MUHs), in response to similarity between EveServices determined by comparing their descriptions.

The authors of this report were heavily involved in the creation of the EvE, and so compatibility between the DIS and EvE models was inherent from the beginning. This has culminated in the Agent-Based Models (ABMs) of the EvE and DIS, the Digital Ecosystem and *distributed intelligence*, being presented as a single model in the following subsections. The DIS's dependency on the EvE means that the DIS can even be considered a subsystem that operates within the EvE.

2.1 Digital Ecosystem Model with Distributed Intelligence

The creation of the EvE Digital Ecosystem model, which started with the EvE Discussion Paper [6], has been achieved through our interactions with LSE, STU, UBHAM, SUN, SOLUTA, and ISUFI. The final model for the EvE was presented in deliverable D6.2 [4]. The Digital Ecosystem model contains elements from mobile agent systems, Distributed Evolutionary Computing (DEC) and ecosystems theory. It will consist of interconnected habitats just as in a biological ecosystem, with each business user being represented by a dedicated Habitat.

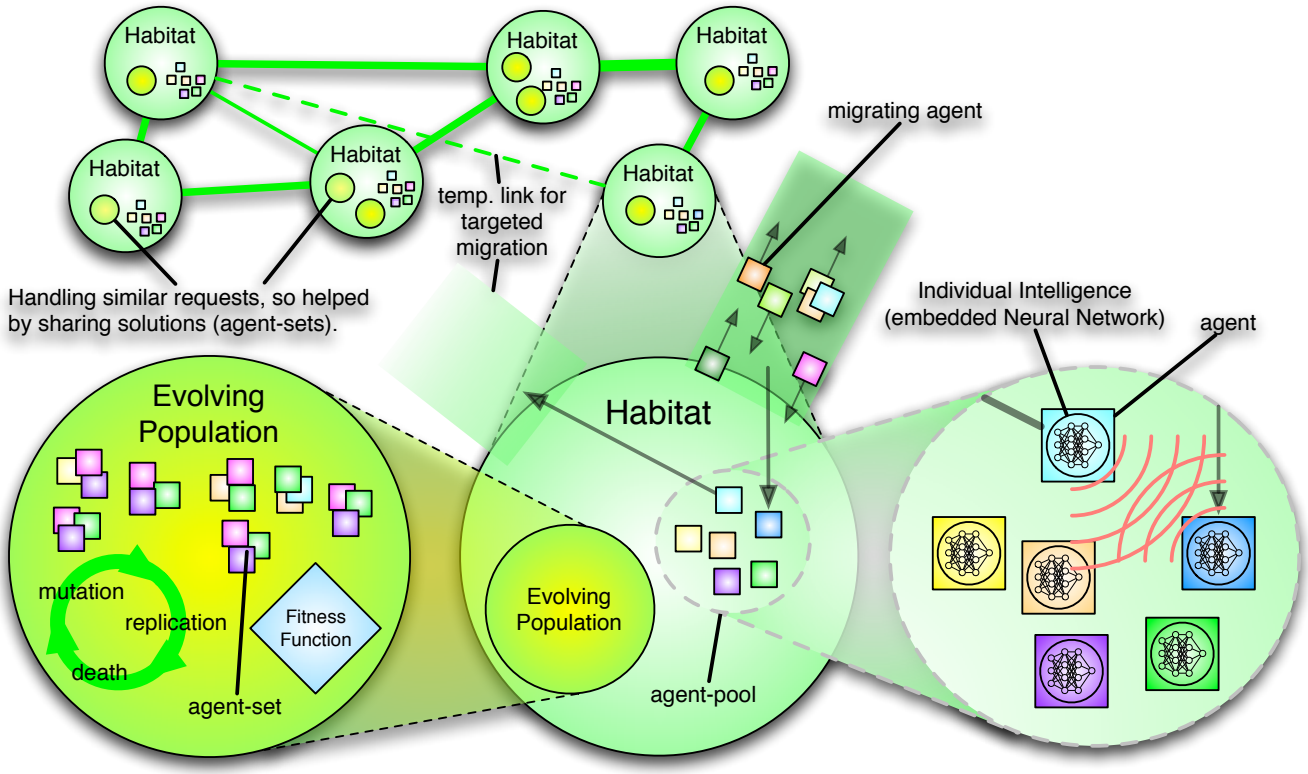


Figure 1: Digital Ecosystem Model with Distributed Intelligence

Each Habitat will provide facilities similar to an Agent Station, including an 'Agent-Pool' which is the set of agents and 'groups of agents' (applications) registered at the Habitat. The 'Agent-Pool' will also provide a 'gene pool' for Evolving Populations. An Evolving Population, similar to an 'Island' in the DEC Island Model [2], will be created to use an evolutionary optimisation process to find the optimal combination of agents, from those registered at the Habitat, to meet requests from the business users. Alternative optimisation methods have been examined in deliverable D8.1 [16], and it was concluded that the evolutionary optimisation was the most effective in addressing the DBE problem. The agents and applications (groups of agents) can then migrate through the interconnected Habitats combining with one another in the Evolving Populations to meet user requests.

The EveServices (migrating pointers to service descriptions) are modelled as mobile agents within the Multi-Agent System (MAS) model of the EvE. Individuals within our Digital Ecosystem are 'combinations of agents' (applications), created using evolutionary optimisation in response to business user requests. These individuals will migrate through the Digital Ecosystem and adapt to find 'niches' where they are useful in fulfilling other business user requests.

The *distributed intelligence* model is summarised from deliverable 6.4, 'Intelligence, Learning and Neural Networks in Distributed Agent Systems' [5]. The *directed-migration* approach has been improved, based on understanding gained through the EvE implementation. Specifically, the 'Habitat service' that will run on the ExE will be able to connect to any other Habitat (service) provided it has its 'Habitat ID'. So, agents can target and migrate directly to the Habitats where they will be of most use, instead of

waiting to come across them through the standard migration mechanism. Therefore, it has been renamed the *targeted-migration* approach.

The *distributed intelligence* will work to complement the evolutionary optimisation indirectly. An agent, when finding similarity with another agent based upon their descriptions, will be able to migrate. It will target its migration to a Habitat where it will have the opportunity to be useful, based upon the MUH of the similar agent. In biological terms, it is equivalent to providing each individual with a relatively sophisticated mechanism of social interaction. It should allow niches to be fulfilled faster, so that the Habitat clusters will reach their 'climax communities' sooner, thereby significantly improving the speed of the 'succession' process. Also, communities will be able to adapt faster to changing conditions. The is explained more fully in Appendix A.

As the EvE increases in size (number of users), the total number of agents (services) available globally will become increasingly large. For the evolutionary optimisation to continue to work efficiently as the EvE expands, optimal subsets of the power-set of agents available globally will be required at the Habitats. The migration probabilities between the Habitats will work to achieve this in a 'passive' manner. Although the mechanism is effective, the migration of agents will be relatively slow and not strongly directed. It allows the agents, based primarily upon success at their current location, to spread in the correct general direction within the Habitat network. The *distributed intelligence* will work in a more 'active' manner allowing the agents immediate highly targeted migration to specific Habitats, independent of success at the current location, instead of the generally directed migration only after success at the current location. This will help to optimise the subset of possible agent-sets found at the Habitats, which are then used in the evolutionary optimisation processes to find applications (combinations of agents) to user requests.

The agents can interact with one another, using their embedded NNs based *individual intelligence* components, to determine functional similarity based on their SBVR descriptions. SBVR is the latest version of the Business Modelling Language (BML), version 2.0. This involves comparing their SBVR descriptions to one another for similarity, and so is independent of the current Habitat's context (user requests).

At the start of the project, and the research, the structure of the BML was unclear and we required an abstraction to proceed with our work. So, we created an abstract BML consisting of numeric tuples to represent service description properties and their values, as presented in deliverable 6.1 'Self-organisation in Multi-Agent Systems' [3]. This abstract BML/SBVR was later embellished, in deliverable 6.2 'Control of Self-organisation and a Performance Measure' [4], with a filter to show that the numeric descriptions accurately abstracts sufficiently rich textual descriptions. A sample is shown in Figure 2.

agent's *abstract* semantic description = [(1,25) , (2,35) , (3,55) , (4,6) , (5,37)]

agent's semantic description = [(Business, Airline),
(Company, British Midland),
(quality, economy+),
(cost, 60 *per person*),
(Edinburgh, *to* London)]

abstract user SBVR query = [[(1,23),(2,45),(3,33),(4,6),(8,16)],
[(1,84),(2,48),(3,53),(4,11),(16,25)],
[(1,23),(2,45),(3,53),(4,6),(16,53)],
[(1,86),(2,48),(3,33),(4,25),(55,13)],
[(1,25),(2,52),(3,53),(4,5),(55,37)],
[(1,86),(2,48),(3,43),(4,25),(37,30)],
[(1,22),(2,77),(3,82),(4,9),(35,8)]]

user SBVR query =
[[(Business,Airline),(Company,Air France),(quality,economy),(cost,60),(Edinburgh,Paris)],
[(Business,Hotel),(Company,Continental),(quality,3*),(cost,110),(Paris,3 nights)],
[(Business,Airline),(Company,Air France),(quality,economy),(cost,60),(Paris,Monte Carlo)],
[(Business,Hotel),(Company,Continental),(quality,2*),(cost,250),(Monte Carlo,2 nights)],
[(Business,Airline),(Company,KLM),(quality,economy),(cost,50),(Monte Carlo,London)],
[(Business,Hotel),(Company,Continental),(quality,3*),(cost,250),(London,4 nights)],
[(Business,Airline),(Company,Air Espana),(quality,first),(cost,90),(London,Edinburgh)]]

Figure 2: Abstract BML/SBVR Descriptions

If agents are found to be similar through the inter-agent comparison, then they can share their MUHs to determine Habitats where they could be valuable. This interaction would occur outside the evolutionary optimisation, in the Agent-Pool. This will significantly change the agent's life-cycle, shown in the lighter shade (blue) in Figure 3. Basically, where and when agent migration can occur will change. There will be more opportunities for agent migration, but more importantly these opportunities will be for targeted migration.

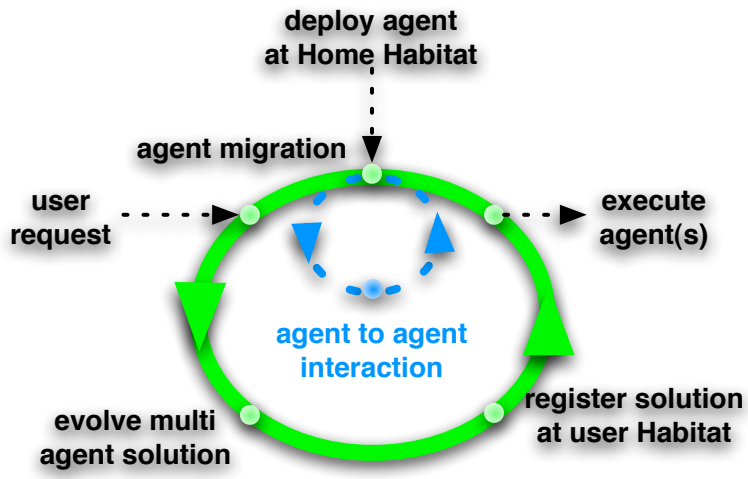


Figure 3: Updated Agent Life-Cycle

The *distributed intelligence* will interact with the ecosystem dynamics of the EvE. Embedding the NNs-based *individual intelligence* components within the agents, not only maintains the consistency of the ABM model of the EvE, but also strengthens the 'agent' definition by giving the individuals (agents) some intelligence and control over their existence.

3 EveService and Intelligence

The existing EveService definition needs to be extended to include the functionality of the agents modelled in the *distributed intelligence*. This component more than any other in the EvE will need to be extended to support the DIS.

3.1 Relation to DBE Service

An EveService is an individual within the EvE, with its genotype defined by its description, and executable code equivalent to its DNA. An EveService is primarily a pointer to the description (SM including SBVR) of the service it represents. So, many EveServices can point to the same description.

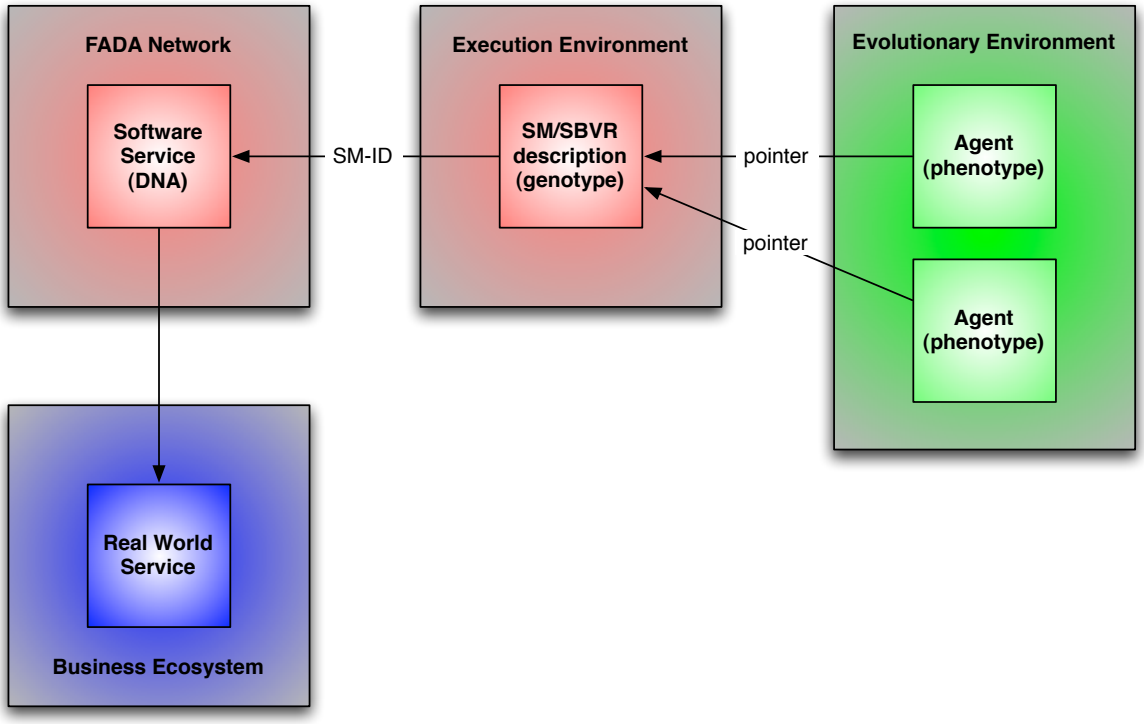


Figure 4: EveService relation to DBE service

A service in the real world, such as selling books, is represented by a software service on a computer of the Small & Medium Enterprise (SME). The service is listed on the FADA network and referenced by the SM-ID. The Service Manifest describes the service in BML (version one or two, we will assume version two, SBVR, for the time being) and is stored in the distributed Semantic Registry of the ExE.

3.2 Aggregated EveService Structure

The EveService is the base unit for the evolutionary mechanism, which means that the evolutionary process optimises the combination of EveServices to a user request. It does not change the EveServices themselves. A question from the very beginning, even before the project started, was the structure of aggregated services. Initially 'chain' and 'tree' structures where considered, since then 'sets' and 'workflows' have been proposed. The 'workflow', which is the most sophisticated, has not only been proposed but implemented with Business Process Execution Language (BPEL) by the computing stream.

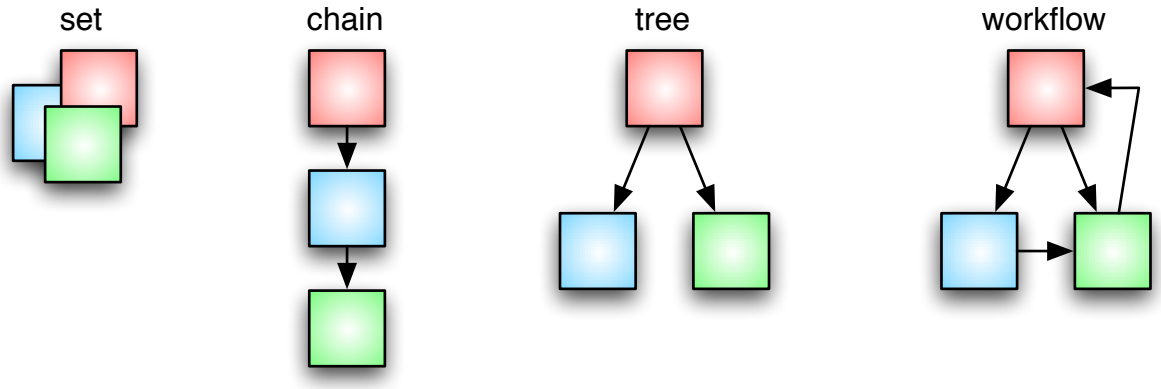


Figure 5: Possible Structures of Aggregated EveServices

Ultimately, we would like the aggregated structure of EveServices to be 'workflows', however for the present they will be 'sets' as the production of 'workflows' cannot be automated. This is caused by the EvE's dependency on the Automatic Composer, which is specifically for generating the 'glue-logic' of 'workflows' between services. The Automatic Composer faces a very significant challenge and the current approach is for it to be semi-automated, requiring minimal user input.

3.3 Life Cycle

The life cycle of an EveService exists within the life cycle of the DBE service (Service Manifest) it represents. The EveService life cycle, with the major interactions within the DBE life cycle, is shown in Figure 6.

The first stage of the service life cycle is when a user creates a Service Manifest using the Service Factory. Once created the Service Manifest is deployed in the distributed Semantic Registry, and then an EveService with the Service Manifest ID is created in the user's home Habitat. The EveService is then copied (migrated) to any Habitat connected to the home Habitat, conditionally on the probabilities associated with the connections. The migrated EveService is now available in Habitats where it is potentially useful, and upon its arrival the DIS will become active. The migrated EveService will interact one-on-one with other EveServices in the local Habitat's EveService-pool. Each EveService will process the other's SBVR description. If similar, they will share Habitats successfully visited from the MUHs. Using a limited number of *targeted-migrations* available for the function of the DIS, the successfully interacting EveServices can then perform a *targeted-migration* (via a copy) to the most promising of the recently acquired Habitats. This migration will lead to further inter-agent interaction, and possibly more *targeted-migrations*.

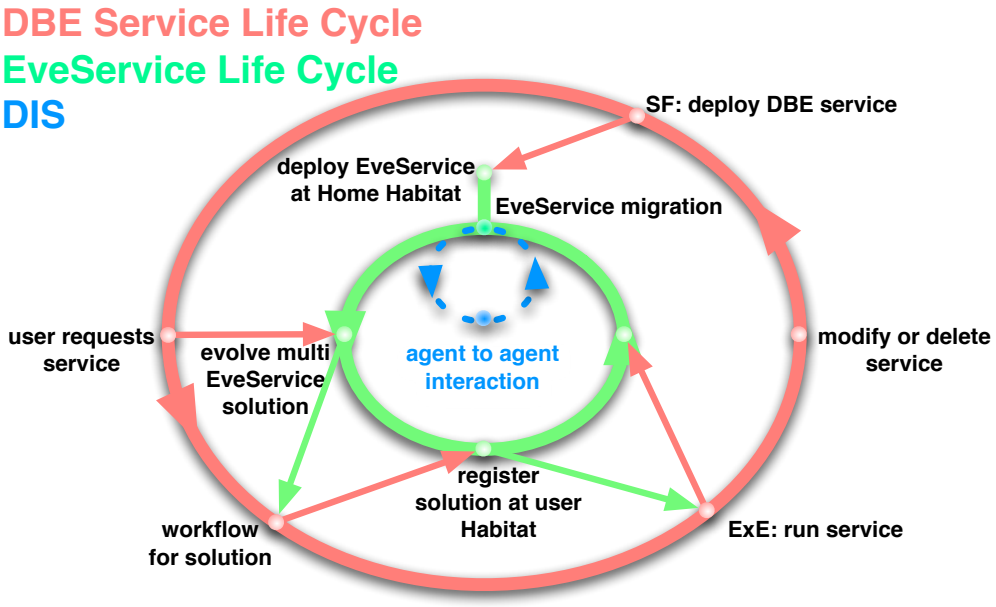


Figure 6: EveService Life Cycle

The second stage of the life cycle is when the EveService is chosen by the evolutionary optimisation in response to a user request. When the user uses the EveService, its copy (migration) to every Habitat connected to the Habitat where the EveService is currently located is then attempted. It is 'attempted' as there is a probability associated with the connection when attempting a migration, similarly to the Island Model [4, 9]. This migration will cause the DIS to become active in the Habitats into which the EveServices arrive, potentially leading to *targeted-migration*.

If an EveService is not used after several user requests, at the Habitat where it is currently located, it risks deletion. It will have a number of escape migrations, and when threatened with deletion it will relocate to a connected Habitat. After the available escape migrations have all been used, so that none remain, it will be deleted. If a Service Manifest is removed from the distributed Semantic Registry, then all EveServices pointing to it must also be deleted. This should be done passively instead of proactively. So when the EveService is being used and when resolution of the pointer for retrieving the Service Manifest fails, then the EveService should be deleted.

The targeted-migration of an agent-set is potentially problematic, because the members (agents) of an agent-set may be directed to different Habitats by the DIS. These differences in the Habitats to be visited by the agents are actually beneficial to the desired behaviour, because they can be used to ensure that agents are not migrated incorrectly to Habitats where they would not be useful. So, we must then contemplate whether to migrate all the agents of an agent-set together, forcing some of the agents to ignore the Habitats they would prefer to visit. Another approach would be to send an agent-set to the preferred Habitats of all its agents, but it would be analogous to using a flooding algorithm, especially when the agent-sets become larger. Therefore, to avoid the issue of 'flooding' and agents visiting Habitats where they are unlikely to be useful, the individual agents of an agent-set should be allowed to migrate independently. This will allow the agents to target specific Habitats where they will potentially be useful. Alternatively, subsets of an agent-set required to visit the same Habitats could be migrated as a single entity, but it would provide little if any additional benefit over the individually targeted-migration.

3.4 Service Descriptions

We have built upon, wherever possible, the components of the core architecture to create the EvE and the DIS. This is obviously the best way to interact with the computing stream, as it avoids determining new requirements for existing components unnecessarily. Unfortunately, with the *distributed intelligence* research it was not possible to avoid creating requirements on the pre-processing of service SBVR descriptions. The *targeted-migration* approach was chosen, over the alternatives in deliverable D6.4 [5], partially because it created the least requirements upon the pre-processing of service SBVR descriptions for the NN-based EveService intelligence component. Also, much time was required to create a *distributed intelligence* that was not only viable research, but had potential to be implemented. Meeting this necessity significantly slowed our progress in creating the *distributed intelligence* model.

The DIS is relatively immune to variation in the size of SBVR-based user requests and agent-sets (responses). NNs generally require strict limitations on the content and format of the information to be processed. Often, pre-processing is required to prepare the data for the input layer of a NN. Although we cannot ignore such concerns, we cannot impose such stringent requirements on the SBVR descriptions of the agents, as the EvE must cater to a large range of businesses and their services. This is managed by an inter-agent comparison for similarity. The input layers of the NNs will be sized proportionally to the SBVR descriptions of the agents that the NNs represent. More importantly, the possible shortfalls in the size of the input layer when comparing other agents' SBVR descriptions is masked by the comparison for similarity. Allowing variable-size SBVR models may appear to prevent a NN-based comparison, because of potential differences in size between SBVR models. However, the terms within the SBVR description will be alphabetically ordered in the pre-processing, which will majoratively be able to manage variable-size SBVR models for NN-based comparisons. This approach will minimise the consequences of straying from a strict information coding regime, which is not possible to maintain with the SBVR descriptions.

However, we must create one requirement, an ordering of the properties within the agent SBVR descriptions. Otherwise inter-agent interaction to compare SBVR descriptions

will not be possible. For the *abstract* SBVR descriptions, used in the modelling and simulations, the integer tuples are ordered according to the attribute name and converted to binary. The range of possible values for the attributes is determined from the Habitat cluster (opportunity space), and then sufficient padding is created for processing the binary values by the input layers of the NN-based intelligence components. This will probably be the most significant challenge for the implementation of the DIS.

3.5 Migration and Usage History

Each EveService requires a record of its migration and usage through its life span. This MUH is essential to the self-management of the EvE and the functioning of the DIS. It is simply a record of the Habitats that the EveService has migrated through, and its use at these Habitats as well as of, the other EveServices which it was used and migrated with. This will allow the **migration feedback** mechanism of the EvE to operate effectively. Specifically, the mechanism needs to know where the EveService-Sets are created, when they are used within a response to a user request. So that, the **migration feedback** mechanism can create new connections or reenforce existing connections, to where the EveService-Sets where created. This will be explained more fully in subsection 6.1.

3.6 Neural Networks for Intelligence

NNs-based *individual intelligence* components will be embedded within the agents, and so distributed throughout the Habitat network. The pattern recognition capabilities of NNs will be leveraged to allow agents to determine similarity based on their descriptions. In the simulations we are currently using multilayer feed-forward NNs, using the backpropagation training technique to specify the required pattern recognition behaviour. The optimal NN structure, such as the neuron connectivity and the transfer function, will be determined through a combination of theoretical reasoning and explorative programming.

3.7 Intelligence Training

The ideal scenario of having a large training set upon which to train a NN will not be possible in the EvE. The preliminary simulations have been with multilayer feedforward NNs with backpropogation training, in which a continuous training regime is followed. As the agent needs to be trained to recognise similarity to itself, during its deployment to the EvE, the first training set will consist of its own SBVR description. At the start of an agent's life-cycle its NN's pattern recognition capabilities would be limited, but still functioning. Agents could then train their NN based on experience. When visiting a Habitat based on an inter-agent interaction, whether successfully or not, it can be added to the training set.

3.8 Information Sharing

The interacting behaviour of the EveServices from the DIS could be considered objectionable by potential users. They may object to the successful use of their EveServices being shared with other EveServices, potentially their competitors as information sharing occurs between similar agents. Users will have to choose if they wish to allow sharing or not. Ultimately, it is a balance between the protection of existing markets and the search for new markets. We expect that users with a significant 'market share' will choose not to share information, whereas users with a poor 'market share' will be convinced to share information. Thus the DIS strengthens the levelling effect of the digital ecosystem approach.

4 EveService-Pool and Inter-EveService Interaction

Logically, the EveService-Pool is a subset of the power-set of all DBE Services available globally. In practical terms, it is a vector of the EveServices and EveService-Sets registered at the Habitat. The EveService-Pool is updated with new EveServices(-Sets) that arrive from other Habitats and EveServices-Sets generated locally by evolutionary optimisation. It is also the location from which EveServices(-Sets) are deleted when they have not been used over many user requests. The most important aspect of the EveService-Pool is that the multi-EveService solutions (EveService-Sets) found using the evolutionary optimisation are stored in the EveService-Pool, making them available to bootstrap the evolutionary optimisation in the future. This is necessary, as the combination of EveServices has value as much as the (atomic) EveServices themselves.

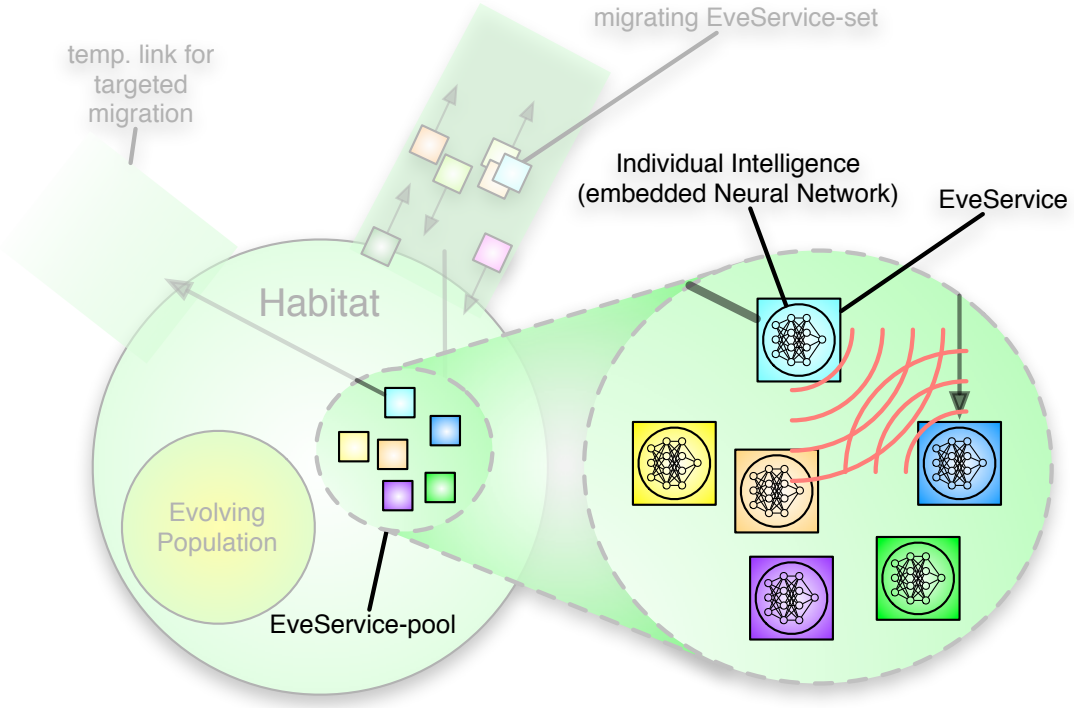


Figure 7: EveService-Pool

Once the DBE reaches critical mass there will be as many Habitats as users, potentially hundreds. There could then be thousands of EveServices, and potentially millions of combinations of EveService-Sets. For this reason, each Habitat is localised to the SME that it was created for, ensuring that the EveService(-Set)s within will have properties that match the properties of the SME. So, the EveService-Pool will contain a subset of all possible EveService(-Set)s within the DBE, the ones that are potentially useful to the SME. This is very important, as the evolutionary optimisation would be too slow if it had to work on the full set of available services. The DIS essentially works to meet this aim.

The EveService-Pool 'conceptually' has been a simple and convenient container of the EveService(-Set)s within a Habitat. The functionality of the DIS makes the EveService-Pool 'conceptually' more important in the EvE/DIS model, because all the one-on-one inter-agent interaction occurs within it, for the operation of the DIS. However, it does not necessarily make it more significant for the implementation of the DIS.

5 Evolving Populations

An Evolving Population object is instantiated in response to a user request. It uses an evolutionary optimisation technique to generate the optimal combination of available EveServices that fulfil the user request. The DIS will not affect the design or operation of the Evolving Populations. Although the Habitat network yields a novel form of DEC, the novelty with the evolutionary optimisation within a single Evolving Population comes from the ability of the Fitness Function to compare SBVR user requests to the SBVR descriptions of EveServices. This task is being managed by STU, with the assistance of the University of Birmingham (UBHAM) and the Istituto Superiore Universitario di Formazione Interdisciplinare (ISUFI). So, we will briefly introduce the components and their functionality.

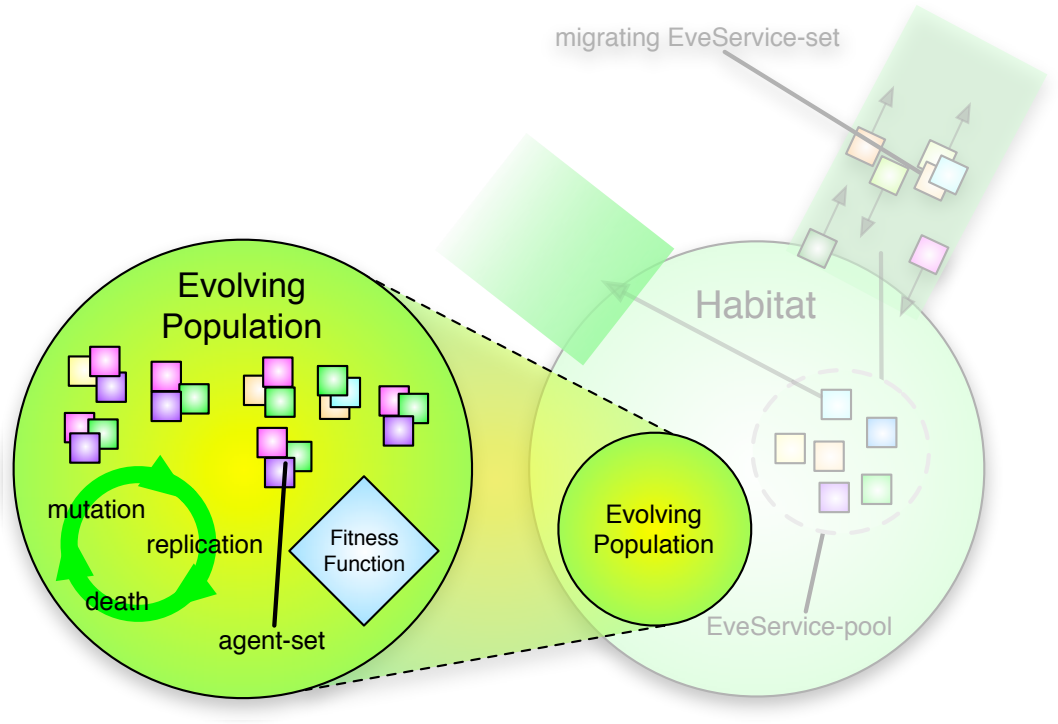


Figure 8: Evolving Population

An evolving population of EveService(-Set)s is the key component for supporting evolution within the DBE. Each evolving population is an artificial environment, having everything for evolution - a population, replication, mutation and a selection pressure (provided by the fitness function).

5.1 Population

When an Evolving Population is created to generate a solution (EveService-Set) to user request, the population is seeded with the contents of the EveService-Pool. This allows the evolutionary optimisation to be bootstrapped by solutions, found earlier, stored in the EveService-Pool.

5.2 Fitness Function

The Fitness Function will mediate the Selection Pressure required to evolve a population of EveService-Sets to the user request. The Selection Pressure guides the evolution of the population towards the desired solution (EveService-Set) by means of a Fitness Function. Over many generations, the optimal solution (EveService-Set) will be generated. Once the solution is found and used it is stored in the EveService-Pool of the user's Habitat, and then a copy is migrated to every connected Habitat conditionally on the connection probabilities.

5.3 Evolution

The evolutionary optimisation is a process that operates with the components defined. Evolution is the process that governs the adaptation of populations to their environment, involving changes to the genetic makeup of these populations. These genetic changes are passed on through successive generations[15, 11]. A population adapts over several generations through a cyclic process of mutation, replication, and death.

6 Habitats and Targeted-Migration

The Habitat is the key component for supporting the ecosystem concept within the DBE. Each SME user has a Habitat, which contains an EveService-Pool with services of potential use to the SME. The Habitat also provides a place for service migration to occur, as the Habitats are interconnected with one another. The union of the Habitats creates the Evolutionary Environment (Digital Ecosystem).

The connections between the Habitats will be self-managed, and the mechanism for this will be discussed later. A Habitat's connections will be initialised to other neighbouring Habitats. If initially located within the wrong cluster, over time it will risk becoming disconnected from the network. This will be prevented by a fail-safe mechanism, in which an SME's use of services (located through the DBE core architecture) will be used to relocate the Habitat correctly within the network.

6.1 EveService Migration

The connectivity in the Habitats will be parallel to the 'opportunity spaces' (business sector and cross business sector interaction) that exist within the SME user base. The Habitat network will adapt over time to the SME interaction. At the global level, we would expect a 'caveman' topology in which each 'cave', a cluster of high interconnected Habitats, represents an opportunity space.

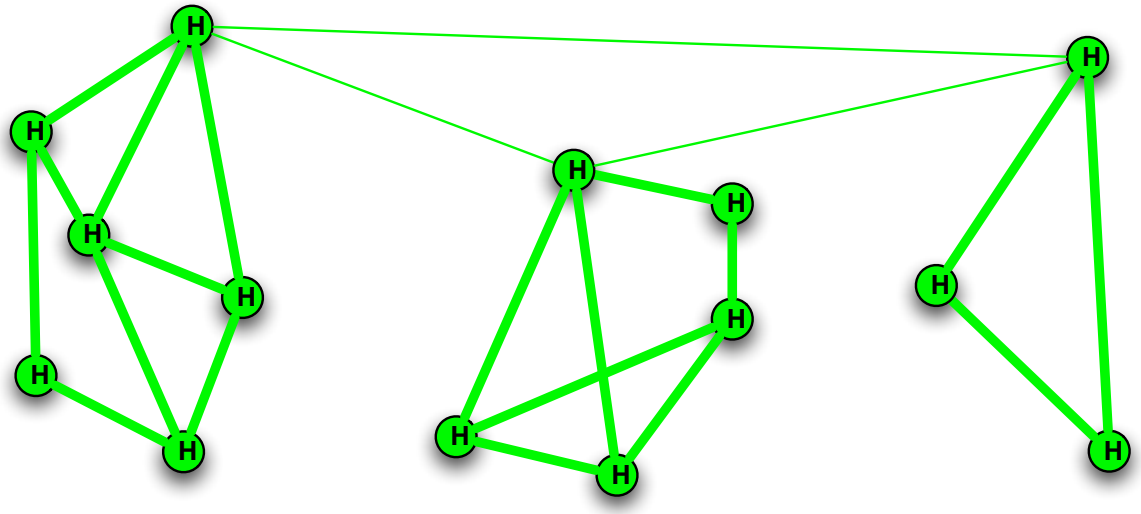


Figure 9: Habitat Network - Caveman Topology - Opportunity Spaces

The interaction and connectivity between Habitats are the migration of EveServices from one Habitat to another. Each connection between the Habitats is bi-directional, and there is a probability associated with moving in either direction across the connection. The connection probabilities affect the migration of EveServices and are updated by the success or failure of the migration. Our current understanding would suggest that the topology within the Habitat clusters (caves) will be dependent on the opportunity spaces they represent, and we expect that most of the clusters will be small-world networks. So the Habitat clusters (caves) will be highly connected with strong migration probabilities.

The migration of an EveService is triggered by its deployment to its 'Home Habitat', or when it is used at any other Habitat. When migration occurs, a copy of the EveService is made at each of the connected Habitats, dependent on the probabilities associated with the connections. These copies are initially identical, but then the MUH of each migrated EveService is updated, and this differentiates it from the original.

The success of the migration, the **migration feedback**, is determined by the use of EveServices at the Habitats to which they have migrated. When a solution (EveService-Set) is found to a user request, then the individual EveService migration histories can be used to determine where the used EveService has come from and to update the connection probabilities. The challenge is in managing the 'feedback' to the connection probabilities for migrating EveService-Sets, because for an EveService-Set the value is within the combination of services that could have migrated from different habitats before being assembled into a new set. This benefit of where the combination, or subsets of the combination, were created needs to be passed on to the connection probabilities.

The **escape range** is the number of escape migrations possible upon the risk of death. If an EveService migrates to a Habitat and is not used after several user requests, then it will have the opportunity to migrate (moved not copied) randomly to another Habitat. After this happens several times the EveService will be deleted (die). The escape range is a parameter that will be better determined by future work from the partners and practical experience. Ideally the escape range will be scaled to the size of the Habitat cluster which the EveService exists within. This is not as simple as it may sound, as the view of a cluster very much depends on the point of view of the Habitat. Clusters will have varying topologies depending on the opportunity spaces they represent. So, a home Habitat within a cluster, from its own information alone, would not know the average number of migrations required to reach any other Habitat within the cluster (i.e. the optimal number of escape migrations to specify when deploying EveServices). Temporary values will be suggested, on the premise of updating them later or creating a definition that is dynamically responsive to the state of the system.

6.2 Targeted-Migration

Although the inter-EveService interaction of the DIS (logically) occurs within the EveService-Pools, the consequence of this interaction is in the additional targeted-migration of EveServices, which can occur between connected and unconnected Habitats.

When targeted-migration occurs between connected Habitats it will speed up the natural migration of the EveServices. When it occurs between unconnected Habitats, it will help the Habitat network to adapt to changing conditions faster than it otherwise would. In effect, during targeted-migration the DIS short-circuits the hierarchical structure of the caveman topology, to avoid being unnecessarily slowed by it. This may seem counter-productive after creating such a topology, but the caveman topology is what allows the Habitat network to specialise solutions to specific users and specific user requests. This ability leads to a weakness, specifically that the Habitat network can be slow to adapt to changing conditions, which the DIS will address. The DIS will help catalyse the formation of caves, and caves with the correct nodes, within the caveman topology. The

desired effects of the DIS on the Habitat network will be achieved by integrating the targeted-migration with the existing migration feedback mechanisms.

6.3 Habitat Clustering

The Habitat network will allow for the spontaneous formation of communities via clustering within the Habitat network. Many successful service migrations will reinforce Habitat connections increasing the probability of service migration. If a successful migration occurs through multi-hop migration, then a new link can be formed between those Habitats. Unsuccessful migrations will lead to connections (migration probabilities) decreasing, until finally the connection is closed.

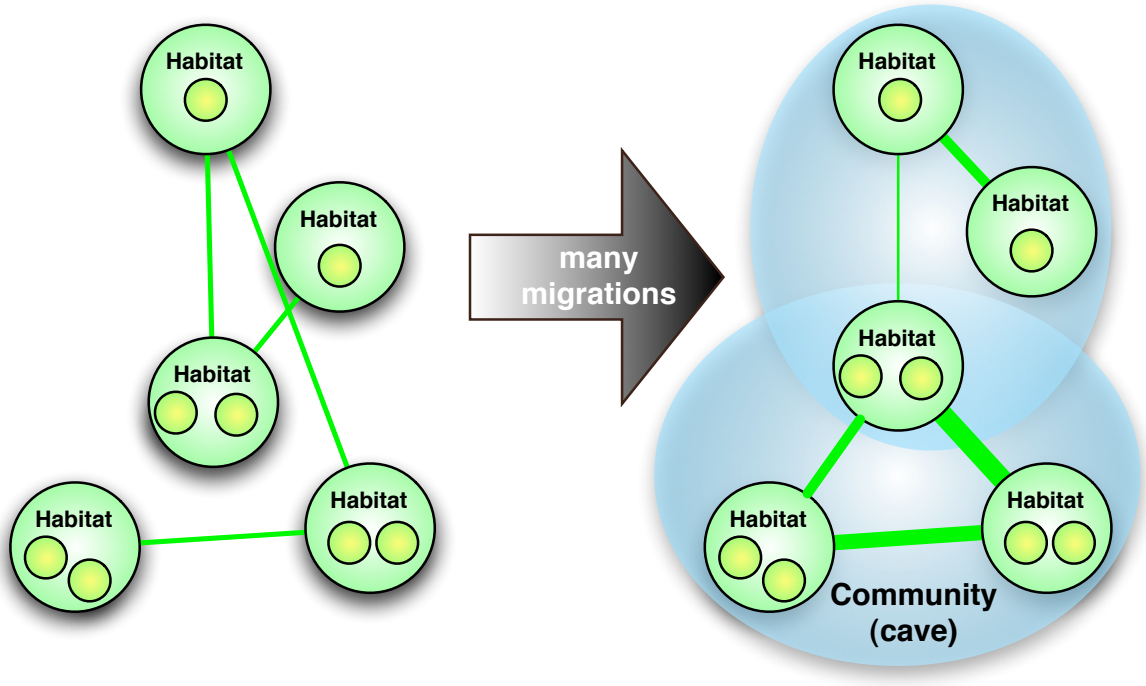


Figure 10: Habitat Clustering - Community Formation

The clustering/community formation is like joining a club; you can be a member of more than one. In the same way Habitats can be clustered into more than one community. The clustering will potentially lead to communities clustered over language, opportunity spaces, nationality, geography, etc.

6.4 Distributed Evolutionary Computing

The 'distributed' evolutionary computing component comes from evolving populations being created in response to 'similar' requests. So whereas traditionally there are multiple evolving populations in response to one request, here there will be multiple evolving populations in response to similar requests. This is shown in Figure 11 where the colour of the evolving populations indicates similarity in the requests being managed. For example, the four Habitats in the left of Figure 11 may all be travel agents, the group of three with

evolving populations may be looking for package holidays within the same continent. A 'great deal' solution (EveService-Set) found and used in one Habitat, will be migrated to the other connected Habitats where it will be integrated into any evolving populations via the local EveService-Pool. So this will potentially help to optimise the search of similar package holidays at the Habitats of the other travel agents. This will also work in a time-shifted manner, because the 'great deal' solution (EveService-Set) will be stored in the EveService-pool of the Habitats to where it is migrated. So it will potentially be available to optimise a similar request placed later.

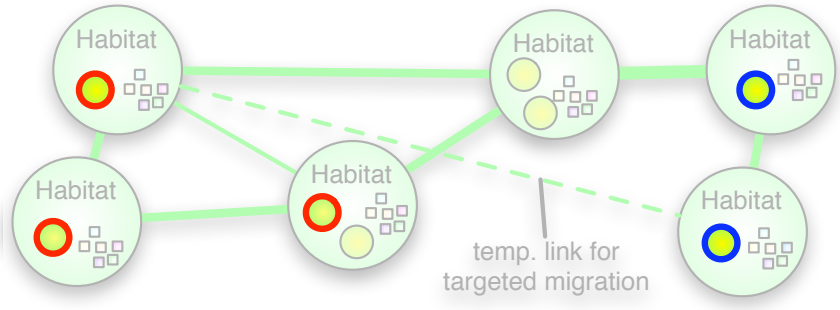


Figure 11: Distributed Evolutionary Computing in the EvE Digital Ecosystem

This is based on the premise that once the DBE has sufficient 'critical mass', in terms of the SME user base, there will then be similarity in the requests of SMEs within the same opportunity space (business sector).

6.5 Fail-Safe Mechanism

A Habitat may become disconnected or located in the wrong cluster. This will most likely occur when a Habitat is initially connected to the wrong cluster to begin with, when the facilities provided are not used by the user, or the SME moves into a new opportunity space (changes business portfolio). If the EveService-Pool is insufficient to find even a single solution, then the Habitat is probably incapable of finding an optimal solution even in the long-term over several requests, showing the Habitat to be located within the wrong cluster. If the system initially fails to connect the Habitat to the correct cluster in the first place, then it is doubtful it would succeed a second time. These scenarios can be resolved by making use of the user's interaction with the core architecture. Solutions provided by the Recommender can be used to connect the, wrongly connected or unconnected, Habitat to the correct cluster.

6.6 Relation to DBE Network Architecture

Consideration has been given to integrating the EvE Digital Ecosystem with the DBE core architecture by making each Habitat a core ExE service. For each SME, whether producer or consumer, their Habitat service will be located on their DBE servent application.

The evolving populations, instantiated in response to user requests, are contained within the Habitats. A Habitat service is created when an SME joins the DBE. The connections

between the EvE Habitats in Figure 12 represent the bi-directional sharing of EveServices between the Habitats.

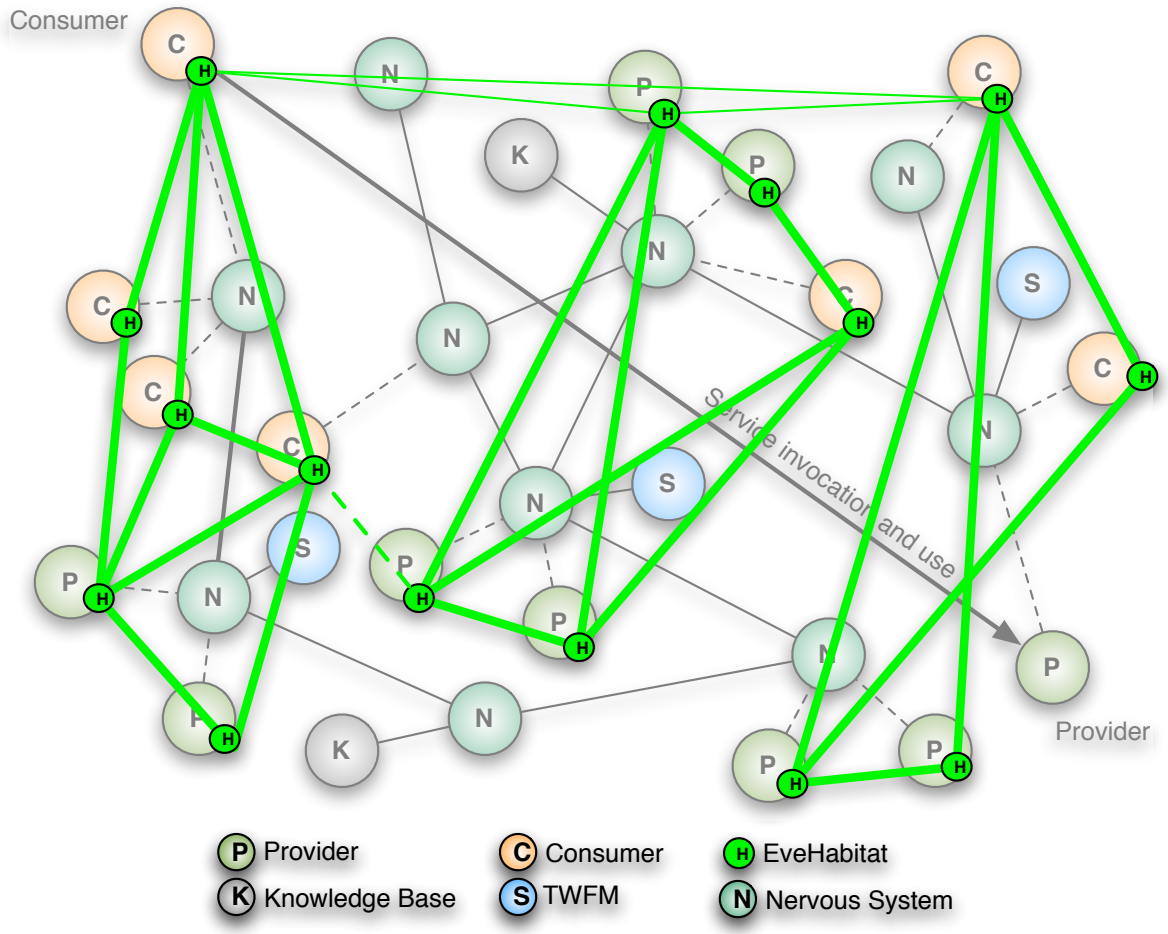


Figure 12: EvE Digital Ecosystem relation to DBE Network Architecture

The Habitats can be run anywhere, just as in grid computing where the user is blind to where the processing occurs. A distributed processing arrangement, where supernode DBE servents handle several Habitats for several SMEs should be possible. The EvE will be distributed within the DBE network as shown in Figure 12.

7 EvE/DIS Architecture Requirements

The EvE will exist within the ExE. The Habitat will be a core service located upon the DBE servent. The other EvE components will exist within this Habitat service.

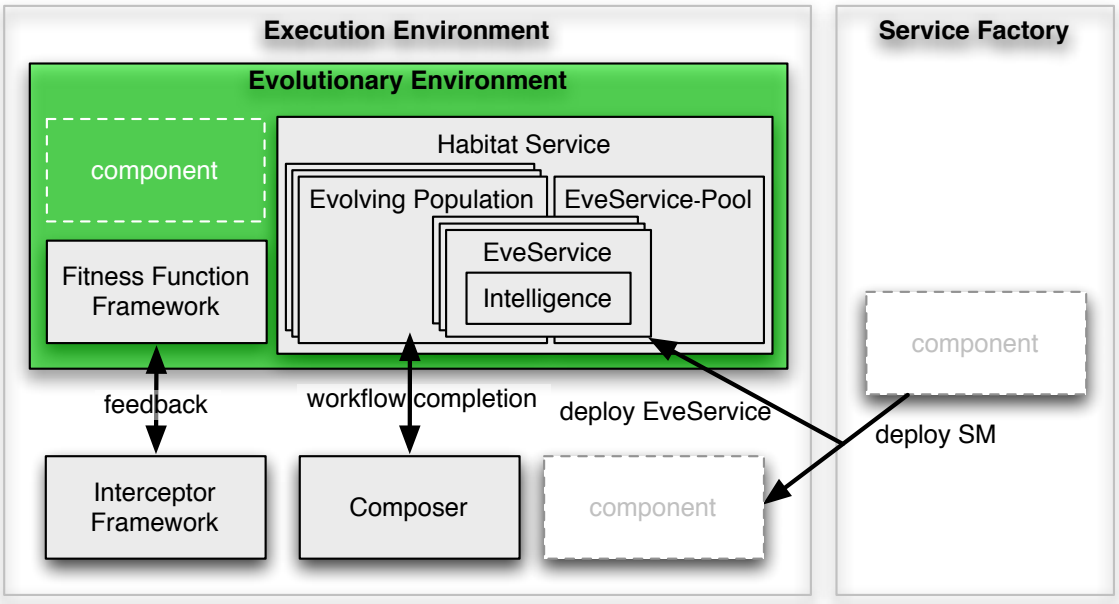


Figure 13: Evolutionary Environment Components

In Figure 13, the key interactions from the EveService life-cycle (Figure 6) are shown.

7.1 Structural Requirements

7.1.1 EveService and Intelligence

EveServices and EveService-Sets are both individuals within the EvE, whose genotypes are their SBVR descriptions (Service Manifests). The SM-IDs of the services they represent, and data structures for their MUHs, will be required for the desired functionality.

EveService
An EveService's primary function is as a pointer to the SBVR model of the service it represents, for both the evolutionary optimisation of the EvE and the distributed optimisation of the DIS. However, a considerable amount of additional structures is required to support this behaviour.

- SM-ID (unique identifier)
- home Habitat-ID (habitat of owner)
- MUH (event history)
  - migration event (from one Habitat to another)

        * habitat (where migrated from)
    - usage event (at a Habitat)
        * habitat (where used)
        * fitness (when used)

- failure counter (number of times failed to be used in response to a user request)
- escape counter (number of times can migrate away to avoid death)
- Intelligence
- DIS migrations counter (number of migrations for DIS behaviour)

EveService-Set
The requirements are similar to the EveService, with the noticeable exception of the Intelligence component. When an EveService is created, it becomes a structure holding two or more EveServices.

- aggregate of multiple SM-IDs (unique identifier)
- home Habitat-ID (habitat where created)
- Migration and Usage History (MUH) (event history)
    - migration event (from one Habitat to another)
    - usage event (at a Habitat)
- failure counter (number of times failed to be used in response to a user request)
- escape counter (number of times can migrate away to avoid death)

The *home Habitat-ID* defines where the EveService-Set would have been created, which occurs when the EveService-Set solution (when found in response to a user request) is stored in the EveService-Pool. So, the EveService-Set and the *home Habitat-ID* define for any EveService of the set, which other EveServices it was used and migrated with.

The MUHs of the EveServices within an EveService-Set must also be updated when the MUH of the EveService-Set is updated. This is necessary because the individual EveServices may become separated from the EveService-Set, or combined in another EveService-Set. This will ensure complete MUHs for the EveServices, vital for the fitness estimation within evolving populations, the Habitat clustering mechanism, and the migration targeting of the DIS.

Intelligence
The major addition of the DIS will be the Intelligence component, which will encase the NN for similarity recognition between the SBVR descriptions of services.

The size of the *input layer*, the number of neurons, is proportional to the SBVR description of the agent in which the NN is embedded. Generally, a single *hidden layer* is sufficient, and depending on the desired functionality, is usually significantly larger than the *input layer* [13]. The *output layer* consists of a single neurone, with values between zero and one, to be used in providing an answer to the question of similarity.

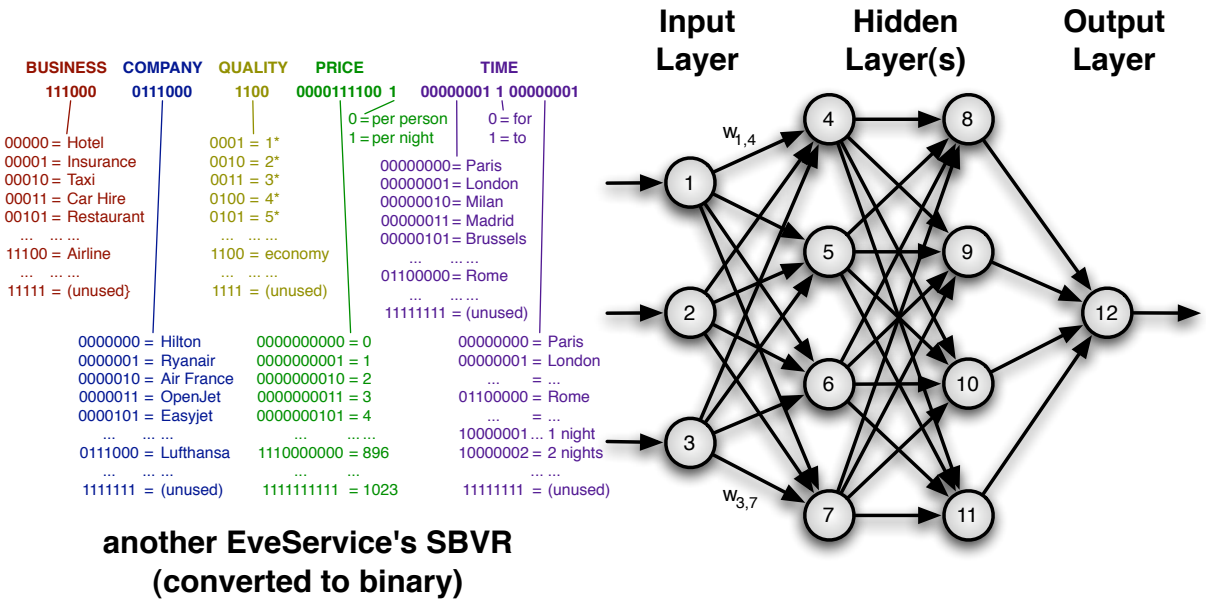


Figure 14: NN Structure for the Intelligence Component

The weights, $w_{i,j}$, between neurones are randomly initialised and then trained to the *real* numbers that provide the desired functionality. We are currently using a sigmoid transfer function, in which the output, y, of neurone is the sum of the weighted input values, x, applied to a sigmoid function. This function is expressed mathematically as:

$$y = \frac{1}{(1 + e^{-x})} \tag{1}$$

This has the effect of smoothly limiting the output within the range of 0 and 1. Our simulations currently use a threshold of 0.90, for the neurone of the *output layer*, to determine similarity. This value may need to be proportioned to the SBVR vocabulary of the Habitat cluster (opportunity space) and/or the experience of the EveServices, as it defines how focused the targeted-migration will be. So, the Intelligence component will also require an *experience counter*.

Our research is currently investigating the optimal NN to be used, and we cannot currently provide a final recommendation. However, we are currently experimenting with multilayer feedforward networks, as they are known for their wide applicability [13], and because they have proven successful in our preliminary results (please see the Appendix).

The greatest challenge of the DIS implementation will be the pre-processing of service SBVR descriptions to be used with the NNs, in the same way that the greatest challenge of the EvE is the processing of the service SBVR descriptions for fitness determination.

The pre-processing will require an ordering of the properties within an SBVR description [17]. This will then need to be encoded in binary for processing by the NNs.

7.1.2 EveService-Pool and Inter-EveService Interaction

The Habitat EveService-Pool is the set of EveService(-Set)s available at a Habitat. It will vary between the set of atomic services available at the Habitat, and the power set of the available atomic services.

The inter-EveService one-on-one interaction will not require any additional structures to be added to the EveService-Pool. So, the EveService-Pool needs only to be a vector to fulfil the architectural requirements. However, it may become more sophisticated, depending on the approach of the implementation.

7.1.3 Evolving Population

Currently, the Evolutionary Computing in Java (library) (ECJ) is being used to provide the necessary functionality for the evolutionary process and the population structures. So, there are no specific structural requirements that need to be stated.

The more significant challenge of interfacing the SBVR with the Fitness Function Framework (FFF) is being managed by STU.

7.1.4 Habitat Service and Targeted-Migration

The Habitat service needs to store the EveServices contained within, and the connection probabilities to other Habitats.

- Habitat-ID (unique identifier - could be the User-ID)
- connection probabilities (tuples of Habitat-ID and migration probability)
- request history (including the solution)
- EveService-Pool (holds all EveService(-Set)s within the Habitat)
- active evolving populations

Depending on how the Habitat network is initialised, specifically if many Habitats are turned on simultaneously instead of incrementally, it may be necessary to have a bootstrap mode. In this mode, the Habitats would temporarily share sub-optimal solutions evolved to the user requests. Otherwise, the Habitat network would be entirely dependent on the fail-safe mechanism, which may not be sufficient to compensate on such a large scale. The temporary sharing of sub-optimal solutions will allow the EvE Digital Ecosystem to bootstrap naturally, like a biological ecosystem.

7.2 Behavioural Requirements

This subsection aims to provide the behaviour of the EvE and the DIS in more detail than described previously. The behavioural specification of the EvE has been improved since the EvE Architecture Requirements document [2], and the interaction with the DIS has also been included.

This behavioural specification is where the separation between the EvE and the DIS can be seen more clearly. They have so far required the same infrastructure, data structures and access to the same data. The behaviour of the EvE and DIS can now be separated, because the EvE depends entirely on events from the user to operate, whereas the DIS depends indirectly on events from the user and directly on self-generated events. Within the behavioural specifications of the EvE, the DIS appears in three different ways. Firstly, when an EveService is deployed, the Intelligence component and its NN need to be initialised. Secondly, when a usage event is created, for the MUHs of an EveService, the 'DIS migrations counter' needs to be incremented. Finally, when an EveService(-Set) is added to an EveService-Pool, the DIS needs to be activated.

The numerical parameters which specify the conditionals in Figures 15 through to 19 are based on our simulation, and so are tentatively proposed. They will be better informed by future work from ourselves, the partners and practical experience.

The following figures (sequence diagrams) have been significantly reduced in size. If viewing the PDF then please zoom in, as the figures have sufficient resolution. Alternatively, full size versions of the figures can be found at www.iis.ee.ic.ac.uk/ g.briscoe/D6.6/.

The following conventions are followed in the sequence diagrams; a '*' indicates iteration, while '[' and ']' encase conditionals. Where possible the entire loop of an iteration is indicated with a '*', else just the conditional of the loop is indicated with a '*'.

The sequence diagrams are not expected to be implemented literally. They are intended to help clarify the interactions, and so help the software developers understand the logical interactions that need to occur within their implementation of the DIS.

7.2.1 EvE : New User

When a new user joins the DBE, the user's Habitat needs to be created and most importantly connected to the Habitat network. This scenario is not affected by the DIS, and is shown in Figure 15.

7.2.2 EvE : Deploy Service

When a user deploys a service to the DBE, it must also be deployed as an EveService to their Habitat. The SM-ID of the new DbeService is required to create the new EveService and its NN-based Intelligence. It is then copied to the EveService-Pool of the user's Habitat. From there the migration of the EveService occurs, which involves migrating (copying) the EveService probabilistically to all the connected Habitats. The copying of

an EveService to a connected Habitat depends on the associated migration probability. If the probability were one, then it would definitely be sent. This scenario is shown in Figure 16.

7.2.3 EvE : Fail-Safe

A Habitat can become totally disconnected under certain conditions. Firstly, if the EveService(-Set)s within the EveService-Pool consistently fail to satisfy user requests. Secondly, if the user's shared services should be undesirable. Under these conditions the Habitat could be located within the wrong cluster, and if so the EvE would initially fail to produce satisfactory results.

When a user finds and uses a service via the Recommender, it can be used as a fail-safe mechanism for the EvE. It can be used to refresh the inter-Habitat connections of the user's Habitat, by creating an EveService in the user's Habitat to represent the service, and with that establishing new inter-Habitat connections. So, the fail-safe mechanism will be able to correctly relocate a Habitat not located within the proper cluster. This assumes there is sufficient critical mass in the SME user base, such that there are viable Habitat clusters to join. The DIS plays no part in this mechanism, which is shown in Figure 17.

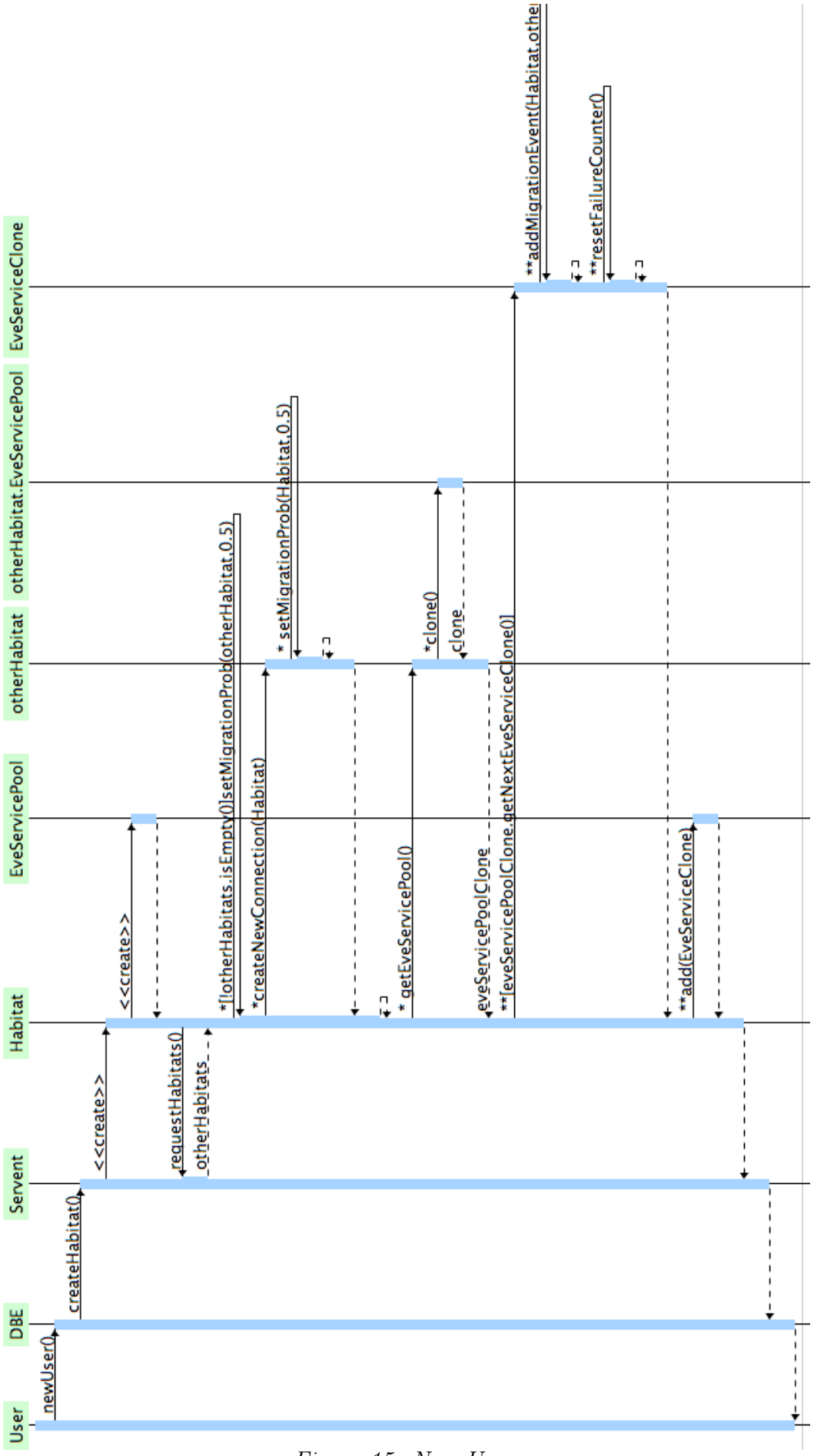


Figure 15: New User

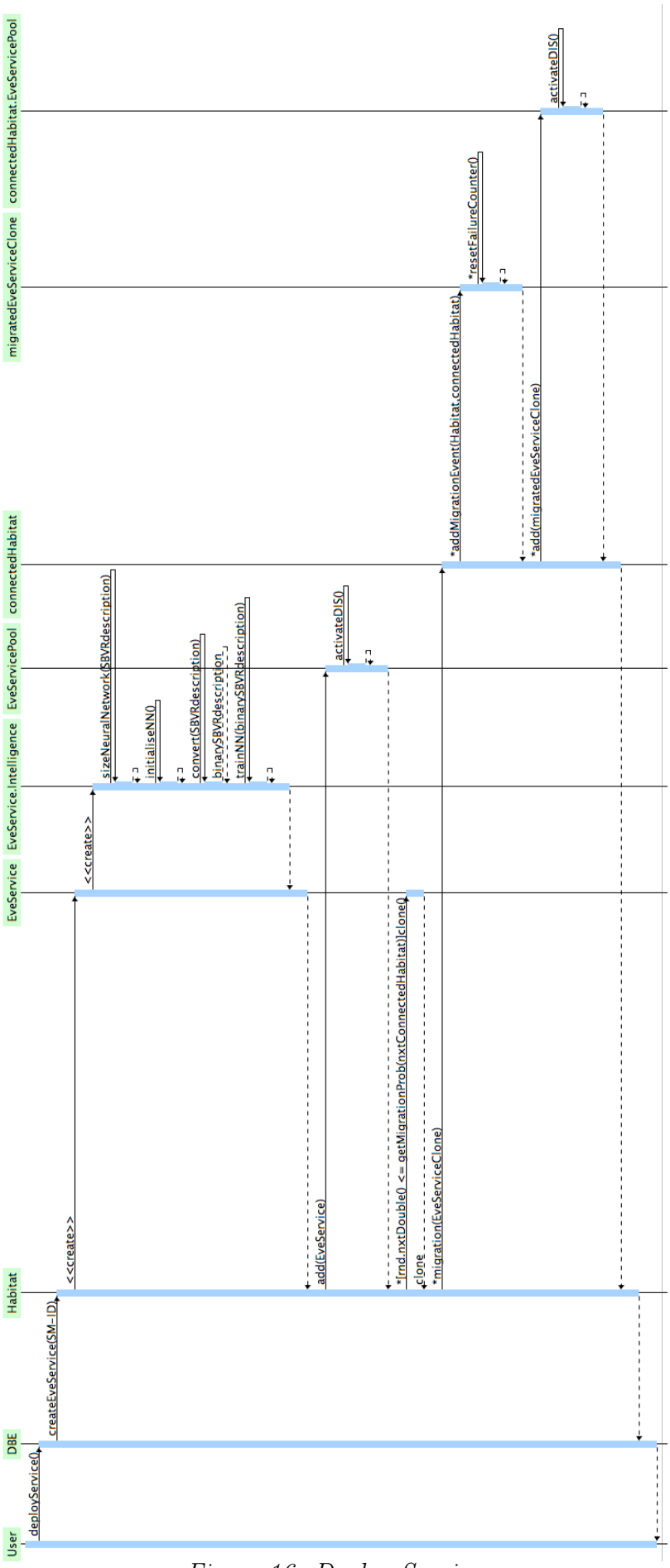


Figure 16: Deploy Service

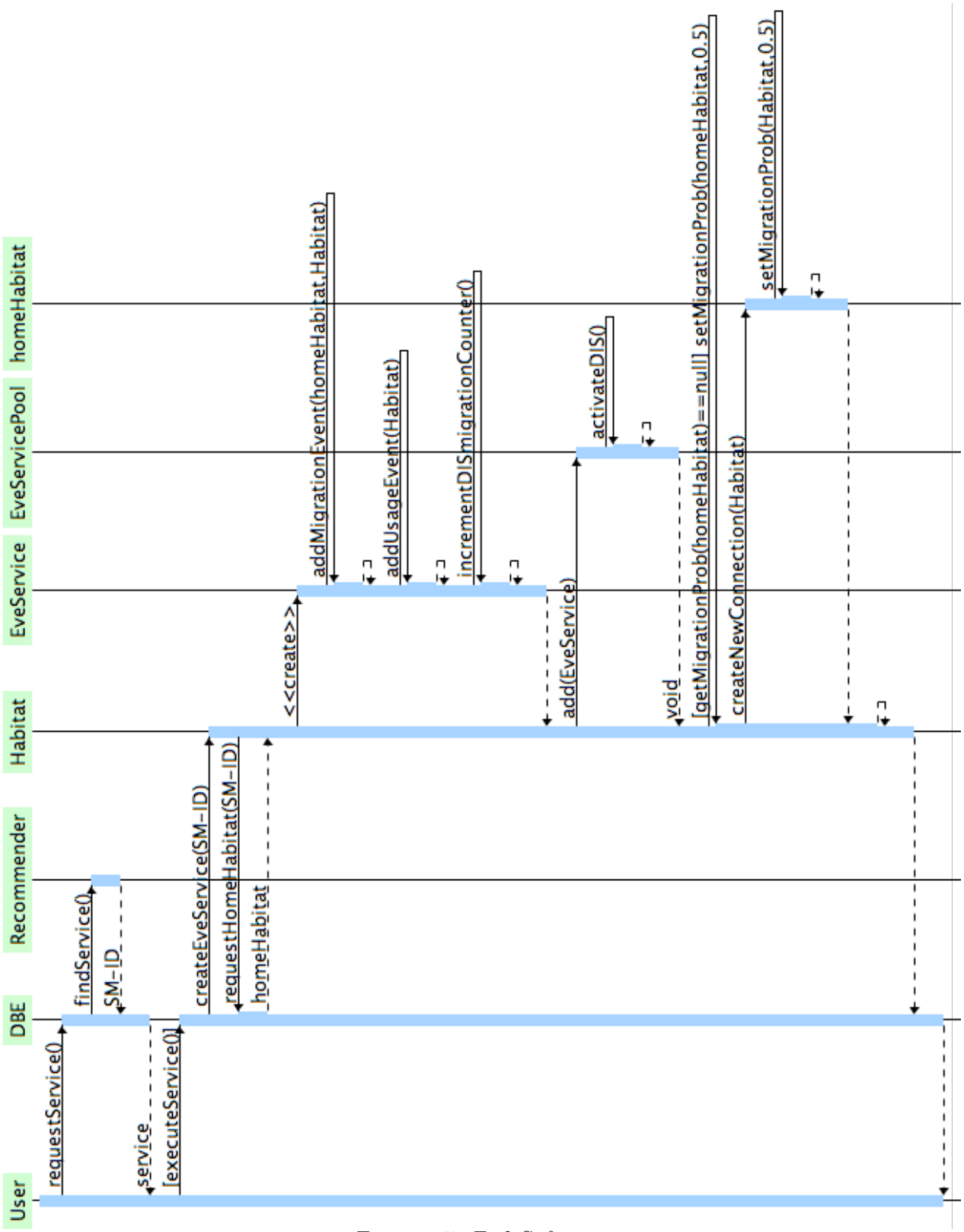


Figure 17: Fail-Safe

7.2.4 EvE : User Request for Services

When a user requests a service(s) from the DBE, the request is passed to their Habitat where it will create an evolving population to find the optimal solution. This involves generating a fitness function from the request and then allowing an evolutionary process of mutation, replication and death to occur until the optimal solution is found. This scenario is shown in Figure 18.

If a user chooses to execute the EveService(-Set) optimal solution, they will have to complete the workflow using the Composer. The execution of a service is the trigger for the EveService(-Set) migration, the Habitat clustering mechanism, and one of the triggers for the DIS one-to-one inter-EveService interaction.

7.2.5 DIS : EveService Addition to EveService-Pool

Whether through deployment or migration, when an EveService(-Set) is added to an EveService-Pool, this will under most circumstances trigger the DIS core behaviour. This core behaviour, the one-to-one inter-EveService interaction, is shown in Figure 19.

The EveService, being added to an EveService-Pool, will trigger the comparison of itself to the other EveServices within the EveService-Pool. This involves processing the SBVR descriptions, through its embedded NN-based intelligence, to check for similarity. Upon finding similarity, the EveService will visit the best Habitat (most likely to be used) from the MUH of the EveService it is interacting with.

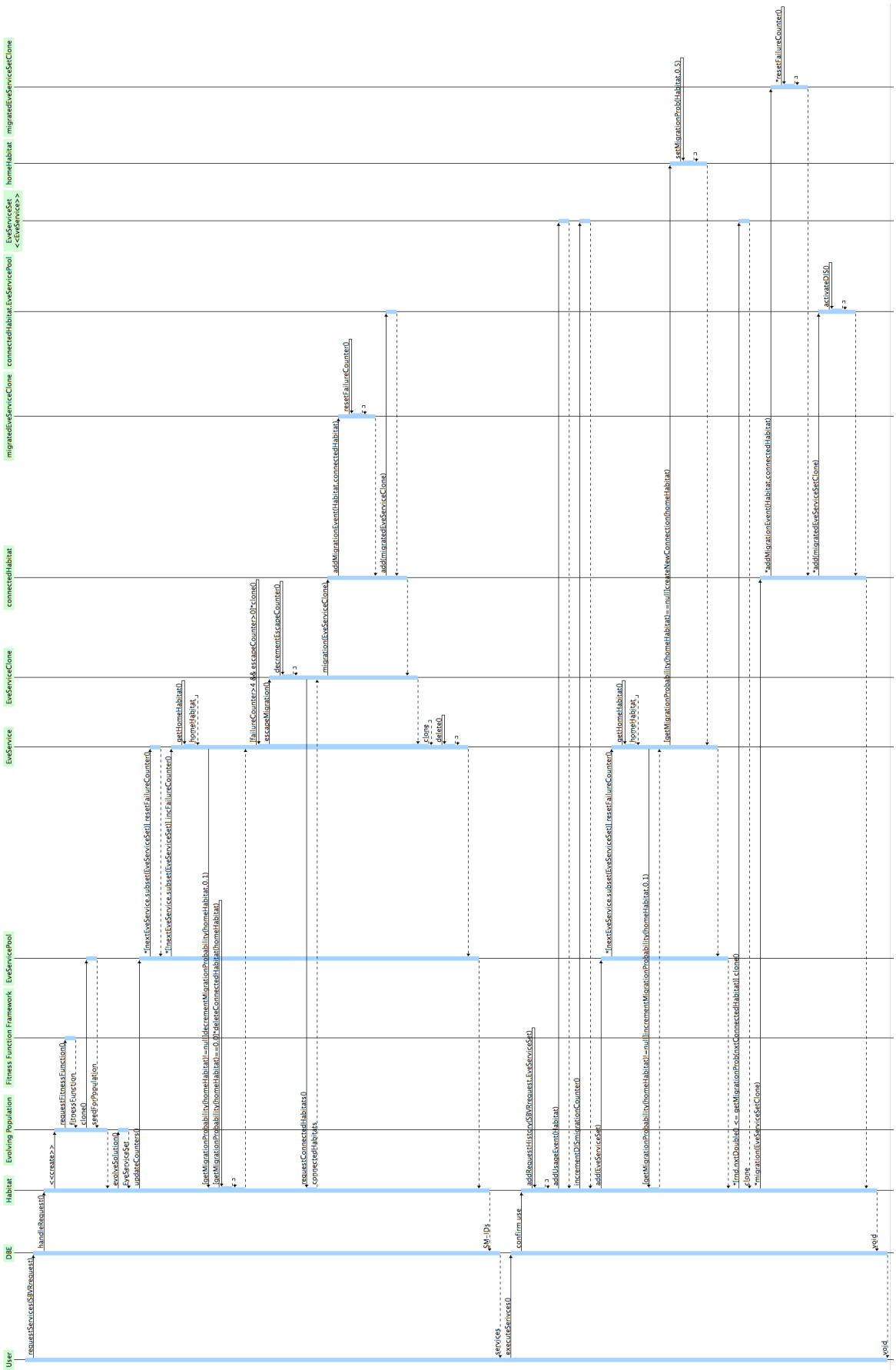


Figure 18: User Request

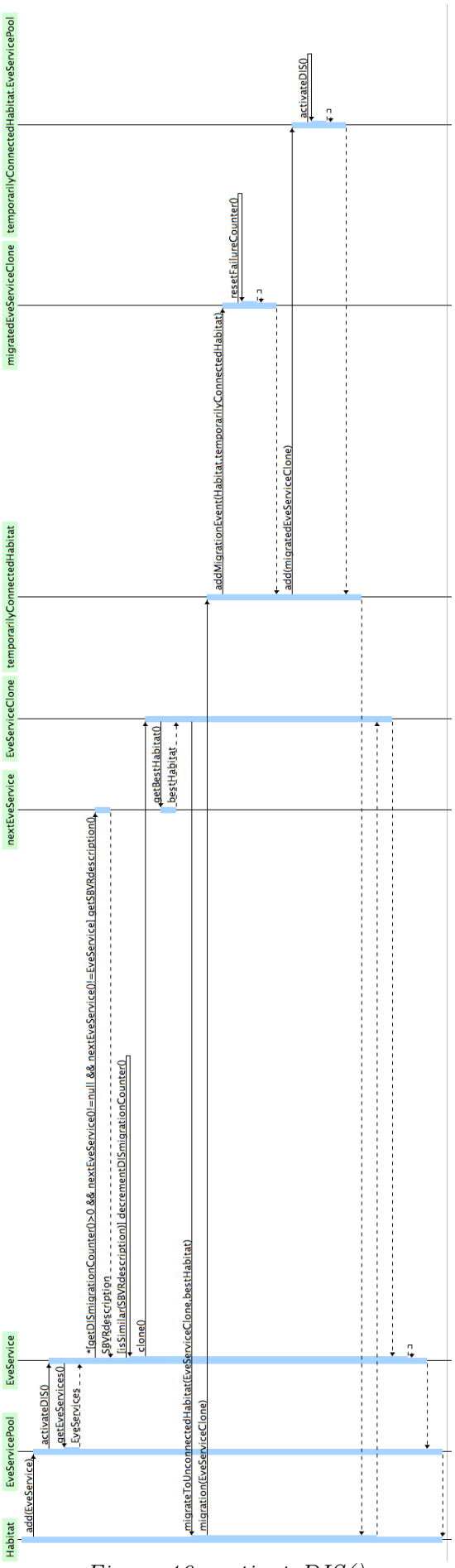


Figure 19: activateDIS()

8 Conclusion

8.1 Achievements

The *distributed intelligence* model has been improved and finalised. The *targeted-migration* approach chosen for the *distributed intelligence* model, is an improved version of the *directed-migration* approach. The *directed-migration* approach was presented and recommended, above the other alternative approaches, in our previous deliverable D6.4 [5].

The *distributed intelligence* model has been adapted to a requirement specification for the high-level architecture of the DIS.

The architectural requirements of the EvE have been finalised, made possible by our recent simulations, and the EvE implementation effort led by STU.

The DIS is not only compatible with the EvE, assured by the compatibility of the *distributed intelligence* model with the Digital Ecosystem model, but constructively interacts with the ecosystem dynamics of the EvE.

The preliminary research results for the *distributed intelligence* model, presented in the Appendix, are very promising and justify the research continuing in its current direction.

The value of these contributions to the DBE is that it provides a completed, and now optimised, EvE platform which uniquely differentiates us from the competition [1].

8.2 Limitations

The only technique considered to provide a learning capability for similarity recognition is NNs. The EvE is not a traditional straight-forward genetic algorithm, so there are many potentially effective techniques, but they are all untried. We chose NNs to add the learning capability because it is a well-established technique, and considering the limited timeframe, our main experience is with NNs. For the same reason we chose to start our simulations with multilayer feedforward NNs, because they have been shown to be a widely applicable proven technique.

The architecture of the DIS is modular, so if the implementation of the DIS with a NN-based intelligence proves not be viable because of the pre-processing limitations with SBVR, there is no reason an alternative technique could not be used. The FFF [14] may provide the best alternative, as it is being constructed to compare SBVR service descriptions with the SBVR user requests for services. Adopting such an approach may reduce the learning capacity within the DIS, but this is the potential risk of dealing with the SBVR.

8.3 Relation to Project Activities, WPs and Tasks

We will interact with the task C53, Implementation of the Distributed Intelligence System, which will be integrated with the EvE implementation and DBE core architecture, collaboratively with STU and SUN [7]. The task C42 and this deliverable are intended to provide the fundamental objects and desired behaviour of the DIS, and then communicate them to INTEL for DIS's implementation in C53. INTEL has already been involved with the transfer of the scientific Digital Ecosystem model, for the computing implementation of the EvE. They also hosted a meeting to discuss the EvE and DIS integration, in which we introduced the model of the DIS from deliverable D6.4 [5], and they assisted with the implementation of the EvE. They are also reviewers of this deliverable, all of which will smoothen the knowledge transfer for the implementation of the DIS, and we will continue working with INTEL to support the implementation process.

We will also continue with the task C42, to interact with STU and SUN, ensuring that the Digital Ecosystem model of the EvE is integrated optimally and faithfully into the DBE Core Architecture.

We will also be continuing with the task S5, investigating the optimal kind of NN and the optimal NN structure for the *distributed intelligence* model, with the potential application to DIS. We will also consider SVM for the DIS, to address the limitations explained in the previous subsection.

8.4 Progress and Future Work

Regarding the objective of 'the research, design and implementation of a Distributed Intelligence System (DIS) based on intelligence based on self-organisation to optimise and enhance the Evolutionary Environment (EvE)' [8], we have so far partially completed the research, which is continuing in task S5, sufficiently for the architectural specification of the DIS in task C42, to support the implementation process in C53. We will continue with the task C42, in supporting task C53 in achieving the objective of an 'implementation of a DIS which optimises the EvE' [8].

Regarding the objectives of 'synchronisation with the science and computing streams to ensure that the scientific outputs are channelled appropriately into the existing high-level design of the EvE' [8] to 'ensure that the EvE high-level design is integrated optimally and faithfully into the DBE Core Architecture, upon which the DIS is dependent' [8], is shown by the implementation of the EvE, led by STU with the assistance of SUN and INTEL, remaining faithful to the Digital Ecosystem model.

Regarding the objectives of 'the dissemination effort to distribute and explain our results to the computation work package contributors in C53 via C42' [8], the 'creation of a high-level design specification for the Distributed Intelligent System which augments the EvE, for use in the implementation of the Distributed Intelligence System' [8], 'finalisation of the EvE Architecture specification' [8], and 'a design of the DIS which is complementary to the EvE' [8], this deliverable is explicitly provided to meet these objectives and to complement our dialogue with the partners involved.

Regarding the research objectives of whether 'intelligence can optimise the evolutionary process' [8], 'how does this distributed intelligence interact with the ecosystem dynamics' [8], and whether 'software components that are part of genetic selection can be intelligent in themselves' [8], is partially answered by the now finalised *distributed intelligence* model and the preliminary results presented in the Appendix. The *distributed intelligence* model optimises the Agent-Pool at each habitat to support the evolving populations created to respond to user requests, thereby creating a distributed optimisation to support the local evolutionary optimisation. The objectives are only partially answered, because although the theoretical justification exists, the results presented in the Appendix are only preliminary. So we will be continuing with task S5, further investigating the *distributed intelligence* model, to meet fully these objectives. This will include investigating SVM for the DIS, to address the limitation of only a single technique being considered for the intelligence component of the DIS.

8.5 Summary

The *distributed intelligence* model has been finalised with the adoption of the *targeted-migration* approach, for which the preliminary simulation results are promising. The *distributed intelligence* model has been adapted to an architectural specification for the DIS, which has been integrated with the architecture of the EvE, which itself has been finalised. So, the engineering process of the DIS can begin and the *distributed intelligence* research will continue.

References

[1] J Aparicio. D2.3 Software Roadmap. *Digital Business Ecosystem*, Contract no 507953, 2006.

[2] G Briscoe. Evolutionary Environment Architecture Requirements. *Internal*, April 2005. Available from: `http://www.iis.ee.ic.ac.uk/~g.briscoe/Eve/EveArchRequirements.pdf`.

[3] G Briscoe and P De Wilde. D6.1 self-organisation in multi-agent systems. *Digital Business Ecosystem*, Contract no 507953, 2004. Available from: `http://www.digital-ecosystem.org/Members/aenglishx/linkstofiles/deliverables/Del_06.1_DBE_Self-Organisation_In_Multi-Agent_Systems.pdf`.

[4] G Briscoe and P De Wilde. D6.2 control of self-organisation and a performance measure. *Digital Business Ecosystem*, Contract no 507953, 2005. Available from: `http://www.digital-ecosystem.org/Members/aenglishx/linkstofiles/deliverables/Del_06.2_DBE_Control_of_Self-Organisation_and_a_Performance_Measure.pdf`.

[5] G Briscoe and P De Wilde. D6.4 intelligence, learning and neural networks in distributed agent systems. *Digital Business Ecosystem*, Contract no 507953, 2005.

[6] G Brisoce, J Rowe, and P Dini. Evolutionary Environment Discussion Paper. *Internal*, 2004.

[7] P Ferronato. D21.2 architecture scope document. *Internal*, 2005. Available from: `http://www.digital-ecosystem.org/Members/aenglishx/linkstofiles/deliverables/Del_21.2_DBE_Core%20Architecture%20Scoping%20Document_Release%20C.pdf`.

[8] M Giorgetti, P Dini, and A Nicolai. Deliverable D1.2, Detailed Work-Plan for the second phase. *Internal*, WP6 description, 2005.

[9] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[10] D Green and S Sadedin. Interactions matter: complexity in landscapes and ecosystems. *Ecological Complexity*, 2004.

[11] W G Hale, J P Margham, and V A Saunders. *Dictionary of BIOLOGY*. HarperCollins Publishers Inc., 2 edition, 1995.

[12] J T Heaton. Programming neural networks in java. *Java Developer's Journal*, 7(5), 2002. Available from: `http://jdj.sys-con.com/read/36985.htm`.

[13] J T Heaton. *Introduction to Neural Networks with Java*. Lightning Source UK Ltd, 2005.

[14] T Heistracher, C Masuch, T Kurz, and G Marcon. Towards a fitness function for matching SBVR with genetic algorithms optimization. *Internal Report*, 2005.

[15] A Redmore and M Griffen. *Longman Reference Guides: Biology*. Longman Group Limited, 7th edition, 1994.

[16] Jonathan E. Rowe and Boris Mitavskiy. D8.1 Report on evolution of high-level software components. *Internal*, 2005. Available from: `http://www.digital-ecosystem.org/Members/aenglishx/linkstofiles/deliverables/Del_08.1_DBE_Report%20on%20Evolution%20of%20High-Level%20Software%20Components.pdf`.

[17] M De Tommasi. D15.3 BML framework 2nd release. *Digital Business Ecosystem*, 2005.

A Preliminary Research Results

The following section summarises our preliminary research results with regards to simulating the *distributed intelligence* of the DIS. It was felt unwise to produce an architecture requirement specification for the DIS without at least some experimental evidence of its effectiveness, as it has so far been well-informed conjecture.

A.1 Predictions

Although general statements about ecosystem complexity over time can be made, more detailed explanations can easily become a matter of debate. Not only are we dealing with an ecosystem, but a Digital Ecosystem with its own unique properties, so it was felt wise to contact a colleague with experience in ecological complexity. Dr Suzanne Sadedin [10] kindly offered to assist me and explain the relevant theoretical biology.

There are several scenarios in which the DIS can optimise the EvE. Including changing conditions in the DBE, and the bootstrapping scenario of the EvE. The bootstrapping of the EvE involves activating many Habitats simultaneously, and allowing them to optimise their connectivity and Agent-Pools, through the ecological process of succession.

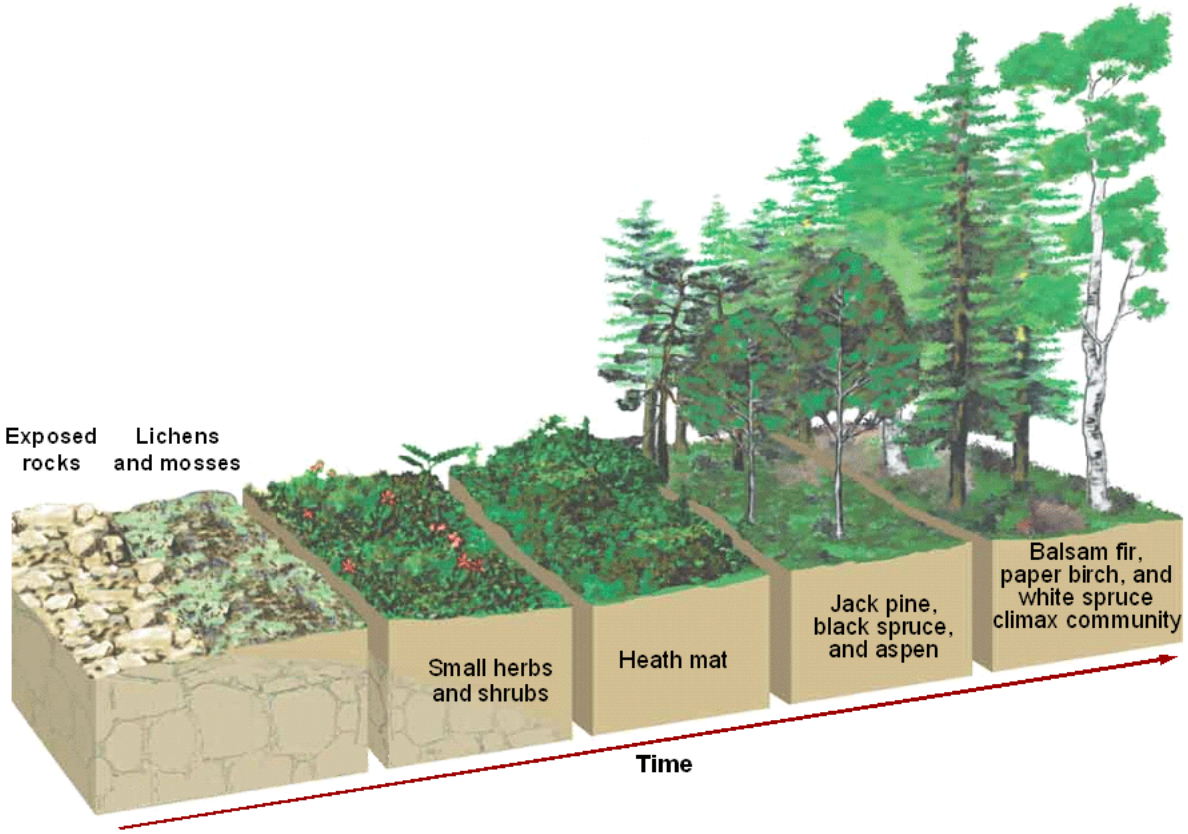


Figure A.1: Ecological Succession

We considered existing theories of complexity in biological ecosystems and how it would apply to the EvE Digital Ecosystem. We looked for a high-level understanding that would apply equally to both biological and digital ecosystems. The complexity has to increase initially or there would not be an ecosystem, and presumably this increase eventually

stops, because there is a limit to how many species can be supported. The period in between is more complicated.

If we consider the neutral biodiversity theory, which basically says network aspects of ecosystems are negligible, we would probably get a relatively smooth progression, because although you'd get occasional extinctions, they would be randomly isolated events whose frequency would eventually balance arrivals, not self-organised crashes like in systems theory. In systems theory, when a new species arrives in an ecological network, it can create a positive feedback loop that destabilises part of the network and drives some species to extinction. Ecosystems are constantly being perturbed, so it is reasonable to think that a species that persists is most likely to be involved in some type of stabilising interaction with other species. So, the whole ecological network evolves to resist invasion. That would lead to a spiky succession process, perhaps getting less spiky over time.

So, which theory is more applicable to a Digital Ecosystem depends on the extent that species in the ecosystem behave as independent, competing entities (smooth succession) versus tightly co-adapted ecological partners (spiky succession). Our Digital Ecosystem despite its relative complexity, is quite simplistic compared to a real biological ecosystem. We have the essential and fundamental processes, but no sophisticated social mechanisms. The DIS will be the first such mechanism. Therefore, currently the neutral biodiversity theory with a smooth succession is more probable. The effectiveness of responses to user requests will be used as an estimate of complexity, from all habitats within the Digital Ecosystem. The DIS can optimise the EVE by helping the agents (EveServices) to find their niches within the ecosystem. Specifically, the DIS should help the EvE to adapt faster. This has all been summarised graphically in Figure A.2.

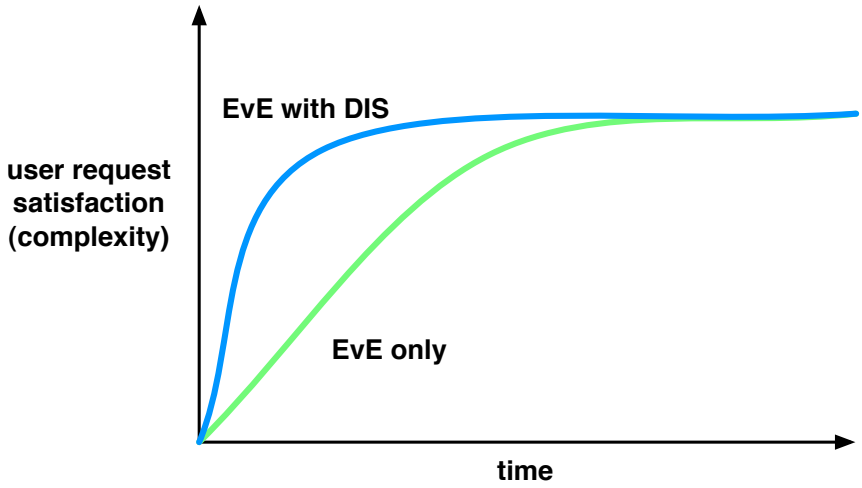


Figure A.2: EvE Digital Ecosystem Complexity Over Time

A.2 Simulation

The global behaviour predicted will also be visible within a single Habitat cluster. A simulation with 100 SMEs and their Habitats was constructed. The SMEs produce services and requests, and are unable to satisfy their own requests. The range and diversity of requests and agents were constrained to average at 60%, so that the full range of results

could be seen. The programming proved to be a significant task, as simulating the DIS required much of the existing EvE simulation to be rewritten, primarily because it had to be changed from a single Habitat simulation to a multiple Habitat simulation.

A.3 Results

The EvE performed as expected, adapting and improving over time to reach the 'climax community' (in biological terminology). As can be seen by the graph in Figure A.3.

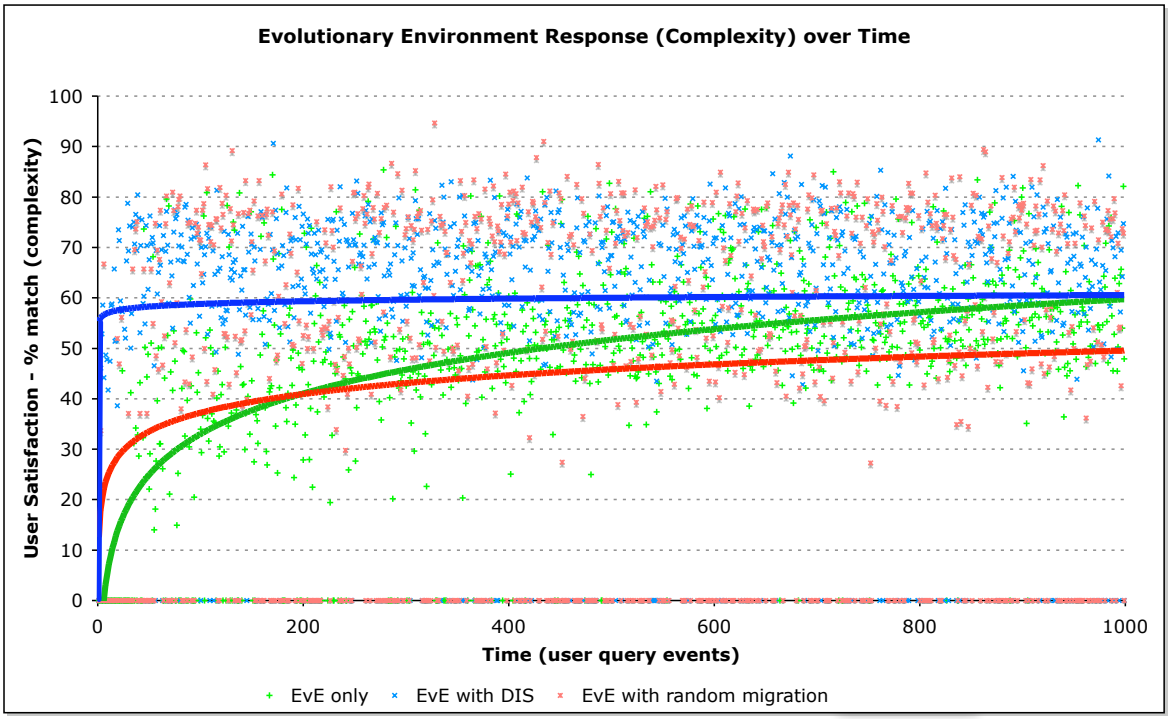


Figure A.3: Graph: EvE Response (Complexity) over Time

The DIS was then activated within the simulation and the same measurements were taken. The effect of the DIS was virtually immediate, so much so we were concerned that the optimisation was fuelled by more than just the desired intelligent behaviour. As the optimisation of the EvE is dependent on agent migration and the DIS does lead to more migration, although targeted-migration, it was possible the improvement was more from the migration than the intelligent targeting of the migration. So we created a scenario with random migration equivalent to the migration of the DIS. The additional random migration, albeit helpful in the early stages, ultimately decreased the efficiency of the EvE. We therefore felt confident in concluding that the improvement from the DIS is primarily from the intelligent targeting of migration, instead of the additional migration that is created.

A.4 Conclusion

The preliminary results are very promising. Sufficiently for the lead author of this document to recommend the *targeted-migration* approach for the *distributed intelligence* of the DIS architecture.

B Simulation - Source Code

The source code for the classes within our simulation, which code for the NN-based *Individual Intelligence*, is provided. This was requested as part of the internal review.

B.1 Network.java

The original Network class is from an article entitled 'Programming Neural Networks in Java'[12], from the Java Developer's Journal (JDJ). We modified it so that it also *implements* the *Cloneable* interface.

```
/**
 * Network
 *
 * Modified from Jeff Heaton's Network.java
 *
 * now implements the Cloneable interface
 *
 */

public class Network implements Cloneable
    {

    /**
    * The global error for the training.
    */
    protected double globalError;


    /**
    * The number of input neurons.
    */
    protected int inputCount;

    /**
    * The number of hidden neurons.
    */
    protected int hiddenCount;

    /**
    * The number of output neurons
    */
    protected int outputCount;

    /**
    * The total number of neurons in the network.
    */
    protected int neuronCount;

    /**
    * The number of weights in the network.
    */
    protected int weightCount;

    /**
    * The learning rate.
```

```
*/
protected double learnRate;

/**
* The outputs from the various levels.
*/
protected double fire[];

/**
* The weight matrix this, along with the thresholds can be
* thought of as the "memory" of the neural network.
*/
protected double matrix[];

/**
* The errors from the last calculation.
*/
protected double error[];

/**
* Accumulates matrix delta's for training.
*/
protected double accMatrixDelta[];

/**
* The thresholds, this value, along with the weight matrix
* can be thought of as the memory of the neural network.
*/
protected double thresholds[];

/**
* The changes that should be applied to the weight
* matrix.
*/
protected double matrixDelta[];

/**
* The accumulation of the threshold deltas.
*/
protected double accThresholdDelta[];

/**
* The threshold deltas.
*/
protected double thresholdDelta[];

/**
* The momentum for training.
*/
protected double momentum;

/**
* The changes in the errors.
*/
protected double errorDelta[];


/**
* Construct the neural network.
*
* @param inputCount The number of input neurons.
```

```
* @param hiddenCount The number of hidden neurons
* @param outputCount The number of output neurons
* @param learnRate The learning rate to be used when training.
* @param momentum The momentum to be used when training.
*/
public Network(int inputCount, int hiddenCount, int outputCount,
               double learnRate, double momentum)
    {
    this.learnRate = learnRate;
    this.momentum = momentum;
    this.inputCount = inputCount;
    this.hiddenCount = hiddenCount;
    this.outputCount = outputCount;
    neuronCount = inputCount + hiddenCount + outputCount;
    weightCount = (inputCount * hiddenCount) + (hiddenCount * outputCount);

    fire        = new double[neuronCount];
    matrix      = new double[weightCount];
    matrixDelta = new double[weightCount];
    thresholds  = new double[neuronCount];
    errorDelta  = new double[neuronCount];
    error       = new double[neuronCount];
    accThresholdDelta = new double[neuronCount];
    accMatrixDelta = new double[weightCount];
    thresholdDelta = new double[neuronCount];

    reset();
    }


/**
* Returns the root mean square error for a complet training set.
*
* @param len The length of a complete training set.
* @return The current error for the neural network.
*/
public double getError(int len)
    {
    double err = Math.sqrt(globalError / (len * outputCount));
    globalError = 0;  // clear the accumulator
    return err;
    }


/**
* The threshold method. You may wish to override this class to provide other
* threshold methods.
*
* @param sum The activation from the neuron.
* @return The activation applied to the threshold method.
*/
public double threshold(double sum)
    {return 1.0 / (1 + Math.exp(-1.0 * sum));}


/**
* Compute the output for a given input to the neural network.
*
* @param input The input provide to the neural network.
* @return The results from the output neurons.
*/
```

```
  public double []computeOutputs(double input[])
      {
      int i, j;
      final int hiddenIndex = inputCount;
      final int outIndex = inputCount + hiddenCount;
      for (i = 0; i < inputCount; i++)
          {fire[i] = input[i];
      }

      // first layer
      int inx = 0;
      for (i = hiddenIndex; i < outIndex; i++)
          {
          double sum = thresholds[i];
          for (j = 0; j < inputCount; j++)
              {sum += fire[j] * matrix[inx++];}
          fire[i] = threshold(sum);
          }

      // hidden layer
      double result[] = new double[outputCount];
      for (i = outIndex; i < neuronCount; i++)
          {
          double sum = thresholds[i];
          for (j = hiddenIndex; j < outIndex; j++)
              {sum += fire[j] * matrix[inx++];}

          fire[i] = threshold(sum);
          result[i-outIndex] = fire[i];
          }
      return result;
      }


/**
 * Calculate the error for the recogntion just done.
 *
 * @param ideal What the output neurons should have yielded.
 */
public void calcError(double ideal[]) {
  int i, j;
  final int hiddenIndex = inputCount;
  final int outputIndex = inputCount + hiddenCount;

  // clear hidden layer errors
  for (i = inputCount; i < neuronCount; i++) {
    error[i] = 0;
  }

  // layer errors and deltas for output layer
  for (i = outputIndex; i < neuronCount; i++) {
    error[i] = ideal[i - outputIndex] - fire[i];
    globalError += error[i] * error[i];
    errorDelta[i] = error[i] * fire[i] * (1 - fire[i]);
  }

  // hidden layer errors
  int winx = inputCount * hiddenCount;

  for (i = outputIndex; i < neuronCount; i++) {
    for (j = hiddenIndex; j < outputIndex; j++) {
```

```
        accMatrixDelta[winx] += errorDelta[i] * fire[j];
        error[j] += matrix[winx] * errorDelta[i];
        winx++;
      }
      accThresholdDelta[i] += errorDelta[i];
    }

    // hidden layer deltas
    for (i = hiddenIndex; i < outputIndex; i++) {
      errorDelta[i] = error[i] * fire[i] * (1 - fire[i]);
    }

    // input layer errors
    winx = 0;  // offset into weight array
    for (i = hiddenIndex; i < outputIndex; i++) {
      for (j = 0; j < hiddenIndex; j++) {
        accMatrixDelta[winx] += errorDelta[i] * fire[j];
        error[j] += matrix[winx] * errorDelta[i];
        winx++;
      }
      accThresholdDelta[i] += errorDelta[i];
    }
  }

  /**
   * Modify the weight matrix and thresholds based on the last call to
   * calcError.
   */
  public void learn() {
    int i;

    // process the matrix
    for (i = 0; i < matrix.length; i++) {
      matrixDelta[i] = (learnRate * accMatrixDelta[i])
                   + (momentum * matrixDelta[i]);
      matrix[i] += matrixDelta[i];
      accMatrixDelta[i] = 0;
    }

    // process the thresholds
    for (i = inputCount; i < neuronCount; i++) {
      thresholdDelta[i] = learnRate * accThresholdDelta[i]
                        + (momentum * thresholdDelta[i]);
      thresholds[i] += thresholdDelta[i];
      accThresholdDelta[i] = 0;
    }
  }

  /**
   * Reset the weight matrix and the thresholds.
   */
  public void reset() {
    int i;

    for (i = 0; i < neuronCount; i++) {
      thresholds[i] = 0.5 - (Math.random());
      thresholdDelta[i] = 0;
      accThresholdDelta[i] = 0;
    }
    for (i = 0; i < matrix.length; i++) {
      matrix[i] = 0.5 - (Math.random());
```

```
      matrixDelta[i] = 0;
      accMatrixDelta[i] = 0;
    }
  }

  /**
   * Object cloning.
   */
    protected Object clone()
        {
        try{
            Network n = (Network)super.clone();
            n.globalError=globalError;
            n.inputCount = inputCount;
            n.hiddenCount = hiddenCount;
            n.outputCount= outputCount;
            n.neuronCount = neuronCount;
            n.weightCount = weightCount;
            n.learnRate = learnRate;
            n.fire= fire;
            n.matrix=matrix;
            n.error = error;
            n.accMatrixDelta= accMatrixDelta;
            n.thresholds = thresholds;
            n.matrixDelta = matrixDelta;
            n.accThresholdDelta = accThresholdDelta;
            n.thresholdDelta = thresholdDelta;
            n.momentum = momentum;
            n.errorDelta = errorDelta;
            return n;
            }catch(Exception e){e.printStackTrace();}
        return this;
        }

    }
```

B.2 Intelligence.java

The Intelligence class is a modified version of the XorExample class from an article entitled 'Programming Neural Networks in Java'[12]. We modified it to remove the user interface, and provided methods for the pre-processing of the abstract numeric SBVR of our simulation. It no longer *extends* JFrame or *implements* the *ActionListener* interface, but instead *implements* the *Cloneable* interface.

```
/**
 * Intelligence
 *
 * Modified from Jeff Heaton's XorExample.java
 *
 *
 * no longer extends JFrame or implements the ActionListener interface
 *
 *
 * now implements the Cloneable interface
 *
 *
 * Existing methods have been and new methods have been created for
 * the pre-processing of the abstract numeric SBVR of the simulation.
 *
 */

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import java.util.*;


public class Intelligence implements Runnable, Cloneable
    {

    /**
    * The background worker thread.
    */
    protected Thread worker = null;

    /**
    * The number of input neurons.
    */
    protected int NUM_INPUT;

    /**
    * The number of output neurons.
    */
    protected int NUM_OUTPUT = 1;

    /**
    * The number of hidden neurons.
    */
    protected int NUM_HIDDEN;

    /**
    * The learning rate.
    */
    protected double RATE = 0.5;
```

```
/**
* The learning momentum.
*/
protected double MOMENTUM = 0.7;


/**
* The training data that the user enters.
* This represents the inputs and expected
* outputs for the XOR problem.
*/
protected double[][] data; // = new double[1][4];


/**
* The neural network.
*/
protected Network network;

int nameNumDigits;
int valueNumDigits;
boolean debug = false;
Random random = DBE.random;
Agent agent;
Description description;


/**
* Constructor. Setup the components.
*/
public Intelligence(Agent agent)
    {
    if(debug)System.out.println("Intelligence.Intelligence()");
    this.agent = agent;
    this.description = agent.getDescription();
    Hashtable attributes = description.getAttributes();
    nameNumDigits = Integer.toBinaryString(Description.numNames).length();
    valueNumDigits = Integer.toBinaryString(Description.numValues).length();
    int inputNeurons = attributes.size()*(nameNumDigits + valueNumDigits);

    NUM_INPUT = inputNeurons;
    NUM_HIDDEN = (inputNeurons * 3)/2 ;
    System.out.println("NUM_INPUT = " + NUM_INPUT);
    network=new Network(NUM_INPUT, NUM_HIDDEN, NUM_OUTPUT, RATE, MOMENTUM);
    }


/**
* Called when the user clicks the run button.
*/
protected boolean isSimilar(Agent a)
    {
    if(debug)System.out.println("Intelligence.isSimilar()");
    setData(a);
    double xorData[][] = getGrid();
    double ret = 0.0;

    for (int i=0;i<1;i++)
        {
        NumberFormat nf = NumberFormat.getInstance();
        double d[] = network.computeOutputs(xorData[i]);
```

```
            ret = d[0];
            }

        return (ret>=0.8);
        }


    /**
    * Called when the user clicks the train button.
    */
    protected void train()
        {
        if(debug)System.out.println("Intelligence.train()");
        if ( worker != null )
            worker = null;
        worker = new Thread(this);
        worker.setPriority(Thread.MIN_PRIORITY);
        worker.start();
        }


    /**
    * The thread worker, used for training
    */
    public void run()
        {
        if(debug)System.out.println("Intelligence.run()");
        setData(agent);
        double xorData[][] = getGrid();
        double xorIdeal[][] = getIdeal();

        int max = 10000;
        for (int i=0;i<max;i++)
            {
            for (int j=0;j<xorData.length;j++)
                {
                network.computeOutputs(xorData[j]);
                network.calcError(xorIdeal[j]);
                network.learn();
                }
            }
        }


    public void setData(Agent a)
        {
        if(debug)System.out.println("Intelligence.setData()");
        Hashtable attributes = a.getDescription().getAttributes();
        double[][] input = new double[1][NUM_INPUT];
        StringBuffer inputString = new StringBuffer();
        Object[] attributeNames = attributes.keySet().toArray();
        Arrays.sort(attributeNames);
        for(int i=0; i<attributeNames.length; i++)
            {
            String nameBinary = Integer.toBinaryString(
                                ((Integer)attributeNames[i]).intValue());
            String valueBinary = Integer.toBinaryString((
                (Integer)attributes.get((Object)attributeNames[i])).intValue());

            for(int j=0; j<(nameNumDigits-nameBinary.length()) ; j++)
                {inputString.append("0");}
```

```
            inputString.append(nameBinary);

            for(int j=0; j<(valueNumDigits-valueBinary.length()) ; j++)
                {inputString.append("0");}
            inputString.append(valueBinary);
            }

        int limit = -1;
        if(input[0].length > inputString.length())
            {limit = inputString.length();}
            else{limit = input[0].length;}

        for(int i=0; i<limit ; i++)
            {
            if(inputString.charAt(i)=='0')
                {input[0][i] = 0;}
                else{input[0][i] = 1;}
            }

        this.data = input;
        }


    /**
    * Called to generate an array of doubles based on
    * the training data that the user has entered.
    *
    * @return An array of doubles
    */
    double [][]getGrid()
        {
        if(debug)System.out.println("Intelligence.getgrid()");
        double array[][] = new double[1][NUM_INPUT];

        for ( int i=0;i<NUM_INPUT;i++ )
            {array[0][i] =data[0][i];}

        return array;
        }


    /**
    * Called to the the ideal values that that the neural network
    * should return for each of the grid training values.
    *
    * @return The ideal results.
    */
    double [][]getIdeal()
        {
        if(debug)System.out.println("Intelligence.getIdeal()");
        double array[][] = new double[1][1];
        array[0][0] = 1.0;
        return array;
        }


    protected Object clone(Agent a)
        {
        if(debug)System.out.println("Intelligence.clone()");
        try{
            Intelligence i = (Intelligence)super.clone();
```

```
            i.worker = null;
            i.NUM_INPUT=NUM_INPUT;
            i.NUM_OUTPUT = NUM_OUTPUT;
            i.NUM_HIDDEN = NUM_HIDDEN;
            i.RATE = RATE;
            i.MOMENTUM = MOMENTUM;
            i.data = data; // = new double[1][4];
            i.nameNumDigits = nameNumDigits;
            i.valueNumDigits = nameNumDigits;
            i.agent = a;
            i.description = a.getDescription();
            i.network = (Network)(network.clone());
            return i;
            }catch(Exception e)
                {e.printStackTrace();}
        return this;
        }


    }
```

C Real SBVR Pre-Processing for NN Matching

The issue of pre-processing the SBVR BML service descriptions for the NNs to operate upon, was called the SBVR matching algorithm. Although we were not initially concerned about the naming, we are now concerned that maybe it has caused some confusion. So, to clarify; the issue is not the matching algorithm, which is managed by the NN, but the pre-processing of the SBVR BML service descriptions for the NNs to operate effectively. The problem was to create a suitable encoding strategy for the SBVR BML service descriptions, which would be compatible with the NNs and scalable in the architecture. This document aims to address this issue as follows. Firstly, a very brief summary of the relevant SBVR BML syntax is provided. We then provide the algorithm for the pre-processing of the SBVR BML service descriptions, i.e. its conversion to binary. This is followed by minor updates to one of the EvE/DIS sequence diagrams, to clarify where and when the pre-processing occurs.

C.1 SBVR BML Syntax

I will comment briefly on the key elements from deliverable D6.3, which are required to understand the remainder of this document. The following is from D6.3 and is a few vocabulary entries from the BML metamodel:

subunit
Concept Type: role
General Concept: business entity

unit
Concept Type: role
General Concept: business entity

unit *owns* subunit
Concept Type: partitive fact type
Definition: business entity with the role of unit is a composition of smaller entities (subunits), this relationship meets the need to represent complex organizational structures.
Necessity: Each subunit *is owned by* exactly one unit
Synonymous Form: subunit *is owned by* unit

The key point to stress is that SBVR BML descriptions consist of descriptive elements defined by units and subunits, and also data elements when the model is populated.

C.2 Algorithm: convert SBVR service descriptions to binary

The algorithm firstly reduces the SBVR to its simplest form, then extracts the relevant terms, then applies an average term length, and then finally convert to ASCII binary. This can be defined in the following steps:

1. Reduce SBVR BML service description to simplest form.

2. Extract the business descriptive terms.
3. Create an order set of the extracted terms.
4. Convert the set of terms to binary.

At this stage, the pre-processing is complete and the output can be passed to the NN. The steps of the algorithm are specified below in pseudo-code with explanations and examples.

C.2.1 Reduce SBVR BML service description to simplest form.

This involves converting the more complicated terms to the simpler terms in the models definition. So, reducing the SBVR description to its simplest units and subunits. Show below is an SBVR vocabulary for a car rental service from D15.3:[17]

car rental
General Concept: business entity
Reference Scheme: the name *of* the car rental

car rental *has* name

car storage capacity
Definition: number *of* cars that *can be stored at* the car rental site

car rental *has* car storage capacity

The structure of SBVR BML can lead to syntactically different statements being semantically identical. The following statement is a valid SBVR service description, but not in its simplest form:

INPUT: The car rental 'Hertz', *has* cars storage capacity 200.

Reducing the statement above to one of its simplest forms, yields the statement below:

OUTPUT: The car rental *has* name 'Hertz', *has* number *of* cars 200.

The term car rental cannot be reduced to a General Concept, as this would be a reduction to the BML metamodel, and not the BML model of the car rental service. Also, the addition of extra terms such as the, that or has is unimportant as they will be removed in the following step.

C.2.2 Extract the business descriptive terms.

This is simply extracting the semantically important terms, while ignoring the rest. So extracting the units, subunits, and data elements.

INPUT: The car rental *has* name 'Hertz', *has* number *of* cars 200.

For the statement above, we need simply extract the terms which are underlined. This will give us the following list of terms shown below:

OUTPUT: car rental name Hertz number cars 200

C.2.3 Create an order set of the extracted terms.

The terms extracted now need to be placed in a set structure, and then ordered alphabetically. It is suggested to convert all uppercase letters to lowercase to simplify the ordering. So for our example:

INPUT: car rental name Hertz number cars 200

OUTPUT: [car rental, cars, **h**ertz, name, number, 200]

In the statement above, the first character of the term hertz has been emboldened. This is to show clearly its change in case, and it will remain emboldened in the following steps to show the characters path through the remainder of the algorithm.

C.2.4 Convert set of terms to binary.

This involves standardising the length of the terms and then encoding the terms into a binary format. The standardising of the term length, requires either padding or shortening the terms. Although it is conceptually easy to shorten or pad terms, the difficulty is in defining an optimal term length to be consistent throughout the EvE/DIS. Also, which is the best choice of the available binary encoding strategies, or should we create a custom one.

Average Term Length

There are not currently any available statistics on the average word length in SBVR BML, or more specifically the key terms we extract within SBVR BML service descriptions. Also, I have so far found limited information on the average word length in business English, which is for all words and again not just for important keywords. For English, the average word length is just under nine characters, and is only six characters for 'business English'. Most importantly, we are dealing with the average term length and not the average word length. Also, the more padding characters used the more inefficient the NN matching can potentially become, because it can lead to false positives when descriptions with many identical padding characters are compared. This could potentially be trained out, but it would be computationally more expensive. I am therefore going to make the following reasonable assumptions:

- average word length (AWL) of SBVR BML will be similar to the AWL in business English
- the majority of SBVR BML terms will consist of one word

- terms deviating from these assumptions will be rare

So, the average term length will be 6 characters and we will remove any space characters before padding or shortening the SBVR BML terms in the ordered set from the previous step.

Binary Encoding Strategies

The obvious choices are ASCII, Unicode, or the creation of a custom code. The last would seem unwise and unnecessary. There are surely other encoding strategies, maybe already in use within the DBE core architecture, but we shall focus on the widely used ASCII and Unicode. Unicode is based on ASCII, but has been extended to provide multilingual support, which also makes it more sophisticated (16-bit or 32-bit encoding per character). ASCII is simpler (8-bit encoding per character), but only supports English. The DBE and SBVR BML currently only support English, but promise to support other languages later. ASCII would be sufficient for the current objectives, and it would surely be more computationally efficient. Alternatively, Unicode would additionally meet the future long term objectives, but would be less computationally efficient (more significantly so in the short term than the long term). We believe the choice is ultimately an implementation choice, which must be aware of issues of compatibility, interoperability, and other issues of practicality. For our simulations, which obviously has minimal compatibility or interoperability issues, we have chosen ASCII. The padding character we chose is the null character, , which in binary is 00000000. It is the recommended character for the sake of simplicity and will also make the following example clear. However, it should make little difference which character is chosen for the padding character, provided it is unique. Continuing our example from the previous step:

INPUT: **[**car rental**,** cars**,** hertz**,** name**,** number**,** 200**]**

As this step is more complicated than the previous ones, we will show its intermediate steps. First, the removal of any space characters in terms with more than one word:

[carrental**,** cars**,** hertz**,** name**,** number**,** 200**]**

Then the standardisation of the term length. This requires the addition of the ASCII null padding character for terms shorter than six characters, and the shortening of terms that have more than six characters.

[carren**,** carsøø**,** hertzø**,** nameøø**,** number**,** 200øøø**]**

The next step is to remove any formatting characters.

carren carsøø hertzø nameøø number 200øøø

The next step is to convert each character to binary following the ASCII format. The ASCII hexadecimal is initially shown, as it is easier to read and commonly used.

63617272656E 636172730000 6865727 47A00 6E616D650000 6E756D626572 323030000000

The line breaks shown in the output below are only for fitting the example on the page, and otherwise would not be there.

OUTPUT: 01100011 01100001 01110010 01110010 01100101 01101110 01100011 01100001 01110010 01110011 00000000 00000000 **01101000** 01100101 01110010 01110100 01111010 00111111 01101110 01100001 01101101 01100101 00000000 00000000 01101110 01110101 01101101 01100010 01100101 01110010 00110010 00110000 00110000 00111111 00111111

The actual output would not have the spaces between the encoded characters, which are present to show the individual character encodings.

The potential loss of information when shortening terms will not overly affect the DIS, because it does not require perfect matching to operate effectively.

C.3 Sequence Diagram

The Deploy Service sequence diagram has been updated and now also includes the method SBVRpreprocessing(). The pre-processing of an SBVR BML service description is necessary when a DBE service is deployed, specifically when the EveService is created in its home Habitat. We suggest that the binary version of an SBVR BML service description is stored within the EveService at deployment. This will avoid repeated pre-processing of the service description when the EveService interacts with other EveServices during the periods in which the DIS is activated. The relevant version part of the Deploy Service sequence diagram is shown on the next page, and the full version is in a separate document at www.iis.ee.ic.ac.uk/ g.briscoe/D6.6/appendix/.

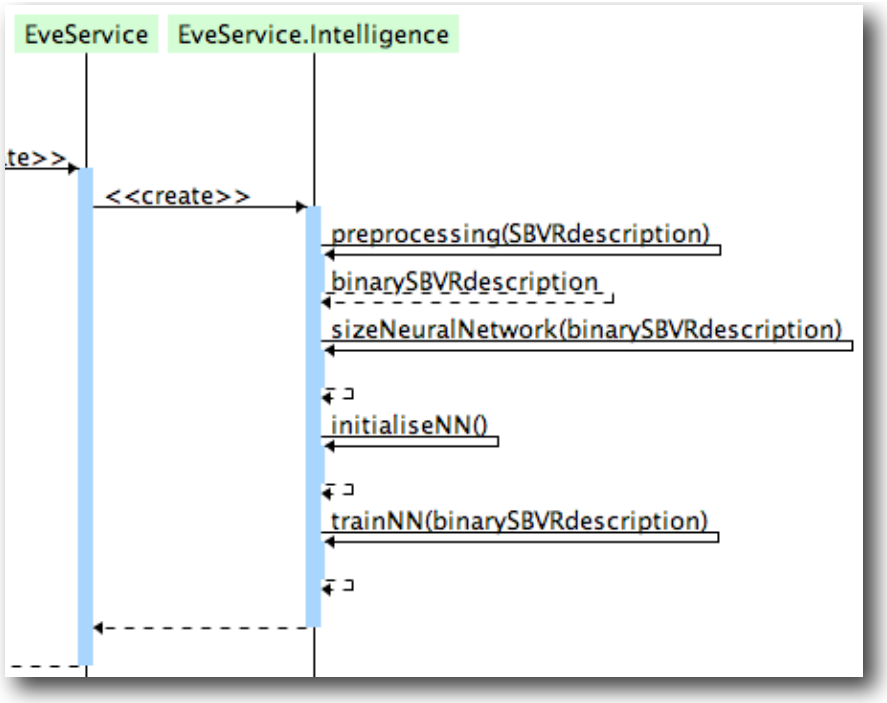
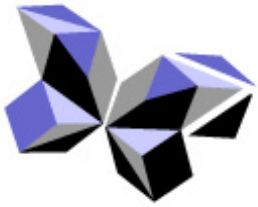


Figure C.1: Section of Updated Deploy Service Sequence Diagram

D Evolutionary Environment Discussion Paper

The following pages are the original Evolutionary Environment Discussion Paper [6] produced in June of 2004, which represented the first synthesis of our ideas in the creation of the Evolutionary Environment. It is included for completeness, as it is an internal document that has not previously been included in any deliverables, unlike the Evolutionary Environment Architecture Requirements [2] document which was presented in the appendix of deliverable D6.2 'Control of Self-organisation and a Performance Measure' [4].



Digital Business Ecosystem

Contract n° 507953

Evolutionary Environment Discussion Paper

Project funded by the European Community under the “Information Society Technology” Programme

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°:

Due date:

Delivery Date:

Short Description: The aim of this document is to discuss work from the science stream and computing stream in respect of the DBE architecture.

Partners owning: ICL, UBham, LSE

Partners contributed: STU, Soluta, UniS, TCD, Sun

Made available to: Consortium and EC

Versioning		
Version	**Date**	**Author, Organisation**
1	01/05/04	Gerard Briscoe, ICL
2	13/06/04	Paolo Dini, LSE
3	24/06/04	Gerard Briscoe, ICL
4	24/06/04	Paolo Dini, LSE

Quality check

1st Internal Reviewer: Pierfranco Ferronato, Soluta.net

2nd Internal Reviewer:

Vision

This paper is a discussion document on what it might mean for (relatively) high-level software components to evolve autonomously. Current evolutionary algorithms for evolving programs (so-called Genetic Programming) suffer from a number of weaknesses. Two critical ones are:

- o While being moderately successful at evolving simple programs, it seems very difficult to get them to scale up and evolve programs with any kind of complexity.

- o The estimated fitness of a program is normally given by a measure of how accurately it computes a given function, as represented by a set of input/output pairs. There is no guarantee therefore that the evolved program actually does the intended computation.

These issues are particularly important if we want to evolve high-level, complex structured software.

An abstract model

Let's first start with an abstract model of a system of evolving software. We imagine that there is some underlying set of components. Each component has two properties: a piece of executable code (perhaps stored on some remote machine via a proxy) and a guarantee that specifies something about what the code does.

Examples:

- o each component is a line of code in some programming language
- o each component executes some function within a spreadsheet
- o each component provides some kind of web service
- o each component provides the front-end to a “real-world” service

Now the problem is that we want to find a collection of components that satisfies some given specification. It may be possible to do this using some optimisation algorithm. But it may also arise as the product of an evolutionary process. The DNA in this case would correspond to a structure which specifies which components are to be used and the relationship between them. Mutations may occur by switching some components into and out of the structure, or by changing the relationships. Crossover (recombination) may occur by combining elements of two or more structures into a new structure.

In this way, a piece of DNA has an interpretation (or phenotype) as a large complex-structured piece of software which can execute some task. However, there is more. Because each component also carries a guarantee, or specification, of what it does, it ought to be possible to use some software validation technique to verify whether or not the overall collection satisfies the required specification. Indeed, this can be the heart of the fitness function which drives the evolutionary process.

Structure

Collections of software components may be composed together in different structures. If they are providing real-world services, then maybe all we require is a set of such services. If they are lines of code from a programming language, then the components will be ordered as a sequence. A collection of web-services may have a tree structure. And functions in a spreadsheet may be related by a directed network

of inputs and outputs.

The kind of structure that is being used makes a big difference to a number of aspects of evolution. A set may be represented easily by a binary string, in which the bits indicate whether or not a component is in the set. Mutation is done bitwise and crossover by masks. The result will look like a fairly standard genetic algorithm. Sequences of code and services may be represented by variable length strings over some alphabet. The dynamics of evolutionary processes over such strings has been studied a little in the last few years. There are a number of ways of defining mutation and crossover, each introducing their own biases into the kinds of program (e.g. in terms of size or "scale") that will be sampled most often. Standard genetic programming usually works on tree structures, although there has been reent work done on networks (e.g. "Cartesian Genetic Programming"). Theory here is more sparse but some progress has been made. Of particular interest is the phenomenon in which trees tend to grow bigger and bigger as the evolution goes on, without necessarily any corresponding increase

Fitness

We have already suggested that the prime driver of the evolutionary process should be the extent to which a candidate collection can verifiably satisfy the specified requirements. It may be that this can be measured probabilistically, or perhaps some kind of theorem-proving can be used to validate the system. However, there will also be other pressures on fitness. For example, one may seek the most parsimonious solution to a problem (one that provides exactly the specified features and no more), or the cheapest solution, or one with a good "reputation". Some aspects of fitness will be implicit in the evolutionary process (e.g. collections which are often used may gain more fitness) while others will require explicit measures (e.g. price, or user satisfaction).

One way to handle this multiplicity of fitness values (some qualitative!) is to explicitly recognise the multi-objective nature of the optimisation problem. In this way, we are seeking not the single best solution, but a range of possible trade-offs that can be made most optimally. The set of solutions for which there exist no better trade-offs, is called the Pareto-set. Evolutionary techniques have been adapted to solve such problems with considerable success. The main point is that selection has to be driven not by an absolute value of fitness, but rather by a notion of what it means for one solution to be better that another. We say one solution dominates another if it is better in at least one respect, and no worse in any of the others.

Another way in which multiple objectives will arise is from the fact that there will be, at any time, many different specifications needing to be satisfied. It would be a mistake to try to find one solution which attempts to meet all the requirements. Instead, each specification corresponds to a different niche in the evolving ecosystem.

Modules and hierarchical dynamics

In order to construct ever more complex software solutions it seems clear that some form of modularity is needed. This has been attempted by a number of researchers with mixed results. The basic idea would be to make successful solutions at one time step become individual components in the future. That is, it becomes possible to plug-in an entire package into an evolving collection. All components must provide a guarantee as to what it is they compute (or what service they provide). For a module (composite system) this would derive from the specification it was previously evolved

to satisfy. Therefore it will be able to take part in further software validation.

In this way, the system can become open-ended (or “evolvable”). As the sys-tem receives more and more sophisticated requests, so more and more complex software collections become available in the population to be used as modular components.

Evolution of specifications

One final speculation:
It has been suggested that the executable code (or service or whatever) may exist at some remote site. The evolutionary process only requires the existence of the guarantees to work. The actual underlying code or service only comes into play once the whole package has been assembled. There is therefore no strict requirement for the underlying code to actually exist until this stage. That is, evolution could take place entirely at the level of the specifications. Software component providers would only need to actually supply the code once there is a demand for it. That is, when one of their specifications (or guarantees) has been used in the construction of a software solution which meets a user's requirements. The whole system then becomes more like an evolving futures market.

Working Assumptions and Glossary

Important assumptions and terms are explained below.

(1) Requirements for evolution:
population, replication, variation/mutation, selection pressure

(2) Fitness = measure of how successful an organism is in its environment

In terms of web services, this means that fitness only really exists at runtime! Only phenotypes have fitness.

Artificial pre-runtime fitness in service-ecosystem:

fitness = technical & relative & testing & runtime
where: technical = SDL comparison
relative = BML comparison
testing = UniS Testing Harness
runtime = STU Runtime Feedback

Technical in the sense that a service chain can actually run. Relative as in whether it is any use to the SME that made the request.

Ideally the UniS 'testing harness' would form the core of the fitness function. Although it will probably provide measures relative to the request, and for the evolutionary process the values will have to be normalised. Doubtful that it will be a trivial normalisation.

We are suggesting that there may be more than one factor involved in fitness. We expect the business people will come up with some more factors. So what are we going to do to integrate these things? One could take a weighted sum - but picking weights is rather difficult. An alternative approach is to use the idea

of Pareto dominance. We say one thing dominates another if it improves on it with respect to at least one factor, while not doing worse in any of the other factors. Using this notion, we give up on the idea of anything having an absolute fitness value. But at least we can tell when one thing is to be preferred to another.

(3) Push and Pull
Pull is the process of generating service chains solutions upon request. Whereas Push is the process of using SME BML profiles to generate service chains expected to be needed by the SME.

(4) Best-Guess Solution
Term used to refer to the service chain solution provided by the recommender/composer to a user BML request.. Expected to be sub-optimal, hence the need for evolutionary algorithms.

(5) Hierarchy in biology.
cells -> tissues -> organs -> systems (e.g. lymphatic) -> organism -> populations -> community -> ecosystem -> biosphere
[where '->' = 'group to']

(6) Evolving Service Chains, NOT Services
The DBE will not evolve individual services, only the aggregation/recombination of service chains. Services are the base unit for evolution and will be created by SME software providers, and updated by them according to feedback from the DBE.

(7) Epidemiology
We will also try to think about the ecology of infectious diseases (epidemiology). There is some very good research on this subject. Think of software as a disease your computer gets - and passes on!

So far the SM-chain migration patterns and behaviour would seem to fit the behaviour of self-replicating viruses already. Are there significant features missing ?

Working Questions

Anybody, please add...

(1) Is BML decomposable ?

If you have a BML statement X for a complex service, can it be broken down to smaller tasks (atomic services)? I.e. where we can say that X can be considered as atomic services A-B-C-D.

The answer would appear to be yes, although I only understand this at an abstract level currently. If anyone has a more in-depth understanding, pls add....

(2) Is there any way in which we will be able to formally verify that a structure produced by an evolutionary process actually has all the desired properties? In particular, if we have evolved a (possibly complex) piece of software out of its components, can we be sure that it actually satisfies the requirements that it was evolved for? Can this verification process be part of the fitness function?

Yes, I believe this function can be provided by the UniS testing harness, and should therefore be incorporated into the fitness function.

(3) What kind of natural selection process is most appropriate? How are births/ deaths regulated? The number of copies of an individual is supposed to equal the measure of its fitness - but what if we are using Pareto dominance (no absolute fitness value)?

(4) Software services ecosystem - Evolution. Recombination has not really been considered much by DBE people so far. Trying to design a good method for recombining complex structures is a challenge - it's easy to end up with stuff that isn't even syntactically correct - let alone useful. Depending on the structure of the service chains (sets, sequences, trees, networks) there are different options and problems.

(5) Habitats - when should things die? As you say, it is still controversial in the DBE when things should be removed from the system. One possibility is to consider the biological notion of the recessive gene. This can specify something which is perhaps not all that useful - but that you still might want to keep "hidden" in the population, in case the situation changes and it becomes fashionable/useful again.

1 Introduction

The following is a proposed computing architecture to support the core ideas from the science stream in the DBE regarding evolution and ecosystems. The architecture will provide a more practical view on the science objectives in computing terms. Since early in the proposal writing stage, the statement 'services will migrate' has existed. It is we believe a visionary statement, but has troubled us as to how it could be done? Hopefully the following will provide an answer to this question and others.

2 Evolution of Services

To achieve evolution within the DBE, specifically the automatic service-chain recombination/optimisation (based on automatic service composition), is a very significant challenge, due to the range of services that must be catered for and the potentially huge number of factors that must be considered for creating an applicable fitness function. There is however a huge body of work and continuing research regarding the theoretical approach of evolutionary computing, and extensive use of such techniques as genetic algorithms for practical real world problem solving. It is therefore possible to extrapolate the key components required to enable evolution, which are of course dictated by the core requirements of evolution in general: population, replication, mutation, selection pressure.

Evolution can only be applied to a population of organisms/services, which of course over generations affects the individuals of the population. Replication and mutation are required so that change can be introduced in a population. The 'selection pressure' provides direction to change, and implies the death of individuals. The 'pressure' selects for those that are 'fit' and capable of surviving the environment to reproduce, and against those that do not have sufficient 'fitness' and die before passing on their 'genes'. Fitness is a measure of an organism's success in an environment. 'Genes' are the functional unit in biological evolution; in the DBE evolution the functional unit is the service. The 'genes' of an organism are strung together in a linear manner; the genes describe an organism's potential, its genotype. They therefore describe and organism's structure and behaviour in the environment, when it is instantiated (phenotype) at runtime.

There may be several different ways of combining services. The simplest is to combine them in an unstructured set. I suppose the "holiday booking" scenario is of this kind: one would like a flight, hotel, car-hire in the set of services, but there is no required order. Suppose that the population contains a set of services. Each service has its manifest which, amongst other things, specifies a set of things it promises to provide. A customer comes along with a set of requirements. It is unlikely that a single services will meet all of these requirements, so we are looking for a subset of the the services which do meet them all. We would also like to minimise the number of services in the subset, in the interests of finding the most economical solution. This problem is known as the "subset covering problem", and is NP-complete. That is, there are no known efficient algorithms for solving it. It is possible that an evolutionary approach would work well, however (though with no guarantee of optimal performance). One could represent a potential solution as a bitstring. Setting bit k to 1 means that service k is in the solution. The fitness would be a measure of how well the corresponding subset meets the requirements. Standard genetic algorithm operators (mutation and crossover) could be used.

A more complex situation is when the services are in a sequence, or chain. We now have variable length strings representing different solutions. Each character in the

string represents a different service. There are two variations, depending on whether duplicated services are allowed or not. If they are not allowed then the set of possible solutions is O(n!), where n is the number of services. This is much larger than 2^n, which is the case for the subset problem. One difficult problem here is how to sensibly define mutation and crossover operators in such a way as to preserve the constraint of no repetitions.

If repetitions are allowed, then the search space is countably infinite. This brings in problems of its own. The chief problem here is that of "bloat", a phenomenon commonly observed in the evolution of variable length structures. Bloat occurs when the size of the solutions in the population grow larger and larger, with no corresponding increase in fitness. There are several theories as to why this occurs, and some measures that can be taken to prevent it. The same applies to if other variable size structures such as trees are allowed.

Evolution is a very specialised task, which needs to be localised to specific habitats and populations, creating niches.

3 Fitness

The fitness function is based on the following criteria:

- SDL comparison within a SM-chain
- BML comparison of a SM-chain to a BML request/specification
- Information provided from the Testing Harness
- Runtime Fitness feedback

The services provided by a component in the DBE are described by the BML part of its "service manifest". Similarly, the services required by a user are also specified in BML. Now, depending on the syntactic complexity of BML, checking whether two such statements match can be highly non-trivial. Let's take an artificial example. Suppose a service exists that produces (positive) even numbers. Its manifest specifies (in BML) that its output is a positive integer and that the integer equals zero mod 2. Now a user comes along and specifies that they want a supply of positive integers with the following special property: the supplied positive integers must equal the sum of two primes. The task of checking whether or not the service satisfies this requirement is equivalent to proving Goldbach's conjecture - one of the most famous outstanding problems in mathematics. Clearly, there will be no efficient automatic way to do this. This might seem an extreme example, but if the BML allows logical expressions then the task of matching specifications will become equivalent to automatic theorem proving - which is notoriously slow.

A situation that often arises in evolutionary computation is that there may be more than one way to meet the fitness criteria (that is, there are multiple global optima). There is a theorem in theoretical biology that says that, in this case, exactly one solution will survive - the alternative solutions will die out. If we want to present all possible solutions to the user, then we have to prevent this happening. We do that by creating a separate "niche" around each optimal solution. This can be done in a number of ways, which are usually based on the idea of "fitness sharing". That is, if more than one solution in the population is at a particular optimum, they have to share the fitness associated with that point. That makes it less likely that they will all stay there - many of the individuals will move off to other optimum. Using fitness sharing like this, the population will spread out and occupy all the global optima.

A different technique is to use Pareto optimisation, which is referred to in the vision statement. The idea is to treat the problem as a multi-objective one - the objectives are the different requirements of the user. One solution is only considered to be better than another if it can meet a superset of the requirements that the other can meet. For example, if solution A can meet requirements 1,2, 5, 8 whereas solution B can meet 2, 5, 8, then we consider A to be best. If C can meet 1, 2, 8, 9 then it is considered neither better nor worse than A or B.

Regarding gathering of information for runtime fitness. I don't fully understand it yet, so not sure how to integrate it. Although, obviously, information gathered will be passed to the fitness object, which is directing the evolution in the population object from where the service was generated. The fitness function therefore needs to be structured to handle such information and weighted appropriately.

4 Ecosystems

An ecosystem is a complex entity and has great diversity. It contains many communities, which in turn contain many populations in varying habitats. Organisms migrate through the ecosystem into different habitats competing with other organisms for limited resources. Evolution occurs to all living components of an ecosystem (excludes for example rocks, generally). The 'selection pressure' varies from one population to the next depending on the environment that is the population's habitat. An ecosystem develops from a simpler state to a more complex (although more energy-efficient) climax community, by a process of 'succession', where the genetic variation of the populations changes with time, until the climax community is reached. After this point genetic change (evolution) is no longer dependent on time, i.e. it stops.

The ecosystem analogy asks many interesting questions, regarding the applicability of ecosystem processes to the DBE. This includes the DBE equivalent of Food Webs and Energy Pyramids.

5 Evolutionary Environement Logical Overview

5.1 Habitats

Each SME has a single 'habitat' object. It provides a pool of services specially selected for the SME, by favouring services which follow the principles of the SME's BML profile, and holds 'population' objects for evolving populations of 'service-chains' to specific user requests. It also provides a place for service migration to occur, as the 'habitat' objects will be interconnected with one another. These interconnected 'habitats' create the strictest interpreation of the term 'Digital Ecosystem' . See Fig 1. The habitat object is therefore the key component for supporting the ecosystem model within the DBE. The connections will be initialised randomly or based on SME BML profiles. This will be based on the 'fitness histories' of the migrating services and will allow habitat clustering to communities, which will be discussed later in this document.

When using Pareto-sets for fitness determination, the solution of multi-modal problems with sharing, and multi-objective problems with Pareto optimsation, often works best when the population is spatially distributed. This is because partial solutions (solutions to different niches) can evolve in different parts of the population. In a mixed population, individuals are always interacting with each other, which does

not allow time for this kind of specialisation.

One common way to distribute the population is the "island" model, which seems to map well onto the "habitat" idea. In the island model, there are a number of small populations which are all undergoing evolution. Every so often, however, there is a small amount of migration of highly fit individuals from one population to its neighbours. Empirically, this seems to work well—though there is little theory. The situation does seem similar, though, to the "meta-population" models of theoretical ecology. A similar model has been used for determining investment portfolio strategies .These issues will be looked into more detail in the next few months.

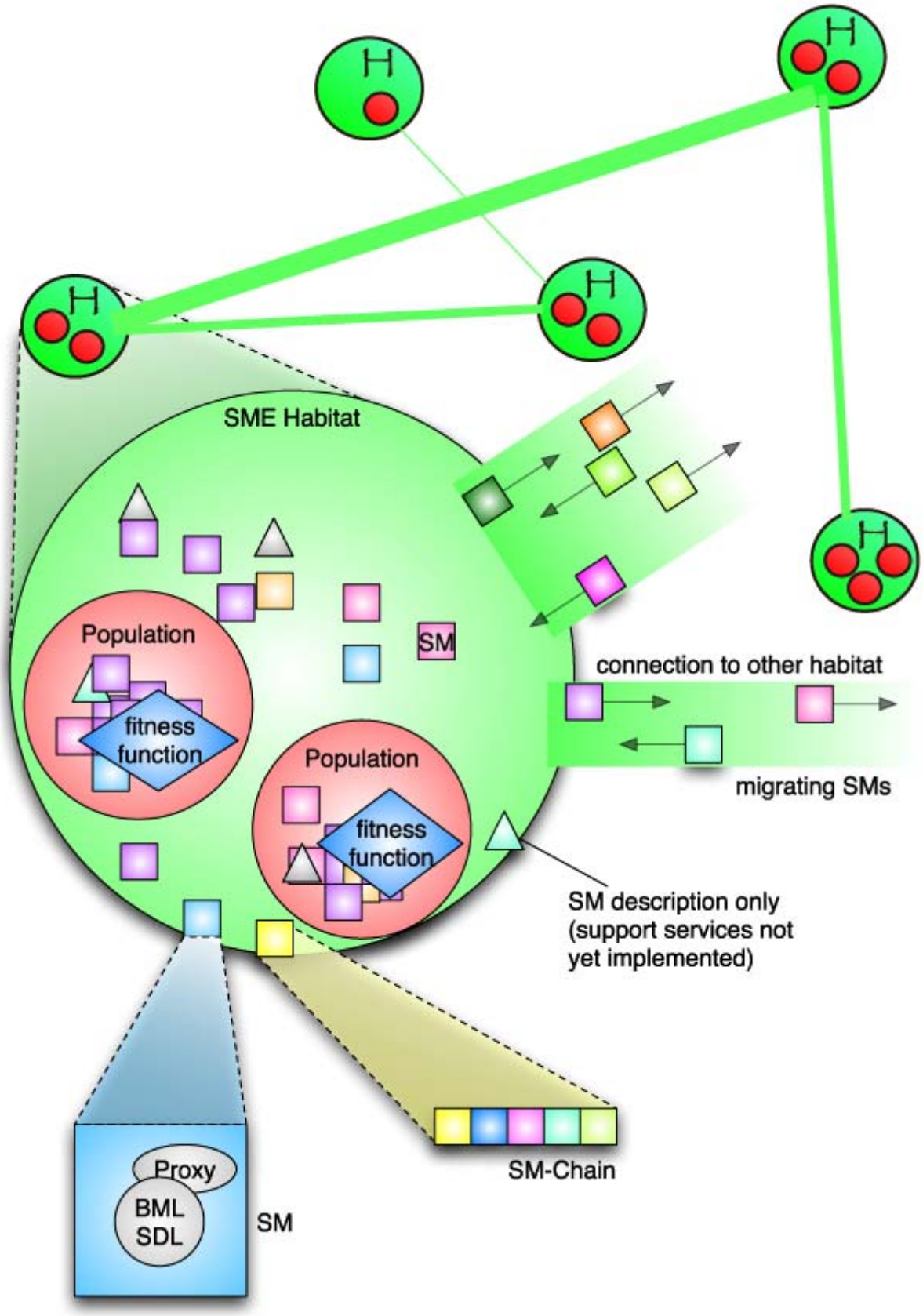


Fig 1. Logical Overview of Evolutionary Environment

The distributed populations of the island model form a network. The topology of this network can affect the evolutionary behaviour of the system. For example, highly regular networks may have different behaviour from randomly connected networks. So-called "small world" networks share some features of both. We may not be able to control the topology of the DBE network, but it is worth bearing in mind that it can have a significant effect. We will communicate with Sun and ICL on the dynamic network topology issues.

One of the key clashes between the 'biology' of the DBE and the computing has been the issue of the death of services. Specifically, in biology organisms die, and if 'services' are organisms then do they die? However, with the realisation that 'services' are more akin to genes, the question then becomes the loss of genes. How genes are lost can be stated in many ways, but the most useful point of view is the following; genes are lost when they have no niche in the environment. The 'home habitat' of a service is where it can find a niche, and therefore a location from which it can never die. It poses the question of whether services can and should be entered into the DBE via their 'home habitats', and spread by service migration through the interconnected 'habitats', joining 'service-chains' wherever they find additional niches.

With the interconnected habitats, the complexity is sufficient to warrant the name ecosystem, which in its developed state (climax community) is an incredibly complex structure. This opens the possibility of applying ecosystems theory to the ‘Evolutionary Environment’.

5.2 Populations

The 'population' object is instantiated with the user request (coded in BML) with which a 'Fitness Function' object is instantiated to provide the necessary fitness function. The fitness function provides a 'selection pressure' for evolving the population of 'service-chains' present.

Even if the problem of matching BML statements can be resolved (or side-stepped), one is still left with the problem of trying to evolve solutions which match a specification. That is, the great majority of the solutions generated will not match the specification entirely. It seems clear, then, that the degree of a match should somehow be incorporated into the fitness function. However, there is another problem with trying to combine services together - some services might not be compatible. For example, combining a flight to New York with car hire from Los Angeles and a hotel in Prague. One could simply disallow such combinations (that is, throw them away as soon as they are suggested). But that could be a mistake. Consider a service chain A-B-C. Suppose we make a point mutation to it to get A-D-C, but that this solution is not viable. There may, however, be another viable solution A-D-E which is better than the first. But if only single point mutations are allowed, this solution will never be reached. Therefore, it might be worthwhile tolerating non-viable solutions in the hope that we will be able to traverse them to new viable ones. One could do this by attaching some penalty function to the fitness of these individuals.

A third alternative is to implement some kind of repair algorithm (analogous to DNA repairing). Every chain that is produced is submitted to this algorithm. If it is viable, nothing happens. Otherwise it is repaired (typically in a "greedy" style) until it is viable. Whether or not this is a good idea depends on how isolated the viable solutions are - if they are too isoalted from each other, it may be better to use a penalty function and try to build a bridge between them.

The population object is the key component for supporting evolution within the DBE. The 'population' object is an artificial environment, having everything for evolution - population, replication, mutation and a selection pressure. Although, as many generations must pass in the evolutionary process to generate the optimal solution, the possibility for bootstrapping the process using the 'Best Guess Solution' from the 'Composer' must be considered.

The 'population' object also provides the opportunity for 'Pushing' solutions to the SMEs, by instantiating the object with 'BML service requests' generated from the SME's BML profile and past experience information form the Knowledge Base.

5.3 Communities - Habitat Clustering - Service Migration

There will be hundreds of habitats at least, and potentially three or more times the number of populations at any one time. There will then be thousands of services, each with several copies within the interconnected habitats. Each habitat is localised to the SME it was created for, and therefore the services within it have properties which match the properties of the SME, as specified in the SME BML profile.

Successful service chains will migrate through the DBE and will be available for reuse. If a new ‘push’ Population object is instantiated from the SME BML profile and is similar (will not be identical) to a request in another habitat where a successful SM chain was created, then the migration rate and habitat clustering will ensure the successful service chain will be present to take part in the new Population object, bootstrapping the evolutionary search process.

This infrastructure allows for the spontaneous formation of communities via habitat clustering. Many successful service migrations will reinforce habitat connections increasing the rate of service migration. If successful migration occurs due to a multi-hop migration, then a new link can be formed between those habitats. Unsuccessful migrations will result in connections (rate of migration) decreasing, until finally the connection is closed.

The clustering/'community formation' is like joining a club, you can be member of more than one. In the same way habitats can be clustered into more than one community. The clustering will lead to communities clustered over language, opportunity spaces, nationality, geography, etc. as shown in Fig. 2.

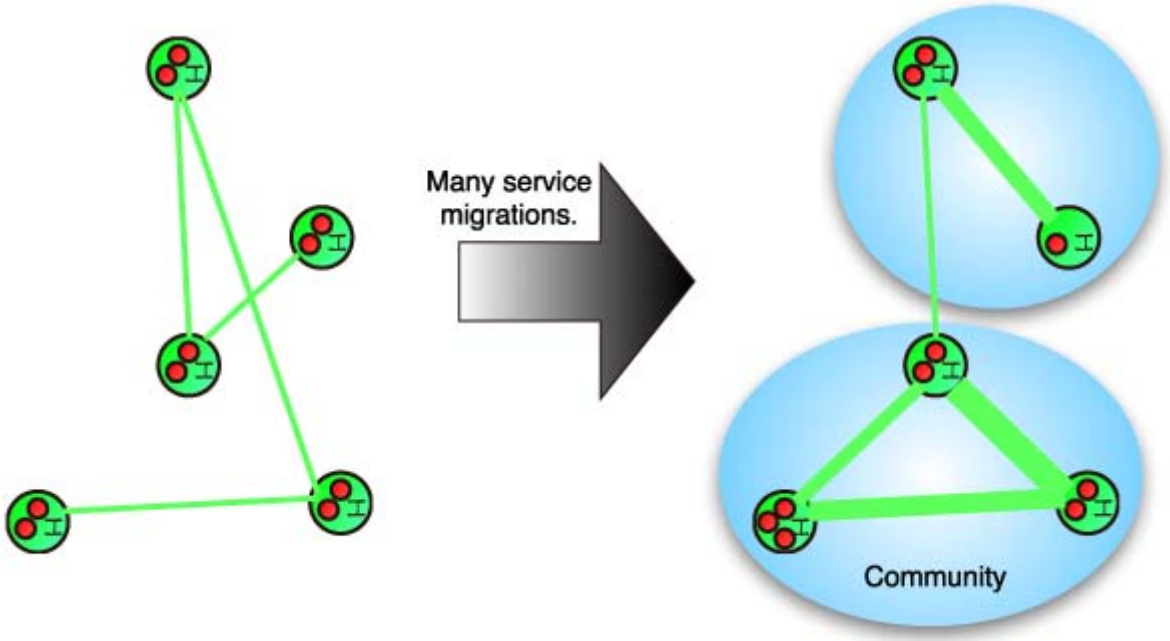


Fig 2. Community Formation - Habitat Clustering by Service Migration

1.6 Science Key Components Class Diagram

The class diagram in Fig. 3 shows the key components and their relations to one another for clarification of the descriptions given earlier.

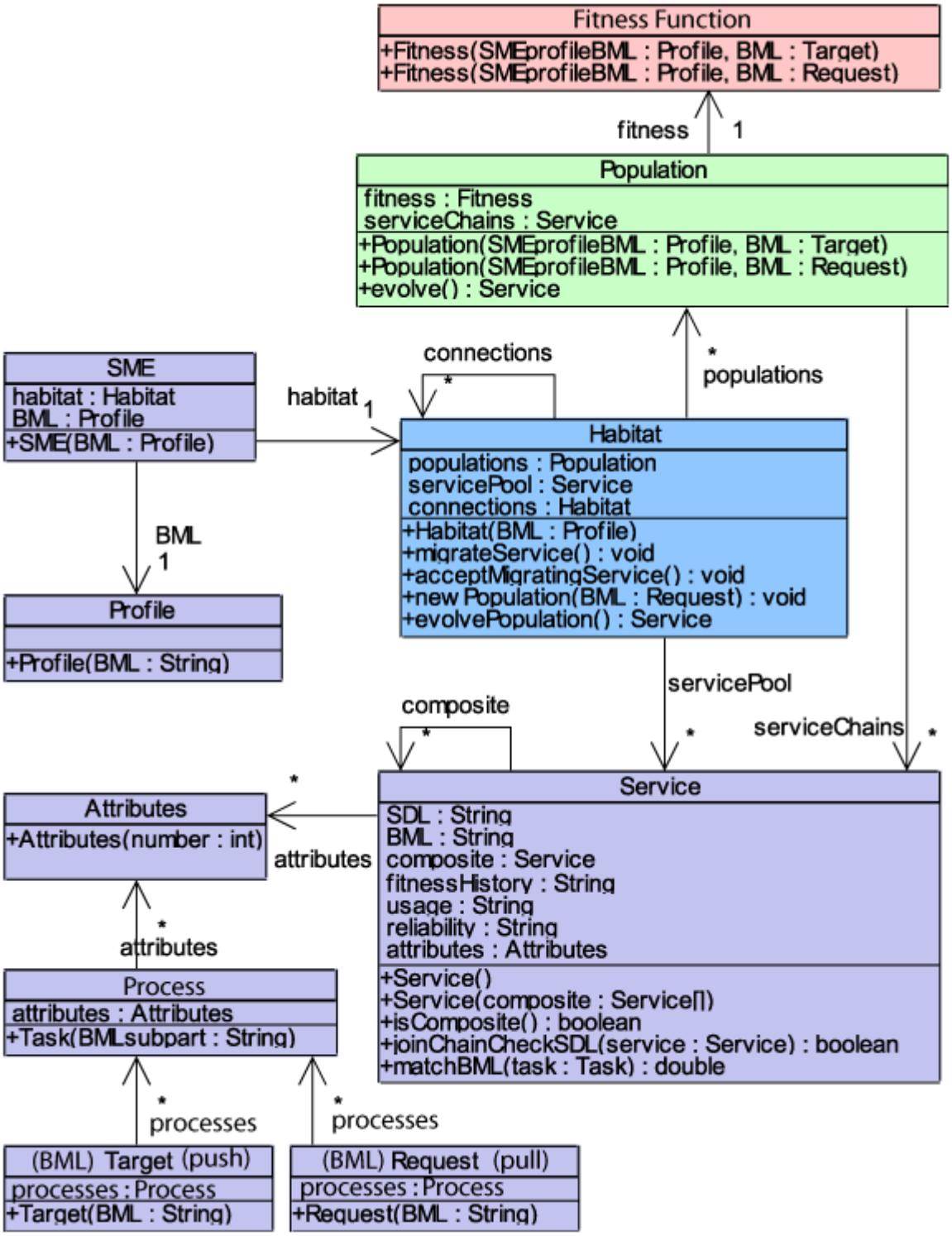


Fig 3. Class Diagram of Evolutionary Environement

1.7 Integration with DBE Core Architecture

This is the currently accepted view on the integration of the Evolutionary Environment with the DBE Core Architecture. At a black box level, the habitat/population components replace functionally the recommender/composer when handling user requests, but they operate in radically different ways. The habitat/population components provide additional functionality, such as service migration. The habitat/population components are a more self-contained system, as the habitats are interconnected and share information (SM/chains) and provide a more scalable system, still capable of high specialisation for each SME.

The Population component is contained within the Habitat object, and will be instantiated as needed to user requests and perceived user needs(generated from SME profiling and the SME BML profile). When a population object is instantiated, a 'fitness function' (selection pressure) object for evolving the population of SM chains is instantiated within the population object.

The Habitat component is created when an SME joins and from that point on runs continuously. Habitat objects can be kept running anywhere, just as in grid computing where the user is blind to where the processing occurs. A distributing processing arrangement, where supernodes handle several Habitat objects for SMEs that lack sufficient processing power, should be possible.

When a user request is received, it should be sent to both the recommender/composer and the SME Habitat object, where a population object will be instantiated to handle the request, and evolve the optimal solution. Following the technical annex, when a solution (Best Guess Solution) to a user request is generated by the recommender/composer, it should be fed into the already existing population object, where it can be used to bootstrap the evolutionary process. The population will in effect use the BGS to evolve a more optimal solution using evolutionary computing.

At startup, the number of services in habitats will be low. This creates the problem that the habitats will not be able to support the SMEs they represent. A possible solution to this is to have a single habitat represent a single community, and as the service pool grows, separate/duplicate the habitats to the different SMEs.

Fig. 4 shows the current view of the Evolutionary Architecture and how it is integrated in the DBE core architecture. Fig. 5 shows the roles of the various partners contributing to the DBE Evolutionary Environment, as we understand them today.

Feedback on both figures is welcome

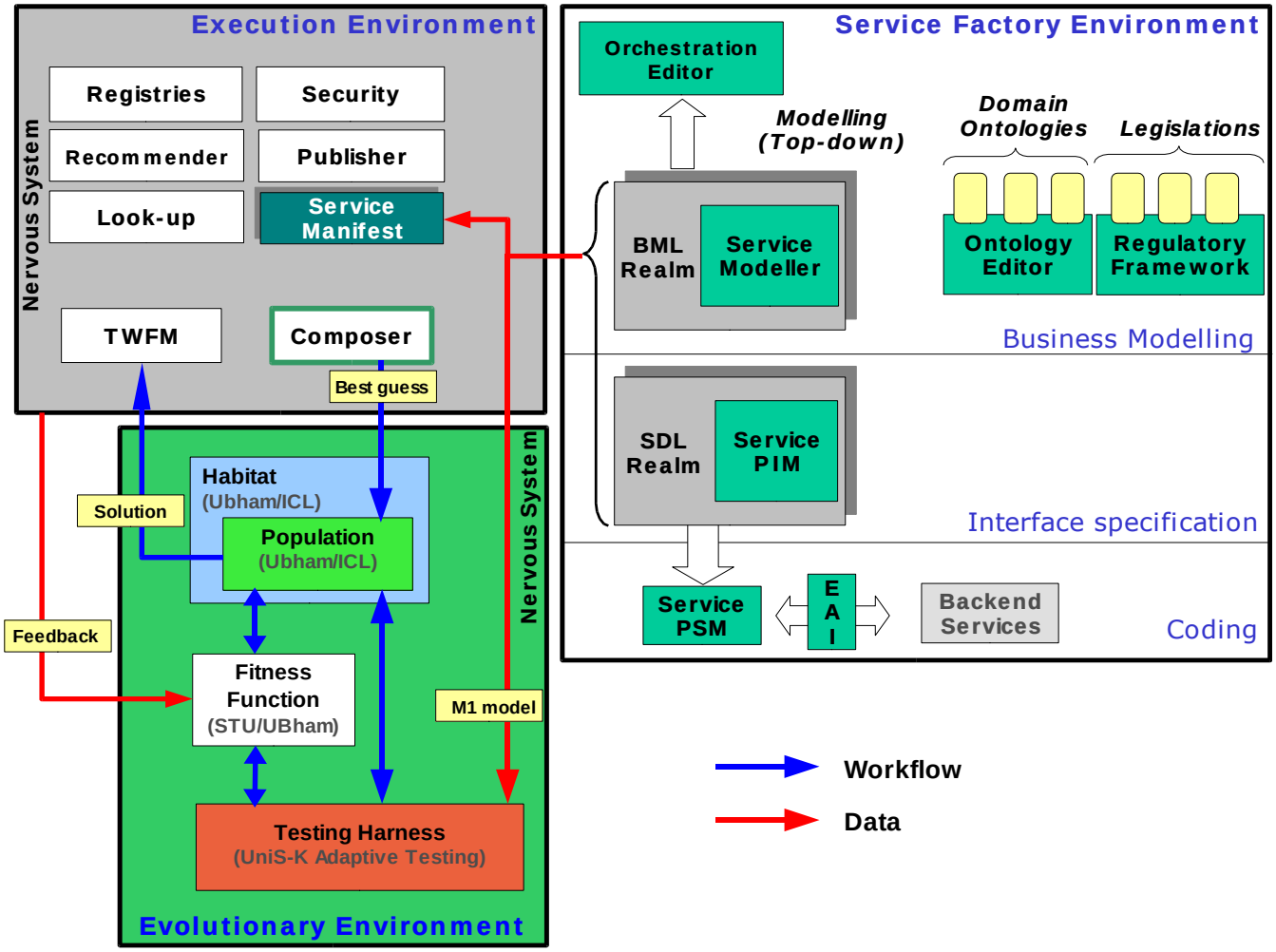


Fig 4. Evolutionary Environement integrated with DBE Core Architecture

Partner Roles – Evolutionary Environment

(This is not final. Details of simulations and roles to be discussed at upcoming meetings)

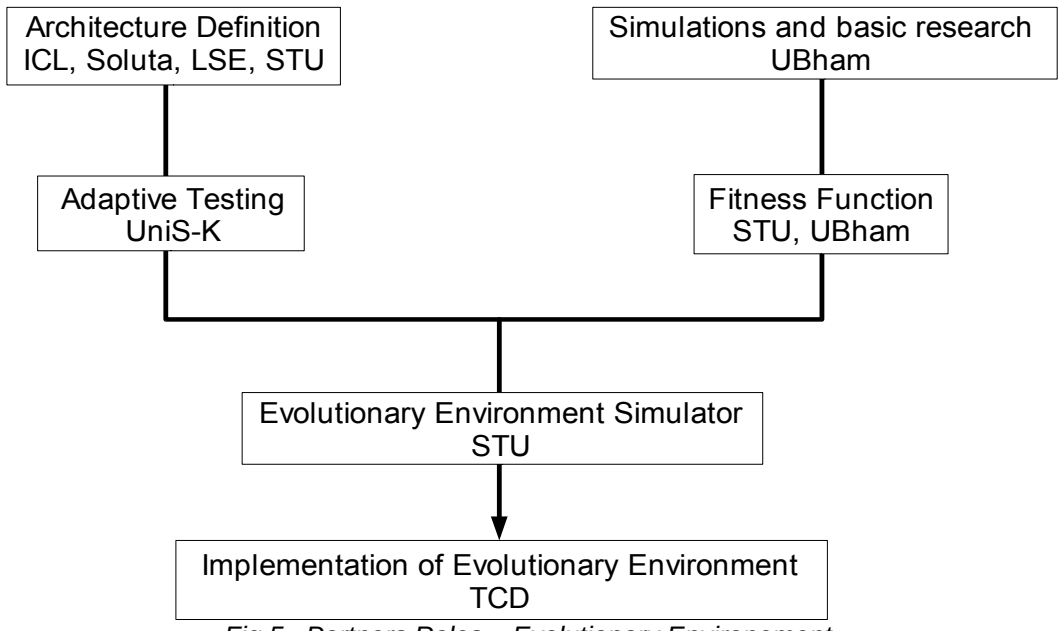


Fig 5. Partners Roles – Evolutionary Environement