Contract N° 507953

Workpackage 9
Model of Fitness Landscape

Deliverable 9.1
Report on Fitness Landscape

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

Contract number: 507953
Project acronym: DBE
Title: Digital Business Ecosystem

Deliverable Nº: D9.1
Due date: 30/04/2005
Delivery date: 26/04/2005

Short description:
This report examines scientific and computational aspects of the concept of fitness that is a fundamental driving force in evolution and therefore also in the biological simile that the DBE project consortium utilizes for the design and the implementation of an open infrastructure capable of supporting spontaneous evolution, composition, and adaptation of software components and services.

Partners owning: STU (Thomas J. Heistracher, Thomas Kurz, Giulio Marcon, and Claudius Masuch)
Partners contributed: STU, LSE, ICL, UBham, FZI, ISUFI
Made available to: DBE Consortium and European Commission

Versioning		
Version	**Date**	**Author, Organisation**
1.1	26/04/2005 (final submission)	STU
1.0	07/04/2005 (first submission)	STU
0.9	23/03/2005	STU
0.1	31/01/2005	STU

Quality check
1st internal reviewer: Miguel Vidal, Sun
2nd internal reviewer: Maurizio De Tommasi, ISUFI

Contents

List of Figures

List of Tables

Executive summary

This report examines scientific and computational aspects of the concept of fitness that is a fundamental driving force in evolution and therefore also in the biological simile that the DBE project consortium utilizes for the design and the implementation of an open infrastructure that is capable of supporting spontaneous evolution, composition, and adaptation of software components and services. As the role of STU is to interface productively between scientifically focused and computationally focused partners in the project, the present work intends to 'translate' concepts from science to computationally feasible approaches. Therefore, this report gives first an overview of the work *context* related to fitness to explain the interlinkage with other partners. Second, the concept of the *Evolutionary Environment (EvE)* is discussed in the light of the design-to-implementation process. Third, *fitness* and *fitness landscapes* are explained and the mapping from biological to computational and mathematical understanding is prepared. Next, a *mathematical framework* for fitness function is introduced that is also applicable outside the EvE. Finally, the prototypical *fitness landscape simulator* is described, which is currently based on genetic algorithms. We conclude with an outlook to future capabilities of the current fitness model.

Chapter 1

Introduction

Das Neue ist niemals ganz neu. Es geht ihm immer ein Traum voraus. [a]
(Ernst Bloch, 1885 – 1977)

[a] The new is never completely new. A dream always precedes.

This report is part of WP9, Model of Fitness Landscape, and relates mainly to work performed by STU in tasks S12, Mathematical Model of Fitness Landscape, and task C4, Simulator of Fitness Landscape, and also to a smaller extent to task C37, Simulator Evolutionary Environment, which recently commenced.

Scientific and DBE-related objectives

The *general scientific objective* of WP9 is to develop a scientifically sound framework for the application of fitness[1] that is a fundamental driving force in evolution and therefore also in the biological simile that the DBE project consortium utilizes. The *DBE-related scientific objectives* of WP9 are to study, plan, and prototypically implement functionality for the 'growing' of DBE services by means of evolutionary paradigms facing the challenge of applicability in industry-level software systems.

Aims of this paper – success criteria

This document provides helpful if it achieves the following objectives:

- to provide a conceptual model ranging from mathematical framework to integration aspects into architecture including a discussion of a prototypical implementation of the fitness landscape simulation;
- to show where fitness considerations 'come into the game' in the multi-disciplinary DBE project;
- to provide a conceptual basis for application of fitness function;

[1] See Chapter 3 for an in-depth explanation of *fitness* in the context of this work

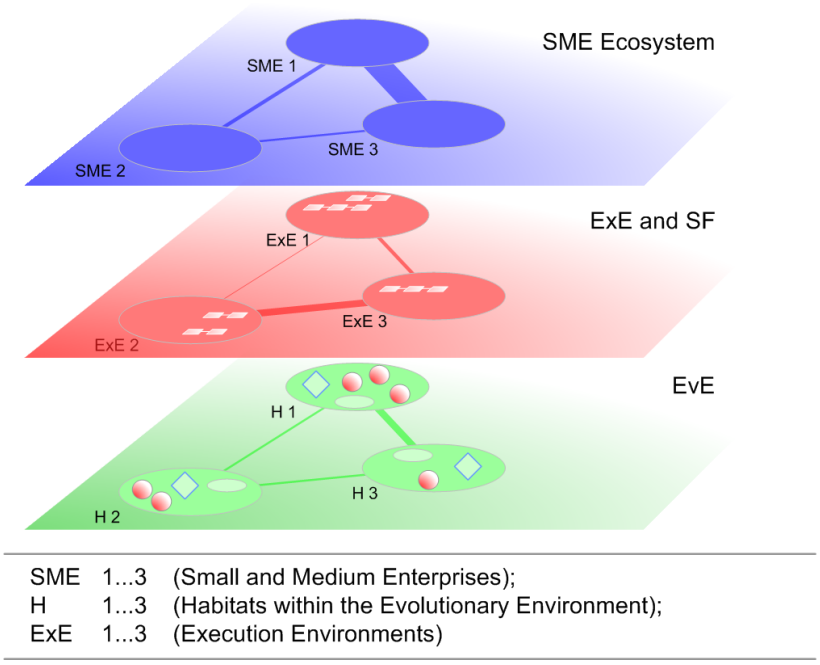

Figure 1.1: View on DBE Systems

- to offer a sound base for EvE service implementation;
- to build the basis for a DBE demonstrator that can communicate the benefits of the biological simile applied to a distributed (software) service environment;
- to provide the basis for determination of the best suited cost-function(s).

This report intends to interlink scientific and computational streams regarding the fitness mechanism and to provide the role of an information hub. Moreover, business concepts and requirements are taken into account to point this work towards a practical direction. A considerable period of time is needed in this stream to raise questions, to trigger discussions, to work on glossaries, to circulate position papers and the like. Therefore, parts of the effort in WP9 was made also for this important synchronization. This work represents the current status of related activities with the option for adaptation, specifically when considering the 'distributed implementation' of the DBE in phase II of the project.

DBE's BIZ, SCI, and COM system layers

There is no final decision within the DBE project consortium up to now whether the focus of work should be more on real-world services or on software services in the future, hence both scenarios are dealt with in this work. For a related discussion of short-term and long-term applications in the context of the DBE architecture, see [Fer05], *DBE Core Architecture* document.

The first scenario (*DBE of real-world services*) is that the whole system is based on companies which provide services like transporting goods, letting rooms, baking bread and so forth. These business processes are modelled in a Computation Independent Model (CIM) and then represented by a software box which enables handling. For example, we put a pair of shoes in a box with a label to be able to identify, stack, and ship them (the essential part of this combination is the shoes and not the box with the picture of the shoes on front of it). For a useful stacking and shipping it has to be asserted that the shoes optimally fit in the box and they need minimum space in the warehouse. This example is both simple and abstract but it points out the core focus in the DBE real-world intention. In the DBE project we try to include biological concepts to improve the e-business environment.

Somehow contrary to that is the scenario of a *DBE of software services*. Imagine the development of an open source operating system. A lot of programmers are working on several parts of this complex software service. If we are able to automate the testing for such a system in a way that new components can be added and tested without a lot of manpower, the operating system can be optimized along the fitness functions of the system. This scenario could also be a fruitful application area of a DBE.

Because of the fact that this DBE of software services does not rely on real-world business processes but more on computing functionalities, we consider the Execution Environments as the underlying base (see Fig. 1.1). In a way transparent to the user, biological concepts are applied to optimize the software components for usage in the SME ecosystem. The resulting interconnections on business level emerge between software producing SMEs, which are providing, e.g., atomic services for more complex software services. These can be considered and visualized as this SME ecosystem.

Most of the components in the DBE can be used independently of the focus on real-world services, or software services. Nevertheless, assessing the fitness of a shoe producer is different from assessing the fitness of a piece of software with regard to interfaces or testing of interaction. On the contrary, as long as both real-world and software services are encoded in the same *DNA* language, there are many similar considerations. Moreover, the *DBE of real-world services* uses also some software components to interact more easily.

This document describes a general framework for fitness with a focus on software services. In future, any service in the Digital Business Ecosystem (DBE) will include at least some simplistic software components thereby extending the present considerations to other application and usage scenarios like real-world applications should be easily possible.

In the following, the three systems of the DBE are outlined:

- *SME ecosystem*,
- *Execution Environment (ExE)*,
- and *Evolutionary Environment (EvE)*.

Partner	Interfacing topic
LSE	Science vision
ICL	Biological simile & high-level EvE architecture
TCD	Implementation of the FL
UBham	GA
Soluta	SF architecture
Sun	ExE architecture and implementation
ISUFI	BML
FZI	User profiling
TUC	Knowledge base
UniS	Model testing

Table 1.1: Partners and related topics

The *Evolutionary Environment (EvE)* includes mechanisms we know from biology into the DBE. The interconnected habitats, usually one per SME, are the scientific representation of the SMEs. Basic information about the SME, e.g. user profiles, the services provided and used by the SME, and all necessary information for optimization is resident in and linked to, respectively, the habitat objects (see also [Bri04b]). Beside the information about services and interaction, these habitats include so-called population objects. If a user requests a service, the habitat creates a population of services from the local service pool that could match the requirements and optimize them by using for example a Genetic Algorithm (GA). In the first step, the fitness of service chains is evaluated by the comparison of CIM strings. This EvE is one of the starting points of the biology $\Leftrightarrow$ computing mapping. This issue is discussed further in Chapter 2.

The *Execution Environment (ExE)* represents the structure proposed by the architectural contributions (see DBE Architecture Documents [Fer04, Fer05] and [HKM+04]). The direct interaction of services as in Fig. 1.1 is only a logical view because the currently proposed architecture implements a FADA-like system that is based on a proxy architecture. Nevertheless, the simplification in Fig. 1.1 could be a helpful way to visualize the mapping of the different systems of DBE.

Finally, the *SME ecosystem* is a kind of representation of the real business processes within the DBE. The regional opportunity spaces and the influences of the market could be evaluated on that level by a business simulation. The idea of the SME Ecosystem and the related mapping arose because something was missing to build a Digital *Business* Ecosystem.

Partners' relationship

Table 1.1 gives an overview of the work relation with other partners in the project. In addition, the work inter-relationship is depicted in more detail in Fig. 2.2. Note that partners from all major streams in the project are represented, once again stressing the interfacing role of the present work.

When applicable, the respective partners will be referred to in the following chapters of this document.

Document structure

As a main goal of this report is to provide a conceptual framework with respect to fitness, it is structured from 'general' to 'specific'. First, the *context* in which this work has been performed is explained and the inter-relations with the major streams in the DBE project are described.

Second, the concept of *Evolutionary Environment* (EvE) that has been developed recently in a joint effort by several DBE partners is explained. The understanding of the role of EvE is vital for the following discussion of *fitness* and *fitness landscapes* where both an identification of fitness aspects and a refinement of the description of the methodological requirements can be found.

Next, a *mathematical framework* for handling fitness issues in the DBE project is introduced and detailed. It enables an adaptive fitness calculus for best performance, even with respect to a single SME, and is flexible as regards the choice of the cost function. Its main applicability is within the EvE but it also can be useful for other streams in the project.

Following that, the *fitness landscape simulator* is detailed and the functionality and applicability for DBE services is demonstrated. The demonstrator exhibits the selection of candidate combinations of services from an existing pool and the related fitness optimization based on genetic algorithms. The goal of this functional prototype is mainly suitability assertion; the simulator is currently not directly interwoven with a DBE service notation, what is due to the fact that the CIM framework for DBE services is not stable at the time of writing. Currently there are, however, intensive discussions with related partners.

Finally, the report concludes with an *outlook* to next capabilities of the current fitness model.

Audience and usage of this document

This report is intended for stakeholders from all the streams in the DBE project to facilitate the understanding of the applicability of the suggested biological simile. Specifically, for

- *scientific* stakeholders, it should denote a common view of fitness function that is based on current knowledge and methodology, for
- *computational* stakeholders, it should demonstrate that this approach to fitness can be implemented speaking in terms of computability and also in terms of distributed operation later on, for
- *business* stakeholders, it should sketch the benefits of a fitness concept in inter-business application environments and show this via the functionality of a working simulator, and for

- *socio-economic* stakeholders it should provide some arguments for practicability and added value of the DBE approach.

As regards *usage* of this document, anybody is welcome to read Chapters 1 and 6. For mere formal and scientific aspects, Chapters 4 preceded by Chapter 3 are most relevant. For architectural implications, Chapters 2 and also 3 are considered essential. Regarding implementation, Chapter 5 preceded by skimming Chapter 4 might be helpful. Considering business implications, an overview from Chapter 5 and then Chapter 6 should mediate applicability of the model suggested.

Chapter 2

Evolutionary environment

The strange thing about evolution is that everyone thinks he understands it.
(quoted in Stuart A. Kauffman, *Investigations*)

2.1 Biological systems

The digital business ecosystem idea, as originally conceived by Nachira [Nac02], draws concepts from the current understanding of systems in nature that we generally call *biological systems*.

The extension of biological concepts to the business domain maps intuitively the entities and processes of markets onto the concepts of ecosystems and evolution. The digital component adds a dimension to this metaphor, increasing its expressivity and opening new paths of investigation.

Already before creating such mappings, computer scientists and mathematicians were interested in imitating the optimisation processes of nature, at the same time biologists and chemists were revealing the related mechanisms.

A new branch of computing was then born, also if not recognized at the time, called natural or bio-inspired computing, with areas of research such as genetic algorithms (inspired by evolution), cellular automata (life, plants' stomatas, patterns of seashells), emergent systems (ant colonies) and neural networks (brain). Each of these areas is a completely independent field with lots of specific scientific journals, specific conferences and large communities. But, nevertheless, they have common research goals, usually dealing with investigation of the limits of computability, breaching the constraints of the models of computation taking advantage of parallelization and the search for innovative optimisation approaches.

This section briefly introduces biological systems with the aim of giving a background for the understanding of the biologically-inspired optimization methods presented in the next section.

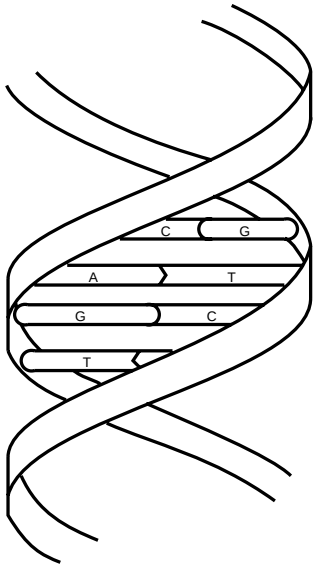

Figure 2.1: The DNA is a complementary double helix of a four letters alphabet, encoded in a chemical code, that carries hereditary information.

2.1.1 Cells, DNA, RNA and proteins

Long before the advent of scientific methods, our ancestors gave the name *life* to several things that apparently (at least for them) had nothing in common. But our current knowledge tells us that all of these things are made of cells, certainly not diminuishing the attractiveness of the subject, but on the contrary raising more and more questions.

Cells are the units of which living beings are built: although all the cells share the same fundamental mechanisms, they give rise to an incredible variety of species - today, on the Earth, in the numbers of millions. These living beings all absorb raw materials from their surroundings and use them to generate other copies (or similars) of themselves: it is in this reproduction that they perform the magic of the *heredity*, or the continuation of the species through the spreading of information specifying the characteristic of the progeny.

It is even more amazing that inside every cell, from whatever organism it comes from, the information is maintained inside the same kind of chemical code, that for its structure is called deoxyribonucleic acid – DNA (chemically it is composed by alternating phosphate and sugar groups, kept together by hydrogen bonds between pairs of organic bases). This chemical structure is organized in such a way that we can just see it as a linear string in an alphabet of four characters: these are the four organic bases adenine, cytosine, guanine and thymine (in short, A, C, G, T). The real physical structure is a double twisting helix where the two strands are complementary in the sense that there is a bond between every two subsequent bases of the two strands and these bonds can only be guanine-cytosine or adenine-thymine (this condition is called complementarity). Figure 2.1 represents the double helix with complementing bases.

The bond within the bases on the two different strands are weak compared to the

strong sugar-phosphate links of the "backbone", enabling them to easily split apart: a single strand acts then as a template for the production of another complementary strand. Through this process not only new DNA is synthetized (or replicated), but also decodification happens, through which the instructions contained in the DNA itself are put into practice.

Decodification happens through these two main stages:

1. First of all the DNA is *transcripted* into *ribonucleid acid* - RNA, a similar polymer. RNA has a backbone that is very similar to DNA, only that in the sugar group there is an additional OH group; additionally, instead of the base thymine, there is the base uracil, which differs from thymine by the absence of a CH_3 group. RNA presents itself in a single strand: transcription is done by complementing a strand of the DNA.

2. From RNA a protein is synthetized. Proteins are the workers that perform basically all that is happening inside a cell, and can exhibit complex behaviours coordinating by making and doing bindings between themselves. A protein is a long polymer chain of aminoacids that they can be viewed as an alphabet of 20 characters. The *translation* from RNA to proteins is carried interpreting the bases alphabet: each 3 bases codify for an aminoacid. The code is redundant ($4^3 = 64 > 20$), but some sequences are used to signal where to start the translation and where to stop and nevertheless redundancy helps robustness to transcription errors. This code is called the *genetic code*.

It is amazing how all of this happens in a cell, and moreover in the same way in all the cells. Additionally, all the transcription-translation process is done by proteins, that themselves are coded into the DNA.

2.1.2 Genotypes and phenotypes, individuals and environments

We have already seen that the DNA contains the blueprint for the creation of individuals. We have now to understand how information is inherited: discoveries in this field were made long before the structure of the DNA was understood, with studies conducted with the aim of understanding the spreading and mutation of information to offsprings (Mendel's experiments). These functional units of inheritance were called genes: later they were identified as segments of DNA, usually coding for just one protein. The genome of an organism is the set of its genes, and is all the inheritance information that is carrying. The *locus* of a gene is its site in the genome, while *alleles* are alternative forms of the same gene in the genome (alternative forms have minor differences).

The *genotype* of an individual is the set of the alleles that form the genome of an individual, that is all the genes and variations that he has. In *diploid* organisms, for instance, the genetic information is present in two copies that can be slightly different. Recent discoveries have led to different definitions of gene, as after RNA transcription, several post-transcriptional controls can for instance cause production of different types of proteins (by alternative splicing, editing, etc.), or completely suppress the translation.

The *phenotype* of an organism is its physical appearance and constitution; sometimes by phenotype only a particular physical trait is denoted. It is important to understand that a phenotype is not just the expression of a genotype, but is also influenced by environmental factors. A particular phenotypic trait can often be related to one or several genes, explaining in this way their functionality.

In the transmission of genetic information, mutation can take place, changing the genotype and frequently also the phenotype of an organism. The mutations that can randomly happen in a diploid organism are:

- point mutation: a single site (base) in the genome is modified;
- inversions: a segment in the DNA is inverted (changed of direction);
- deletion: a segment in the DNA is deleted;
- translocation: a segment is moved from a location to another one.

Mutations, together with natural selection pressure, produce evolution, that is produce changes for the better. Mutations, being random, can in fact cause negative or positive effects: of course every effect has to be evaluated inside the environment where it takes place. Natural selection helps better mutations to survive, as individuals that had negative mutations will have less chances to reproduce.

We have also seen that the phenotype is not only the expression of the genotype, but also of the environment: the branch of biological science that studies individuals, communities and ecosystems-environments at a higher level is called ecology. In ecology the first principle is that each individual continuously relates with several other elements, that are collectively called its *environment*. An ecosystem is a system where this relation holds between living beings - *biocenose* - and their environment - *biotope*. An *habitat* is the biotope where a particular species (a set of organisms capable of interbreeding) lives and grows. A *niche* specifies how a species relates to its habitat.

2.2 Biologically-inspired optimization

We present here some biologically-inspired optimization methods that have been considered in the DBE project, referencing the corresponding partners' work.

The main approach followed in the project is the use of evolutionary optimization and in particular of genetic algorithms. These algorithms, following a biological simile, try to solve an optimization problem in the following way: every candidate solution is encoded into some particular form, called "individual", that is its genetic representation. From a genetic representation of an individual a transformation into a a phenotype (the candidate solution itself) is done and this is evaluated for its fitness. Strictly speaking in ecological terms, the fitness is the proportionate contribution that individuals can give to future generations [BHT96]. One of the main aims of this document is to explain what has to be taken into account for this fitness calculation and to provide means of doing it. A typical genetic algorithm usually follows these steps:

1. generation of a random population (set of individuals);
2. calculation of the fitness for the individuals in the population;
3. creation of the next population by:
   (a) selecting some individuals for reproduction;
   (b) doing a crossover between the selected individuals;
   (c) mutating new individuals;
4. depending on termination condition go to step 2 or exit.

The termination condition is usually a check on the number of iterations and on the reaching of a good enough individual. The partner that is used as a reference for the utilization of genetic algorithms is the University of Birmingham (UBham): in our case UBham gave the background information for the development of the fitness landscape simulator (see Chapter 5). As the deliverable of UBham is set for month 18, some ongoing work has been made available internally to the consortium and has been published, or is in the process of being, in journals [RH04, Row05].

Another biology inspired optimization in the project is the one achieved through the distributed intelligence system researched by the Imperial College London (ICL) and with some preliminary results already presented in deliverable D6.1 [Bri04a]. The aim of the distributed intelligence is to modify the behaviour of the operators of the genetic algorithm to speed up the evolutionary process. This is achieved through clustering and migration mechanisms between habitats, where the habitats are small-medium enterprises fully equipped for participating in the DBE network. A cluster should ideally group all the companies along opportunities spaces (business sectors).

If we think of a Small and Medium Enterprise (SME) hosting several populations of running genetic algorithms where each population is the current status of the optimisation process looking for a fit service for a certain request, as all of these populations are in the same SME, it is natural to see the SME as a habitat (more precisely a biotope, but habitat is the currently accepted term). A cluster is then a group of habitats sharing something (something similar to a biome). The movement of individuals from one habitat to another inside a cluster would then be a migration: this is actually the mechanism that will be used by the distributed intelligent system. For more information about biological simile in the DBE see also [Bri04a, Bri04b, BD04].

A more technical view on a distributed optimization system is achieved by Trinity College Dublin (TCD) and presented in deliverable D23.1 [Dow05]. TCD proposes an innovative distributed algorithm called Collaborative Reinforcement Learning that performs an optimization process without any global knowledge, achieving the aim of a completely distributed optimization. Although their presentation is quite technical, the underlying biological model is that of emergent systems where the agents, from simple rules of interactions between them, give rise to a complex emergent behaviour.

Work package 12 tried to address from the optimisation point of view two very important issues in the project, namely the criteria for pairing up companies with

similar profiles (clustering) and the way of finding the best solution to a service request [PGT04a, PGT04b]. The two issues were addressed developing two different models for the digital business ecosystem and using neural network approaches. These and other models are still under development by University of Surrey (UniS) and are not yet ready for integration.

A good overview of the work in progress in integrating these biological inspired ways of optimization in the digital business ecosystem project, together with hints of what now evolved into the science team agenda, is provided in the London School of Economics' deliverable D18.1 [Din04].

2.3 EvE architecture in DBE

Starting from the ecosystem simile, the Evolutionary Environment (EvE) term emerged in the DBE. From the initial ideas proposed by Briscoe [Bri04a, Bri04b], a clearer and broader EvE architecture picture emerged. Fig. 2.2 shows a brief overview on the EvE, focussing on the fitness point of view. A global overview on the EvE, including relations

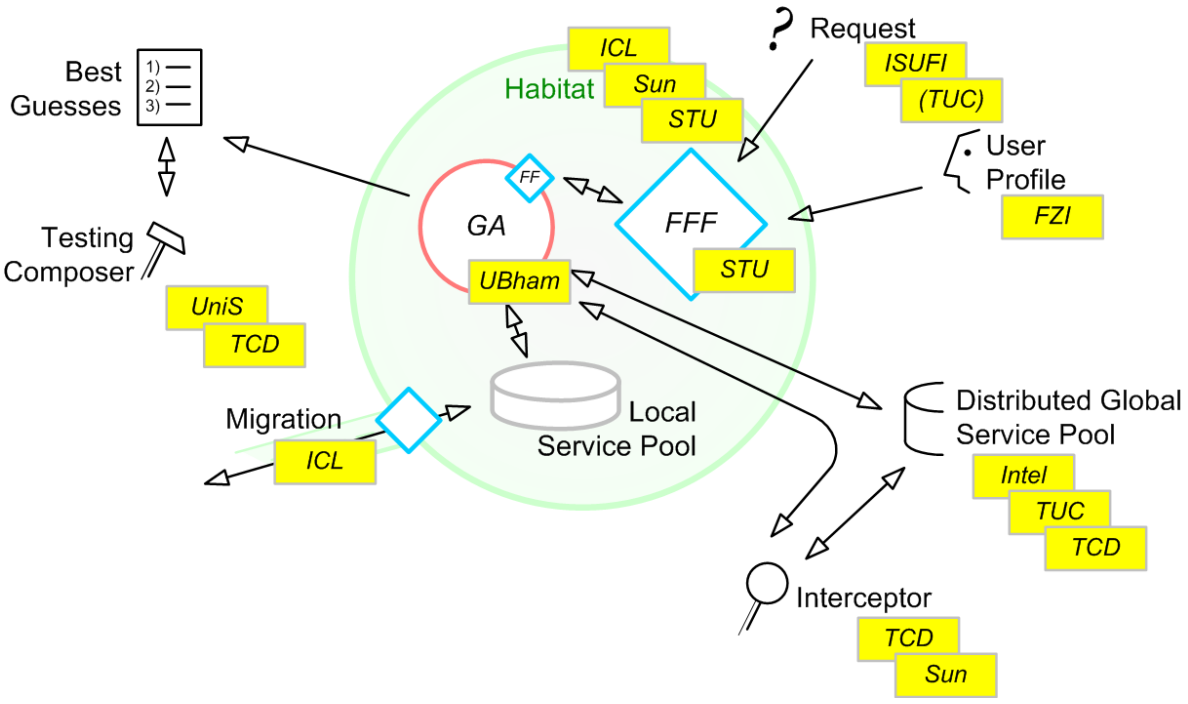

Figure 2.2: Evolutionary Environment (Clipping)

and interfaces as well as its position in the overall DBE architecture can be found in Ferronato's DBE Architecture Documents [Fer04, Fer05].

The Evolutionary Environment (EvE) is the community of habitats in the DBE. A habitat is the logical place where the evolutionary optimisation and selection processes

of the DBE can happen. As mentioned before (see Chapter 2.2), each SME has its own habitat. At this moment, a one-to-one relationship between an SME and its habitat is envisaged. From the computing point of view, the habitat will be implemented as a DBE Service on top of the Servent[1] architecture. Given ICLs architectural definitions, SUN and STU will implement and adapt it to the DBE needs.

As can be seen in fig. 2.2, the Genetic Algorithm (GA) and the Fitness Function Framework (FFF) are the two core components of the habitat. The Genetic Algorithm (GA), that UBham is in charge of defining, is the driving part of the evolution. Within the GA part of the habitat, the optimisation process is running. Here, populations of service chains are created and evaluated. The gene pool for these service chains is the global service pool, which is the DBE distributed Knowledge Base (see [Fer05]), and a local service pool. The local service pool holds services that migrated from other habitats.

This optimisation process then delivers a ranked list of "best guess" solutions. This most fit solutions then are given to the automatic (or manual) composer (TCD), to combine and "glue" the services by adding work-flow information and links between the services' interfaces. At this moment, this process is assumed to work automatically just within certain limits.

After that we can think of an integration test done by the testing framework of UniS. As the application of testing to the EvE is not clear up to now, this is only a probably useful integration point for future cooperation and mirrors some fruitful discussions about integration in the first half of the project.

The Fitness Function Framework (FFF) presented in this document acts as the evaluating part of the evolutionary algorithms. The FFF will provide a fitness function for the GA. This fitness function is adapted by current SME's data. Conditioned by parameters like usage history, preferences, profile, environment, etc., a specific fitness function is instantiated. This fitness function is adapted and parametrised by information and data from the languages part (ISUFI, TUC), and FZI's user profiling mechanisms.

The fitness function will also take into account the run-time data of each service, thus considering the services' historical data: these are gathered through the interceptor framework that will be developed by SUN. Currently, historical data are collected through the AXIS-based interceptor done by TCD, but in the future, as the chosen approach is Servent-centric, this component will be substituted by SUN's Servent-based interceptors.

Fitness evaluation currently can occur on three levels:

- *service* level: semantic matching, run-time service data, etc.;
- *user* level: user profile (user data, preferences, user history, etc.);
- *ecosystem* level: service visibility, networking behaviour, availability, etc.

As the fitness function actually does the evaluation of the solutions created by the GA, it is a core topic for the evolution of the services and the ecosystem itself. The following chapter provides a more detailed discussion.

[1] ServENT = SERVer + cliENT

Chapter 3

Fitness

> *The shoe that fits one person pinches another; there is no recipe for living that suits all cases.*
> (Carl Jung, 1875 – 1961)

This chapter summarizes the conceptual model of fitness developed within WP9. Most of the results are a consequence of several discussions with partners from the business, computing, socio-economic, and science domain. Although, a lot of work was done to agree on a conceptual model, the presented material is just a starting point for applying fitness concepts to the DBE. Nevertheless, we focused on the practical applicability to the DBE and present several use cases for the application of the mathematical framework for fitness function we introduced above.

3.1 Biological fitness

Evolution and therefore also fitness are concepts to somehow optimize systems. Nevertheless, no population of organisms can contain all possible genetic variants and therefore natural selection is unlikely to lead to the evolution of perfect, "maximally fit" individuals [BHT96]. This chapter will summarise fitness in biological terms, explain why WP9 is called *Model of Fitness Landscape* whereas we are concentrating mostly on fitness functions and, finally, will emphasise the constraints when applying fitness and evolutionary concepts to non-biological systems like the DBE.

3.1.1 Fitness Landscapes and previous work on Fitness

Section 2.2 already explained some of the biological background on biologically inspired optimization. Moreover, this section will give a number of definitions to explain the biological simile of fitness in literature and science. We are starting with the definition of fitness in pure biological systems:

> The fittest individuals in a population are those that leave the greatest number of descendants relative to the number of descendants left by other, less fit individuals in the population. [BHT96]

Consequently, those individuals that leave the greatest proportion of descendants in a population have the greatest influence on the characteristics of the population. This biological concept of fitness was adapted by computer scientists and used in genetic programming as an *objective function f* for evaluating artificial individuals, e.g. string creatures. In 1989, David E. Goldberg defined Fitness in his book entitled *Genetic Algorithms* like this:

> Intuitively, we can think of the function f as some measure of profit, utility, or goodness that we want to maximize. Copying strings according to their fitness values means that strings with higher value have a higher probability of contributing one or more offspring in the next generation. [Gol89]

Although the goal of optimization along a fitness function is to optimize to a global optimum, no matter if in biology of computer science, the essence of natural selection is to produce "the fittest available" or "the fittest so far", not "the best imaginable" [BHT96].

Probably the most plastic way to describe fitness functions and global as well as local maxima are fitness landscapes. In [Bak96], Per Bak stated that perhaps our ultimate understanding of scientific topics is measured in terms of our ability to generate metaphorical pictures of what is going on [Gav04]. Maybe therefore *Model of Fitness Landscape* is mostly about fitness functions and their application to the DBE.

> In its simplest form a fitness landscape can be seen as a plot where each point in the horizontal direction represents all the genes in an individual corresponding to that point. The fitness of that individual is plotted as the hight. [BP02]

This definition of fitness landscapes is very simplistic and a lot of work has been done on fitness landscapes which goes well beyond this definition. Nevertheless, fitness landscapes are a visualization tool and they are often introduced as a three-dimensional visualization of a two-parameters Fitness Landscape (FL) (a example is given in figure 3.1). Furthermore, Kauffman for example did work on fitness landscapes over N-dimensional Boolean hypercubes [Kau00]. To keep things simple we explain fitness issues mostly by three-dimensional visualizations.

During the research on WP9 we realized that the title given to the work package, *Model of Fitness Landscape (FL)*, was not reflecting the actual research, as insights on the Fitness Landscape (FL) can be achieved only through research on the fitness function itself. Most of the efforts on understanding the Fitness Landscape (FL) were in fact aimed at studying the fitness function, being the first one just a representation of the last one. Moreover, the conception of fitness in the DBE and the later presented FFF will most probably end in an high-dimensional FL for the different application areas of this concept. However, the three-dimensional visualization is sufficient to describe the principals to a broad audience.

The previous definitions show the close connection of fitness and fitness landscapes but most of the work in this work package was done on defining a fitness function.

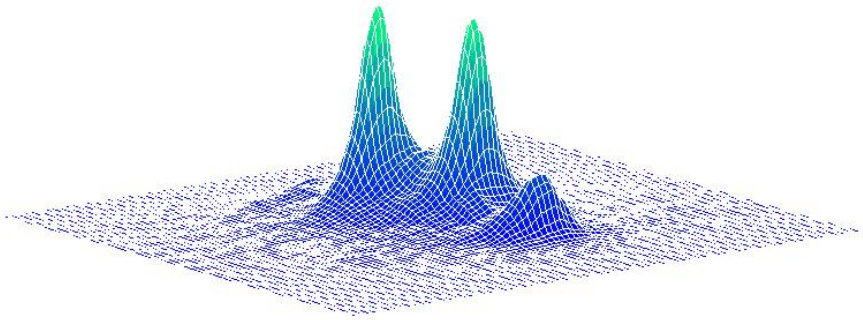

Figure 3.1: Three-dimensional visualization of a two-parameters Fitness Landscape (FL)

Undoubtedly, the FFF, presented in this report results in a FL and therefore we use the term fitness function and fitness landscape

From the several definitions of biological fitness we can clearly argue that fitness significantly depends on the *interaction of the phenotype with the environment*. Nevertheless, S. Wright's 1932 original formulation of fitness landscapes represents fitness of gene combinations [Wri32]. Moreover, fitness that has genetic bases could be seen as the most important one because it is changes in genes that make adaptation and innovations permanent [Gav04].

How these fitness concepts could be applied to the DBE and where these make sense and where not will be discussed and shown in this report. Note here that – no matter whether in biology, computer science, or in the DBE – fitness is a *relative* not an absolute term.

3.1.2 Constraints and applicability of biologically inspired Fitness

Before we go into detail regarding the constraints of the biological simile in the DBE, specifically for the fitness simile, a recapitulation of changes in the behaviour in biology itself is given. In the past, the rules for the survival of a species were mostly set by the environment the individual and its population lives in. Therefore, also the definitions of fitness strongly rely on the influences of the environment, on the individual or population. With the growing complexity of individuals and the increasing capabilities of these individuals, the environment is no longer this crucial for survival of (at least) the human species. If the fitness of a group of human beings is low, the group can change the environment it lives in to a certain degree such that its fitness increases.

Although we are starting with small services in DBE, these services have still an influence on their environment. On one side, the success or failure of a service has an impact on the strength of the service provider and service, respectively, and on the other side the usage of a successful service has also an impact on the user of this service.

Knowing about the definition of fitness in biological terms on one side, but also knowing about the objective of the project to take biological simile and bring it to a practical level on the other side, we are talking about fitness in this paper dependent on a certain context (see also Chapter 4) and in a broader way than defined in biology. The different anchor-points of fitness as well as the mathematical framework takes some biological inspired ideas and combine them on a practical level behind the term fitness.

No matter if we take genetic algorithms or other parts of evolutionary computing it is the trial to take some of the biological concepts to improve current state of the art in different fields. This is always a simplification and so the concept of fitness and evolution in DBE is also a trial to apply at least some biological inspired techniques to solve issues within the project. To make the concepts implementable the computational simile will often not be an exact translation from biology to practice, even though the same terminology is used.

3.2 DBE Fitness conception and interfaces

Starting with the vision of biology inspired optimization and the idea of introducing evolutionary concepts into the DBE this chapter illustrates the core concept and initial fitness considerations in the DBE including interaction points with partners. The Evolutionary Environment (EvE) is at the moment targeted at providing an alternative high risk approach to the recommendation system (WP17 Composer). At the beginning of the project the distributed fitness framework was targeted at enhancing the recommender system. Because of the new insights, as the SBVR approach, this initial plan changed toward a parallel development which focuses on providing a distributed optimization system based on evolutionary algorithms. Breaking WP9 down to its core functionality it is supposed to provide the user the best solution according to a certain request, no matter if the solution is one or a number of composed services. Beside that it has many other capabilities and extension possibilities. Nevertheless, we focus on the principal purpose before adding additional functionality.

As a result of the biology inspired optimization mostly genetic algorithms are used in the DBE to optimize and rank services by the use of a fitness function. This does not mean that other search strategies are not feasible for the DBE, but the current decision is on these genetic programming (see also WP8 Dynamics, selection and Aggregation). For evaluating the performance and suitability of different search strategies, beside others, the EvE Simulator was proposed where it should be possible to plug in another algorithm with little effort. Moreover, the fitness function framework proposed here and the concepts of comparing services are also applicable to almost all other search algorithms. That results from the fact that a fitness function is an objective function which assigns a relative value to services for comparison reasons. As optimization always needs a kind

of objective function the results presented can be easily applied to other optimization algorithms as well.

3.2.1 Usage scenario and interfaces

Figure 3.2 describes the process and data flow during a simplified optimization process. Starting from a request for a service, which is decomposed in sub-services or single features, this request acts as input data for the fitness function.

Together with selected data from the user profiling the fitness function framework builds then a fitness function for the optimization process. Fed by data from the local service pool, for initial populations, and fed by the distributed global service pool, for a broader spectrum of services and therefore a better search result, the optimization process finally comes up with a ranked list of service combinations. One of these service combinations is chosen by the user and after composition it is executed on the servent. As a long term goal, the system should find both, single services and service combinations and the recommender tool and the EvE will be presented as one integrated recommendation system. This is the minimal scenario of a request-based optimization process using the fitness function framework described in Chapter 4.

By using the interceptor framework of SUN Microsystems Iberia (SUN) it should be possible to enhance this use-case with a more dynamic component. Because of the fact that this is only for the long term implementation it is outlined here only briefly. During runtime the interceptors could detect problems of some services and so we can think of a more or less dynamic and automatic exchange of services (see also section 3.5). If a service performs not properly, the data gathered by the interceptors indicate a change and the EvE, or an other component that has to be defined, triggers a new fitness function with hard constraints for the technical interfaces and searches for an alternative service. If the optimization comes up with an almost matching service, this can be exchanged and tested automatically by the same procedure as the first one and if it passes the test the defective service is replaced by the new one.

Although this second part of the use case does not sound practical enough to be implemented, it could be a good way to change one instance of a service of a company with another service of the same company having exactly the same behaviour. That means also a higher operational availability of the services. Moreover, it could be a process for introducing new versions of software components automatically. Also, we here see the practical influence of the environment to the instantiated service phenotype currently running, as the reasons for exchanging parts of a service can be result of failures in the software infrastructure or underlying network components.

The description of the processes in this section are only a vision of future functionality. The ongoing work in WP9 will detail the scenarios and also the interfaces to the partners. Moreover, a lot of the issues go along with the definition of the EvE which is work in progress.

It is interesting to notice that this approach in service composition resembles more a neo-darwinist approach, as evolution is not obtained simply through random mutation and selection pressure, but through composition of services in what is the DBE equivalent

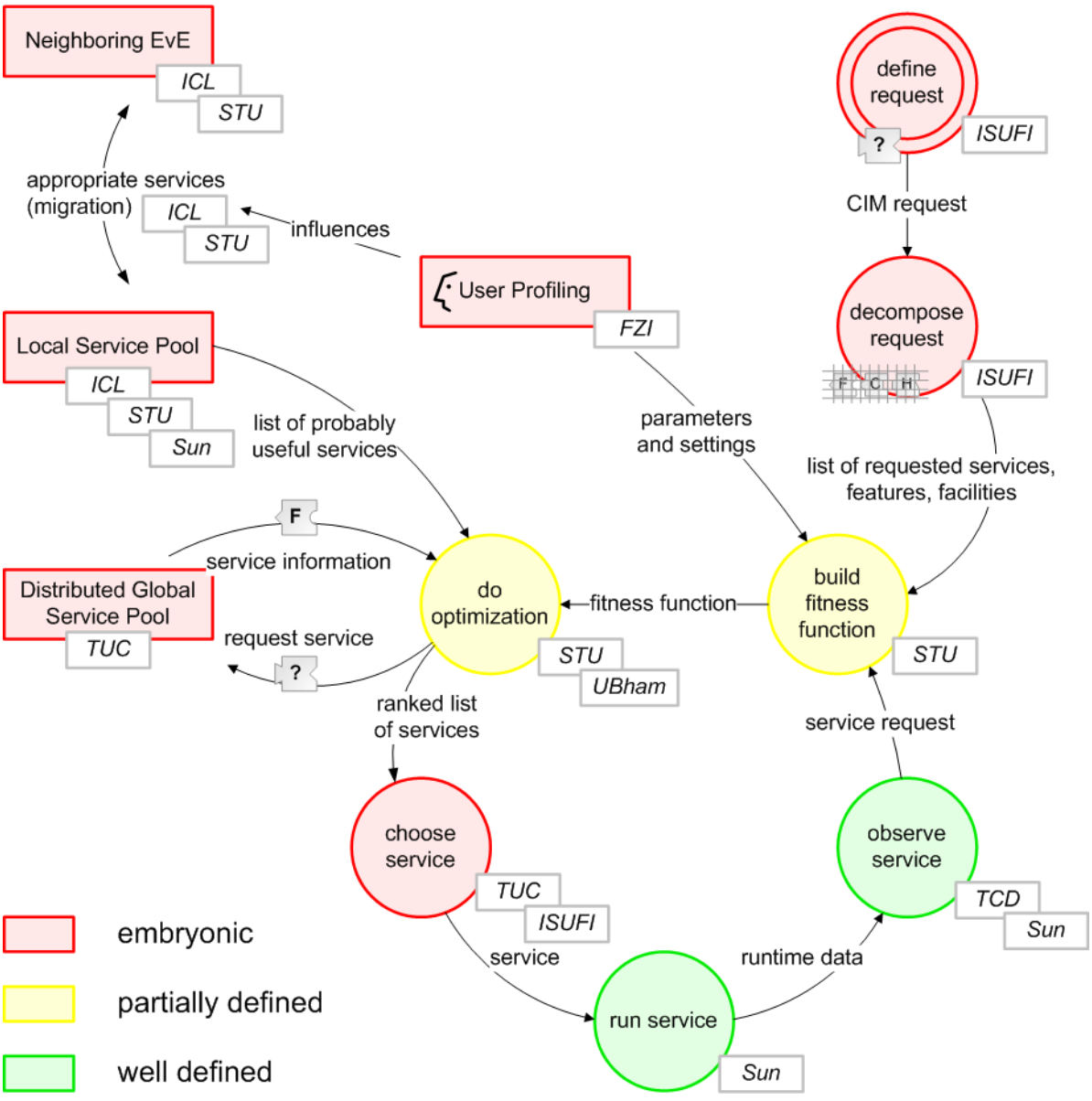

Figure 3.2: Processes and data flows during a simplified optimization process

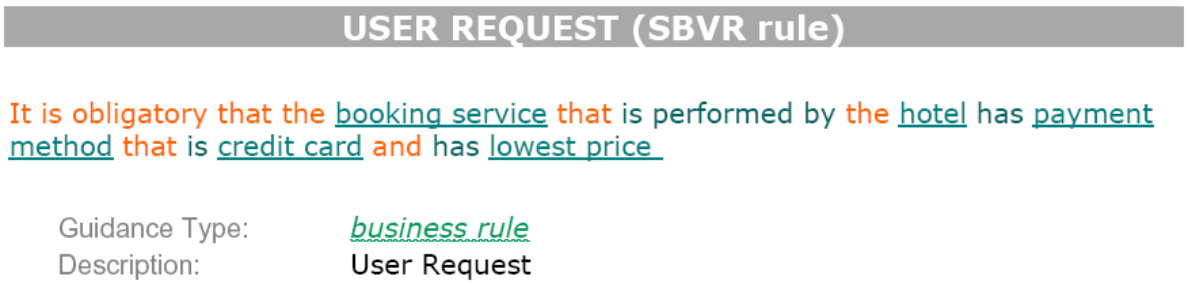

Figure 3.3: Simple and preliminary user request formulated as a SBVR rule

of the symbiotic merger process [MS02].

3.2.2 Data gathering and basic fitness aspects

One of the main research scenarios of WP9 is provide an evolution-based framework for finding and creating, respectively, best fitting solutions based on a user request. Consequently, the central issue is to find and create, respectively, services which provide the functionality specified in a users request. At the moment DBE has two approaches which could express these requests, namely Query Metamodel Language (QML) (WP14 DBE Knowledge Base and WP15 DBE Business Modelling Language) and the higher risk approach of SBVR (WP15 DBE Business Modelling Language). Since both, EvE and SBVR are long term oriented and higher risk than Recommender (WP17 Composer) and QML, the current focus of research in WP9 is on expressing requests and service descriptions in SBVR for usage in the EvE and the fitness function simulations. For the long term a unique user interface for requests have to be found to make the different systems and languages transparent for the user. Note here that the current implementation of the fitness landscape simulator does not take SBVR into account, because SBVR is at the moment in a very preliminary development stage.

Thus the first fitness criterion in all matters is to find services with the same functionality as the user specifies in his SBVR request. A very simple and preliminary user request is given in figure 3.3. The preliminary request is formulated as a SBVR rule. As a SBVR rule could be decomposed in fact types (i.e. terms defining business elements and expressions describing structural or behavioural aspects), e.g. hotel has payment method, the optimization process could avail of this feature (see also *decompose service* process in figure 3.2).

Before we can express such a request, a set of vocabularies have to be defined for the terminology. The basic request is for a booking service and consequently before all the fitness parameters like payment method and price are taken into account, a service has to be found with the description *booking service*. This initial feature comparison acts as the first part of the fitness function in most use cases.

Regarding the data gathering for all the fitness comparisons it is not completely clear until this moment where to store the data exactly. The interceptor framework could

provide useful data. Nevertheless, we assume that the initial comparison and evaluation of fitness will be reducible to a comparison of an attribute like *price* from the request with the equivalent attribute in the service description, no matter if these attributes are stored in the service manifest or somewhere else. Nonetheless, the access to service attributes like *price* has to be assured by the DBE infrastructure.

3.3 Where Fitness comes into the game

The following subsections identify potentially useful anchor points for applying a fitness and evaluation strategy, respectively. Whereas the focus in the first period of the project was mostly on the fitness function in a fitness driven optimization like a genetic algorithm, other application areas like the evaluation of migrated services and the fitness of SMEs are also described in some DBE use cases.

3.3.1 Fitness driven optimization

The first and probably most common understood fitness usage is within the population objects in the Evolutionary Environment. As the population object is another term for applying a genetic algorithm for optimization, the fitness function is used to evaluate the fitness of services during the evolutionary process in this genetic algorithm. As described in section 2.2 the fitness function is the objective function for the optimization procedure.

Starting point of the optimization in this case is the request for a service made by the user. Section 3.2.1 describes the request, decomposition and optimization process in more detail. Although the request is a central point for this optimization other data is also used for improving the optimization (see figure 3.4). For each user request a special genetic algorithm is used to find the optimal solution and a ranking respectively. Consequently, many optimization procedures can be performed in parallel in a habitat.

When the request is set and a fitness function is built for the optimization, data is requested from the outside for evaluation of the fitness of the service combinations. First, the user has set his request which is available to check out the right features or attributes. Second, service data is used from the local as well as from the global service pool to evaluate history data, previous user feedback etc. Finally, the user profile is used for user specific data like preferences for certain features, for example.

After reaching a threshold or after a given time the optimization comes up with a number of ranked services or service combinations. Then the user could end the optimization by using a service out of this list. We can also think of pause the optimization and restart it when new services are available. Furthermore, we can think of an enlargement of the search space to find better fitting solutions.

3.3.2 Evaluation of migrated services

Migration is one of the biological inspired concepts introduced into the DBE [Bri04b]. In biological terms individuals migrate from one habitat to another to find a niche they

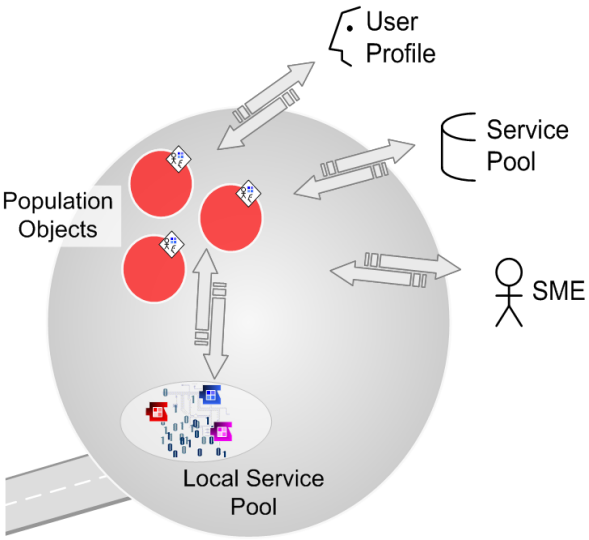

Figure 3.4: Request based optimizations within a Habitat

can live in. Analogously, services or service pointers, respectively, migrate from habitat to habitat (see also Chapter 2) to find SMEs with similar user profiles where they are probably useful and stay in the local service pool. The exact definition of migration is not totally clarified at this moment (for further details see also [BD04]).

The migration of services depends on the wishes of the SMEs, whether they want to share data about service usage etc. or not - migration disabled, and on a kind of bouncer function at the *entrance interface* of the habitats. A restriction of migrating services is needed because of two reasons. First, there is not unlimited memory for the storage of services in the local service pool. Second, only probably useful services are stored in the local service pool. Otherwise the service pool runs out of space without comprising a satisfactory number of services which probably have added value to the SME.

Although this bouncer function is not a fitness function in the biological sense, the fitness function framework can be used to build "fitness" functions for guarding the migration of services. Consequently, before a service even has a chance of being evaluated for fitness relative to a request, by the population object, the user profile is used to evaluate if the migrating service is probably useful for the SME; if not, it is rejected. Furthermore, we could think of a direct migration from the global distributed service pool, e.g. for updates reasons).

3.3.3 Fitness of SMEs

During several discussions frequently the fitness of SMEs was mentioned. That is a concept which has, similar to the bouncing function, no obvious biological simile, because there is nothing like a habitat fitness in nature. Nevertheless, the fitness function frame-

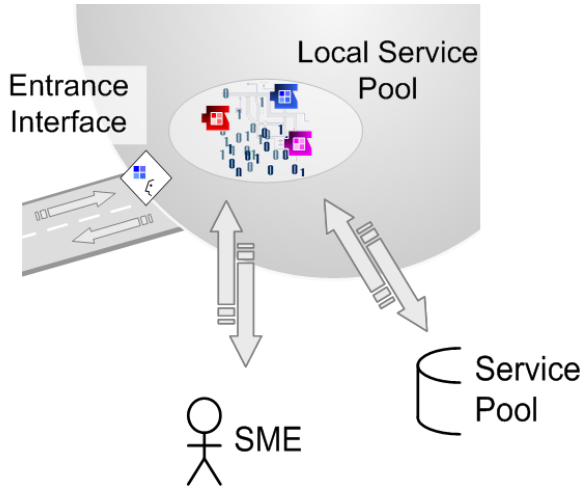

Figure 3.5: User profile based bouncer function usage for migration restrictions

work is flexible enough to build evaluation functions for evaluating the fitness of SMEs if the data for comparison is available. Beside that this will be a long term application of the framework and is only an idea for future research.

3.4 Initially proposed fitness parameters

This section describes some of the fitness parameters and mechanisms which could be taken into account during the optimization process. This is an enhancement of the basic functionality described in section 3.2 DBE Fitness Conception and Interfaces. It has to be stressed here that this list does not claim to be complete or at least sufficient to evaluate fitness for a running system. On the contrary, the listed Parameters should reflect the different anchor points of fitness which were discussed till now and, additionally, show the broad spectrum of applicability of the abstract concept of fitness in the DBE. Although we tried to provide a mathematical framework in this report which is flexible enough to take into account most requirements for future adaptations of fitness attributes, it should be stated here that fitness is always a relative measurement and personalized to the user.

3.4.1 Usage count

The probably simplest but not most marginal fitness value is the usage count of the services. Beside the question of how integrating different versions (usages per version or per product) the usage count gives an accurate feeling of the fitness of a product. Note here, that as many other fitness parameters we have here also the problem of how to relate the usage count of an old service with the fitness of a brand new product.

3.4.2 User feedback – reputation

Beside all fitness parameters that can be used for ranking services, the direct feedback of the user is always one of the most valuable. This feedback starts by giving services credits in terms of a 5 stars model of Amazon, for example. Furthermore, personal impressions can be useful to browse comments of former users of this service to get an impression of the usefulness of this service. Finally, we also can think of a questionnaire with specific questions regarding the service. Nevertheless, there are some issues which should be addressed when using direct user feedback:

- Is the feedback data stored in the history of the service? Has it a life span and is only the latest feedback available?
- Are there security mechanisms to avoid fake entries (manipulation) which lead to a higher ranking of the service?[1]
- Is the evaluation dependent on the personal interest of the user, so that the benefit of the service is low for this SME but very valuable for another? Could that be coupled with a comparison of the user profile?
- Entries like textual notes can be provided for further information but they are not computable, so that the ranking can be influenced. Consequently, should the direct feedback only be used after the ranking for further information?

3.4.3 Growing fitness vector and user profiling

The concept of the growing fitness vector is designed for frequently requested services. It is based on the assumption that services are requested by certain features. Furthermore, we assume the most probable application of such a fitness parameter in scenarios where similar requests are done very frequently and consequently the infrastructure could "learn" which are the most important features of services for a user when he browses for new services.

This growing fitness vector concept can be seen as a B2C as well as a B2B use case. B2C would be a search for hotels by an end user who wants to book a hotel room for vacation. B2B could be a travel agency which wants to search for ideal services for their customers. This example shows that there is sometimes an overlapping in the concepts and there is a potential benefit in concentrating on B2B but also keep an eye on B2C.

For further clarification in the following a short example. The requests in the example are expressed in a minimal version of SBVR as well as the descriptions of the sample services. We also assume a scenario where the DBE is used in a kind of yellow pages for finding real-world services. At this point it is not fully agreed if the main focus of DBE are real-world or software services but the principle of the growing fitness vector should be applicable for both scenarios. Furthermore, we can think of a DBE software service which provides such searching capabilities and optimizations for end users by implementing the growing fitness vector concept.

[1] See also WP32 DBE Regulatory Framework

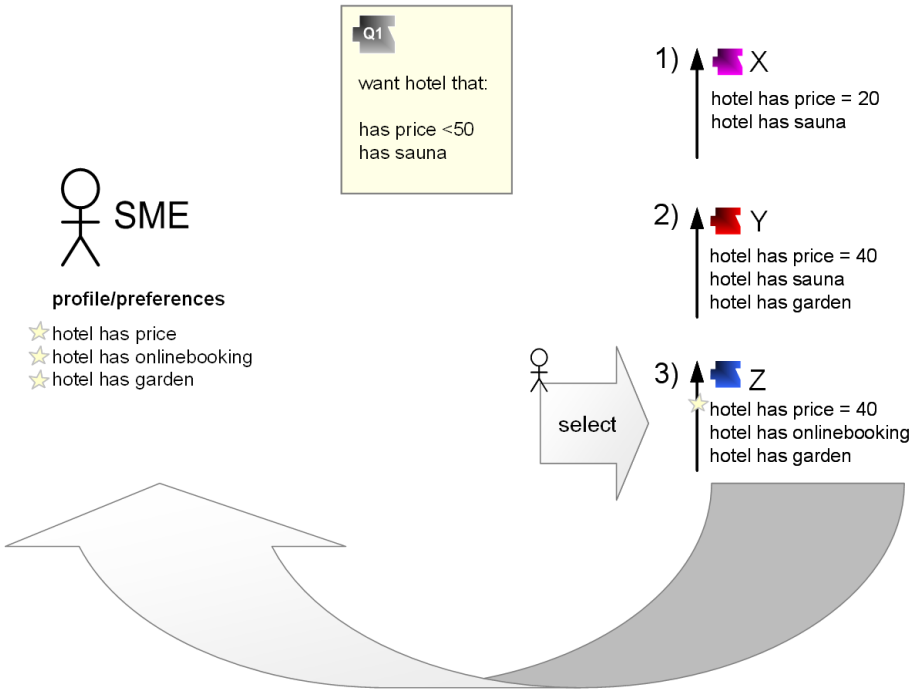

Figure 3.6: Bootstrap for growing fitness vector concept

Starting point for the example is a user request for a hotel (see figure 3.6). The features are described as so-called *fact types*, e.g. *hotel has sauna*. Requested is a hotel room with a price smaller than 50 and where you have access to a sauna. Depending on the requested features the optimization process comes up with a ranked set of services, each with certain features. Although, the optimization process calculates the highest fitness for service X, the user decides to take the third ranked service Z. Now the system checks which features the user requested and which features are provided by the chosen service and assigns a credit to the features which were requested and also found in the chosen service. That is based on the assumption that the user searched for services along certain features and we assume that he also decides along these features.

Beside the recognition of the matching features of request and service, the features of the service are stored for user profiling reasons. This has two main advantages: First, the user profile can be updated with user specific information without a separate interaction with the user and second, features can be recognized which a user is not explicitly requesting but along these the user chooses the services unconsciously. For simplification reasons, the price of the product is only added as a feature and not with a value.

The interfaces to the user profiling are getting clearer but nevertheless, they have to be integrated in the second half of the project. Anyhow, the decision of a user for a service should provide a learning capability for the user profiling, to use the user profile to enhance the capabilities of the optimization ranking for future search runs. We can also think of a part in the user profile which is time limited and therefore, only the

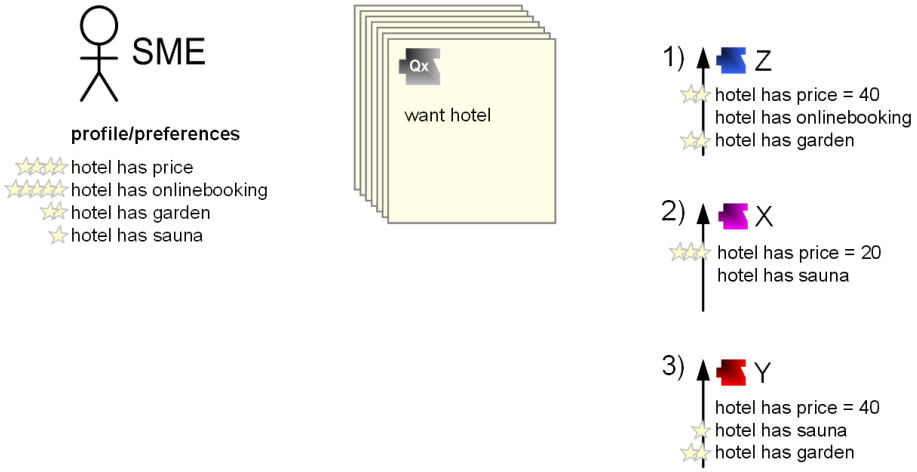

Figure 3.7: Advanced optimization by use of learned user preferences

newest decisions are tracked. Moreover, the parts of the user profile could be taken as a pre-configuration of the fitness function to come up with the best suited search result also before the first interaction of the user with the optimization.

After several searches in which the user has requested and chosen several services the user profile provides a more detailed picture of the SMEs preferences and therefore the fitness function can use this for a ranking specifically adapted to the SME. Moreover, for frequently searched services at least a very simple request should be sufficient for providing practical search results (see figure 3.7).

3.4.4 Interface matching

One vision of the DBE is to automatically compose services (see also WP17 Composer). Consequently, if we plug in a service into another, the output interfaces of the first service have to match the input interfaces of the second service. If we do an optimization according to a request we have to keep in mind that the services we suggest for composition should be composable in terms of matching interfaces. The evaluation of this technical fitness value was also referred to as technical or absolute fitness in the past.

There are two stages of interface matching depending on the language we use. First, the interfaces need to match on a semantic level. That means that the out- and inputs have to contain the same data. Consequently, that is the starting point for the interface matching mechanism. Second, it is possible to compare the interfaces on a syntactic or more technical level. In other words the technical interfaces and data types will be compared. An application of such a comparison and more or less runtime exchange of services within a work flow is nearly impossible without standardization. Nevertheless, that is another starting point for interface fitness.

Not only small software providers are developing interfaces for standardized components. Consequently, if a SME wants to compete with another product on the market it can adapt the interfaces of its own software component to that of the competitor and so

the interfaces are identical and the end user is able to change the service without major modifications of his other components. During the usage of the DBE system this could be another fitness aspect, because it is more secure to use a component where there are more plugins with the same interface.

With the interface matching on a technical level we have identified several constraints and issues of comparing interfaces in the DBE. Some of them are listed here:

- the number of services in the first phase of the project is small and therefore it is difficult to find matching interfaces;
- the diversity of services is high and the first step is always if the service has the requested functionality and then as a second step if the interfaces match.

Although the number of available services and the diversity of services make it practically impossible to compose services automatically, it is important to have a fitness value for matching interfaces. In the first phase of the project the interfaces have a minor role because the services are composed manually but the importance rises with the capabilities of the DBE and the rising number of SMEs and services. For the step-by-step introduction of evolutionary optimization see also section 2.2.

3.4.5 User profiling

Beside the possibilities rising out of the previous described growing fitness vector the user profile could be used in several ways together with the optimization process. According to subsection 3.4.3 the user profiling could give a first starting point where the probable usefulness of a service is evaluated when it migrates to a habitat and so it is mentioned here also in the fitness parameters section. The following points are the result of several discussions and provide a first glance on the expectations and interfaces of optimization and user profiling.

First, the user profiling will include information about the SME. Some of this data will be public and some private. It should includes data which can be typed in by the user or some other authority as well as data which is gathered during interaction with the system. Furthermore, personal interests and preferences will be set which can be used for refinements of the optimization process. According to the mathematical framework in Chapter 4 the data can be used if available and provides an opportunity for the SME to search services more efficiently. The data which is used for the optimization is most probably owned by the SME and private.

Second, the user profiling, especially the dynamically gathered data during usage of the system, is strongly interlocked with fitness data gathering. Therefore, we see the user profiling as a good way for a pre-configuration or personalization of the fitness function. Preferences for searching like: *I don't want to use migrated services during optimization* or *I want only services from companies in my region* could be saved in the user profile and used for the optimization process.

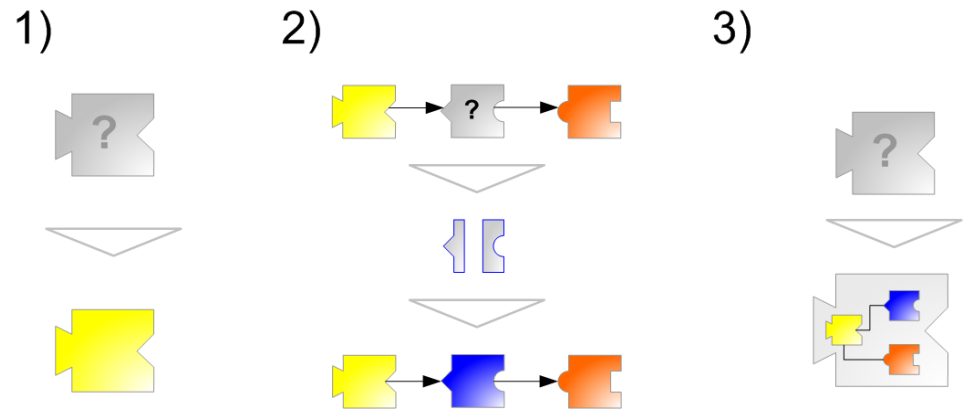

Figure 3.8: Bootstrap process for fitness evaluation

3.5 Bootstrap process for fitness evaluation

Although, fitness concepts are applicable in many cases in the DBE, a step-by-step process is needed to actually implement at least some of these concepts continuously in the DBE infrastructure. Moreover, the recently discovered opportunities from computer science, business and also social science have to be integrated to end with a technically sound system for best benefit for the contractors and users. The following chapter outlines a three-step process for introducing biological inspired ideas practically in the DBE.

3.5.1 Service search

Initially, services are only searched by comparing the Computation Independent Models (CIMs) of the request with the service (see figure 3.8 *1*). That is done by instantiating a genetic algorithm and comparing features. The prototypical implementation of such a scenario by the use of XML based descriptions for the services is documented in Chapter 5. Furthermore, an implementation of such a first optimization problem will be implemented in the EvE service within Task C38. The biology inspired part of this procedure is purely the application of a genetic algorithm adapted to the needs of the DBE.

3.5.2 Service exchange within a fixed workflow

Based on the possibility to find services according to a specific user request, the next step is to exchange services in a fixed workflow. Therefore, after finding similar services to that which should be exchanged the focus lies on the interfaces (figure 3.8 *2*)). First, the interfaces have to match on a semantic CIM level. This first step is not straight forward because the software provider has to model a kind of interface or message which is used to identify docking points of the atomic components. The form of these semantic

descriptions is work in progress and strongly depends on the usage and development of Semantics of Business Vocabulary and Business Rules (SBVR). Beside that also technical interfaces are compared to find services with a minimal integration cost. After proper services are found, the already existing test cases can be used to do integration tests of the new combination. For further details on testing see also papers in WP10 Test Automation. The tests of the new service combinations can also feedback the integration capabilities of certain service combinations. Where all the fitness information should be stored is not clear at the moment but the DSS in WP19 P2P Network Behaviour could seems to be a good candidate solution. Moreover, the detailed interfaces of WP9 Model of Fitness Landscape and WP10 Test Automation is not fully clear until this moment but will be worked out in the next months.

3.5.3 Evolving workflows

Given a large number of services (estimations will result partly from the EvE simulator) the evolution of services without a given workflow is the long term objective of optimization. Based on requirements and expectations from business, optimization and evolution should come up with self-grown service combinations to be then composed by other DBE components like the automated composer. Therefore, we suggest to investigate the possibilities of genetic algorithms for comparing individuals with different structure and workflow. That means that the genetic algorithm creates individuals with different structure - chains, trees, workflows, etc. - and assigns fitness values. Although, that seems to be challenging at least some progress in this field could have great benefit for real evolution inspired optimizations for the DBE.

Chapter 4

A mathematical framework for the fitness function

Il faut d'abord bien savoir le latin. Ensuite, il faut l'oublier.[a]
(Charles de Secondat, baron de La Brède et de Montesquieu, 1689 – 1755)

[a] Initially, Latin should be well known. Then, it should be forgotten.

The definition of fitness for the scope of the DBE project, given the nature of the project itself and the different disciplines involved, calls for an unambiguous language to be used when exchanging ideas between partners. This language acts then as a bridge to carry business ideas to the scientific team and to provide an easy-to-implement output of the scientific research to the computing team.

The language is then a formal way of expressing DBE fitness concepts, rather than being itself a definition of fitness: this is the main reason, but not the only one, why a mathematical framework for defining fitness functions is provided, instead of a mathematically founded definition of a single fitness function. The framework, once established, will be stable and fixed throughout the project, while the fitness function definition will be subject to several changes and adaptations. Together with this main target, several other objectives are met, that started as useful side effects and evolved into functional requirements (analysed in Section 4.1).

In the following sections we will analyze the requirements of a mathematical framework for the fitness function, we will offer an easy-to-understand high-level description

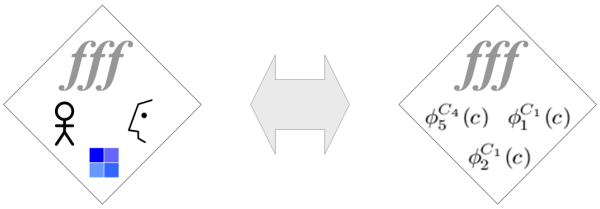

Figure 4.1: Fitness Function logical and mathematical view

of concepts that are useful in satisfying the requirements, and we will finally define the framework currently under study. We will keep always in mind computational aspects of the model, often referring to implementation issues, and propose concrete examples of theoretical ideas. We will also demonstrate how to build fitness functions with the framework by offering examples.

Please note that the previous chapters are required to fully understand the motivations and context of this framework. But note also that terms used more loosely in the rest of the document are used consistently and with a specific mathematical meaning in this section. In particular: a *fitness function* has the solution space of an optimization algorithm as a domain and the set of real numbers as codomain[1]. A *fitness value* is the value obtained applying the fitness function to an element of the solution space. The set formed by taking pairs of elements of the solution space and their corresponding fitness value is the *fitness landscape*: for convenience of representation the solution space is often mapped to a three-dimensional real-valued space. The fitness function is then

$$f : S \to \mathbb{R}, \tag{4.1}$$

where S is the solution space, and a fitness value is $f(x)$, where $x \in S$ is a candidate solution. A fitness landscape requires the definition of an embedding of the solution space in the plane:

$$\phi : S \to \mathbb{R}^2,$$

and is the three-dimensional graph of the following function:

$$g(x, y) = f(\phi^{-1}(x, y)).$$

Please note that here some assumptions on the invertibility of ϕ are made, but this is reasonable in light of the fact that such an embedding should help a human analysis and thus be meaningful (rich in structure) enough to be invertible. Sometimes ϕ^{-1} is not defined for all the points in $\mathbb{R}^2$, in which case a *fitness landscape graph* is a more useful representation: in this there are edges between two points if the related candidate solutions are considered neighbors by the optimization algorithm.

4.1 Requirements

When a user makes a request in the DBE, an optimization process is started to find the best combination of services that can satisfy the user request: the aim of the fitness function is to evaluate the quality of a candidate solution in this optimization process (see Section 3.2). Although this is not the only case where the framework presented in this section can be used, it will be its main application and thus our main focus.

From the problem domain described above we can obtain the following desirable requirements for a mathematical framework for the fitness function:

[1] A set within which the values of a function lie (as opposed to the range, which is the set of values that the function actually takes). Taken from: Eric W. Weisstein. "Codomain." From *MathWorld*–A Wolfram Web Resource. `http://mathworld.wolfram.com/Codomain.html`

- easy adaptability;
- logical behaviour;
- partial evaluation.

These requirements were gathered analysing the project needs as expressed by the business partners and by the scientific partners. They are the rationale for the need of this framework.

4.1.1 Easy adaptability

Investigation on the fitness function requirements revealed that, in the context of the DBE, fitness is a composed and complex concept that should not just take into account a single user request but also other implicitly expressed requirements. This means that just verifying how close a candidate solution is to a user request would reduce the potential capabilities of the evolution that can be achieved through the optimization algorithm.

Mostly for this inner complexity, consensus on the structure of the fitness function has still not been reached between partners but on the contrary it is continuously changing and adapting as more desired functionalities are discovered and implemented.

A requirement for the framework is then that it should allow to easily rewrite parts of the fitness function, thus keeping them separate enough to be independent. It should also allow to easily reorganize the interaction between the parts that defines how they concur in the composition of the final fitness value.

4.1.2 Logical behaviour

When defining fitness-like functions it is evident from the previous requirement that it is useful to split the function into parts that deal with different domains where fitness can have a meaning, or with different aspects of the same domain.

The final fitness value should then be composed from these various values through the language of the framework. It is clear that simple compositions methods (like sums) would not be expressive enough; there should be then the possibility to define reasonable compositions in terms of logical flows. Such a composition flow could for instance define that some part prevails over the others (for instance, technically incompatible compositions of services can get overall low fitness values in our main use case).

The *logical behaviour* requirement refers to this higher level of expressivity in the framework.

4.1.3 Partial evaluation

Once the previous two requirements are defined, we start seeing a fitness function composed of several parts combined in a logical way to contribute to form a final fitness value. It is then reasonable to think that, forcing some constraints on the parts and on

the language for the composition, it should be possible to derive bounds on the final fitness value, and moreover to narrow the bounds by evaluating only some of the parts of the function.

In programming languages (in particular functional ones) partial evaluation means that when having a function that takes n arguments, the first argument can be applied to obtain a new function that takes $n-1$ arguments and that computes the same as the previous function when the first argument is the one applied. This comes from a classical theorem of computability theory and this procedure, related to currying[2], is called partial application [Sch24].

With the *partial evaluation* requirement, we mean that we expect to be able to do a partial application of the function, calculating just the values of some of the parts, to obtain a new fitness function where those parts are already explicit.

It is clear that an immediate advantage of the partial evaluation is the caching of parts of the fitness function in such a way that fitness evaluation is speeded up when called by the optimization process.

But another less obvious advantage is that, by evaluating parts, we can also put stricter bounds on the final fitness value and thus we can reason about the final value also before having it in hand. We could for instance already see that it could not go above a certain minimum threshold, and therefore stop the evaluation and discard the candidate solution.

4.2 Contexts

A problem when reasoning about the fitness function is to model the data in input and to structure it in a way that is:

- abstract enough for grouping and considering all possible kinds of input, but
- with enough structure to make it close to practical issues and to real-world use cases.

From the current architecture of the system [Fer04, Fer05], from reasoning on real-world cases, and from the biological analogy, it is almost natural to think of the fitness function behaviour like a function that evaluates a candidate solution in an environment and gives the fitness value. It is then a function f of this kind:

$$f : E \times S \to \mathbb{R}, \tag{4.2}$$

where E is the environment space (the space of all possible environments), S is the solution space, and $\mathbb{R}$ the real set. You could note that this contrasts with our formal definition of the fitness function in equation 4.1 but this is easily explained by the fact that the environment does not change all throughout the evaluation[3]. In other words,

[2]For another explanation you can see `http://planetmath.org/encyclopedia/Currying.html`

[3]Note that here with environment we mean its model. Although the real-world environment is constantly changing, a single computational optimization process is done in a point in space and time where the environment model is fixed.

using currying as explained in the *partial evaluation* requirement, it would be equivalent to writing function 4.2 as:

$$f_{e \in E} : S \to \mathbb{R}.$$

Where e is the environment in which we are calculating the fitness value of the candidate solutions.

Moreover, the environment is almost naturally hierarchically structured:

- from the biological point of view the hierarchy is biosphere, ecosystem, habitat, individuals;
- from the system point of view the hierarchies can be intuitively obtained by logically organizing UML deployment diagrams and UML class diagrams.

We concluded therefore that our model of the input data should include this hierarchical concept, and we did it through the following definitions.

Definition 4.2.1 (Context - informal)**.** *A* context *is an entity containing data pertaining to one particular domain. Basically a context models logically where some input data is coming from.*

Definition 4.2.2 (Context hierarchies)**.** *A* context *is the* World *context or it contains a reference to another context, called the* parent *context. The* World *context contains no data. For all the other contexts: if a context c_j has context c_i as parent, then c_j inherits all the data of c_i. By recursive inheritance, a context contains all the data of itself and of the recursively parent contexts.*

4.3 Languages

It is worth to do a small excursus on issues related to the choice of the language for the fitness function. When thinking to the optimization algorithm process described above, the fitness function is seen as a computer implementation; this means that it is written in a programming language. But when defining a mathematical framework it is very unlikely that we are going to define a sufficiently rich language to have the same expressivity of programming languages.

It is well known that all general-purpose programming languages have the same strength (same expressiveness); they can in fact do the same work as that of a Turing machine, a formalization of the otherwise informal notion of a mechanical method in logic and mathematics. A language that can perform the work of a Turing machine is said to be Turing-equivalent.

The Turing machine is an abstraction of the steps performed by a computer when executing programs. It is interesting to know the Church-Turing thesis, that states, in one of the versions proposed by Turing [Tur36, Tur48, Tur50]:

> *Every 'function which would naturally be regarded as computable' can be computed by a Turing machine.*

Church's version of the thesis is slightly different, as another abstraction of computation is used (lambda-definable functions and the equivalent class of recursive functions). As this theorem cannot be formally stated, it is impossible to prove it, and for this reason it is often taken for granted and known as the Church-Turing *hypo*thesis. It is also interesting to know that the thesis covers all the computational models subsequently discovered up to now, including quantum computing and probabilistic computing.

A problem with Turing-equivalent languages is that, when reasoning about programs written in such languages, we reach the edge of what is computable. A mathematical framework as we propose for the fitness function, although weak compared to the capabilities of a programming language, would lay down a base upon which some reasoning can be done, obtaining valuable information on instances written in such a language.

We recognize anyway the limits that this would impose, and we agree that it would be desirable to take into account a framework that does not blindly close the opportunity to use programming languages: in fact, the example mathematical model presented in the next section uses both mathematical and computational languages in such a way that it takes advantage of their good properties without being too much affected by the negative ones.

4.4 An example of a mathematical model

An example of a mathematical model that satisfies the requirements described above and that is at the time of writing the best candidate for the framework is presented in this section. It starts with a formalization of the concept of context, giving an intuition on the semantics of the formalization and follows formally defining the language for writing fitness functions together with the semantics for the operators. Finally, some basic theorem are proved that show some promising properties of the framework.

4.4.1 Previous work and contribution

To our knowledge a framework for writing fitness functions has never been investigated as usually the definition of the fitness function, being the objective function, is closely related to the optimization approach used. In the case of extension of discrete-combinatorial optimization problems to their continuous version, for instance, the objective function basically defines also the feasible solutions, that is the points that represent a solution also for the original problem.

In our case, we already underlined the importance of having such a framework. Moreover, the model is not constrained to a particular optimization algorithm: even though genetic algorithms are the main target, as they are the choice for the key evolutionary process in the DBE, the framework can potentially operate for any other optimization process.

The freedom to define from scratch a framework that is modelled after the requirements gathered from the DBE gives us the possibility to look in several directions and to integrate whatever we consider appropriate. During our investigations, we almost by

chance ran into a class of operators that we soon realized adequate for our framework. The example presented in this section integrates such operators and satisfies all of our requirements. Although this framework has not yet been implemented nor used in the real-world, we find it extremely promising: nevertheless, its foundations are constantly questioned and alternative versions or completely different frameworks are taken into account. Until now, this framework has proved to be the one that better satisfies the requirements and required only a few fixes from its original conception.

4.4.2 Formalizing contexts

We define the set of all allowed contexts as a partially ordered set with bottom $(\mathcal{C}, \leq, \bot)$. A context is then an element $c \in \mathcal{C}$ of this set.

Definition 4.4.1 (context set)**.** *We define a context set as a partially ordered set (poset) with bottom* $(\mathcal{C}, \leq, \bot)$.

The semantics of the partial order relation $\leq$ with which the poset is equipped is the following: $c_i \leq c_j$ iff all the data of c_i are also in c_j. We remember that a relation is a partial order if it is reflexive, antisymmetric and transitive. It is trivial to prove that the semantic meaning of the relation satisfies these properties.

The bottom $\bot$ of a poset is its minimal member, that is it is an element $\bot \in \mathcal{C}$ such that for every other element $c \in \mathcal{C}$, $\bot \leq c$. The bottom $\bot$ represents what has been called the World context in definition 4.2.1: it contains no data and it is then the smallest context with respect to the $\leq$ relation.

It is easy to see how this formalization, although simple and without too much structure, is enough to represent mathematically the informal idea of context. Once defined all possible instances of contexts, that is the set $\mathcal{C}$, the relation $\leq$ establishes which set is smaller than another one, representing all the possible context hierarchies (definition 4.2.2): if $c_i \leq c_j$ and $c_i \neq c_j$, then c_i *can* be a parent context of c_j. Seen the other way around, given an element $c \in \mathcal{C}$, if $c \neq \bot$ then c has a parent that is contained in the following set:

Definition 4.4.2 (parent set of a context)**.** *Given the context set* $(\mathcal{C}, \leq, \bot)$*, we define the parent set of* $c \in \mathcal{C}$ *as:*

$$\text{parent}[c] = \{x | x \leq c \text{ and } x \neq c, x \in \mathcal{C}\}.$$

The following theorem follows from this definition.

Theorem 4.4.3 (existence of a parent context)**.** *Let* $(\mathcal{C}, \leq, \bot)$ *be a poset with bottom and* $c \in \mathcal{C}$*. We define the parent set of* c *as* $\text{parent}[c] = \{x | x \leq c \text{ and } x \neq c, x \in \mathcal{C}\}$*. Then* $\text{parent}[c] = \emptyset$ *iff* $c = \bot$*.*

Proof. ($\Leftarrow$) If parent$[c]$ = then, by the definition of parent, there is no $x \neq c$ such that $x \leq c$; but as $\mathcal{C}$ has a bottom, at least $\bot \leq c$; parent$[c]$ can then be empty only if c is the bottom itself.

($\Rightarrow$) If $c = \bot$ then, by the definition of bottom, there is no element smaller than c; more precisely, $x \leq \bot$ iff $x = c$; parent$[c]$ must then be the empty set. □

In this formalization the content of a parent context is automatically inherited from the child context.

Definition 4.4.4 (Context)**.** *Given the poset with bottom of all allowed contexts* $(\mathcal{C}, \leq, \bot)$*, a context is an element* $c \in \mathcal{C}$ *of this set.*

We would like also to be able to classify the instances of contexts by their depth, that represents where the input data is coming from. We define then a function that measures this depth as the distance of a context from the bottom element and we call it rank.

Definition 4.4.5 (Shortest chain)**.** *Let* $c_i, c_j \in \mathcal{C}$*, the context set, with* $c_i \leq c_j$*; the shortest chain from* c_i *to* c_j *is the totally ordered subset* $S \subseteq \mathcal{C}$ *of minimum cardinality that contains both* c_i *and* c_j*. Note that there can be more than one set satisfying this condition, but we usually consider only their cardinality (that is of course the same between all sets), called also the length of the chain.*

Definition 4.4.6 (Rank)**.** *Let* $c \in \mathcal{C}$*, the context set; then rank*(c) *is defined as the length of the shortest chain from* c *to* $\bot$*.*

An equivalence relation $\cong$ is then easy to define:

Definition 4.4.7 ($\cong$)**.** *Let* $c_i, c_j \in \mathcal{C}$*. We say* $c_i \cong c_j$ *iff rank*$(c_i) =$ *rank*(c_j)*.*

We can now partition the set of allowed contexts by this equivalence relation to obtain the required classes. The higher is the rank of a class, the more deep it is and thus the more information it carries. Following the commonly used terminology, we will denote the set resulting from the partition as $\mathcal{C}_{/\cong}$.

To simplify the reasoning, a candidate solution is considered to be a context, in the sense that we can just model the input data on the individual with the same formalism. Note that this means that, reasonably, candidate solutions should be top of chains, that is a context is a candidate solution only if there is no bigger context.

Definition 4.4.8 (Candidate solutions as contexts)**.** $c \in \mathcal{C}$ *is called a candidate solution iff* $\nexists x \in \mathcal{C}. c \leq x$*. In other words, a candidate solution is a maximal element of* $\mathcal{C}$*.*

4.4.3 Formalizing the language

The language $\mathcal{L}$ used to write fitness functions is defined by the following grammar in Backus-Naur Form (BNF), whose semantics will be explained below:

$$\begin{array}{rlll} \mathcal{L} := & \phi_i^C(c) & | & c \in [C], C \in \mathcal{C}_{/\cong}, i \in I. \\ & \mathcal{L} \Uparrow \mathcal{L} & | & \\ & \mathcal{L} \Downarrow \mathcal{L} & | & \\ & (\mathcal{L}) & & \end{array}$$

The first production specifies the terminal symbols of the grammar, that are functions of the kind:

$$\phi_i^C(c) : C \to [-1; 1] \subset \mathbb{R}.$$

Note, from the definition, that such a function is defined to work on an equivalence class $[C]$ of the set $\mathcal{C}_{/\cong}$: it is then able to work only on those contexts that have a specific rank. The index i of the function is just to distinguish them, even tough we can reasonably assume that it is an index on the naturals and thus that the set of functions on a specific context class is enumerable ($I = \mathbb{N}$).

The value returned by these functions carries two meanings: the sign should be positive if the candidate solution is considered to be fit with respect to some aspect, or negative otherwise, while the absolute value should represent how much that candidate is fit (if the sign is positive) or not fit (if the sign is negative).

These functions have absolutely no other constraint beside the ones defined above, so they reasonably are *any computable function* that follows our definition. Note that this means that they can be also Java methods or DBE services. This is how we expect to overcome the limitations exposed in section 4.3: leaving a door open for using programming languages.

The second and third productions define the operators of the grammar. These operators are defined as closed binary operations on the reals:

$$\barwedge\ ,\ \barvee\ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}.$$

The semantics of the operators is the following:

$$x \barwedge y \equiv \begin{cases} x + y & \text{if } x > 0 \wedge y > 0 \\ x + y & \text{if } x \leq 0 \wedge y \leq 0 \\ x & \text{if } x \leq 0 \wedge y > 0 \\ y & \text{if } x > 0 \wedge y \leq 0 \end{cases}$$

$$x \barvee y \equiv \begin{cases} x + y & \text{if } x > 0 \wedge y > 0 \\ x + y & \text{if } x \leq 0 \wedge y \leq 0 \\ y & \text{if } x \leq 0 \wedge y > 0 \\ x & \text{if } x > 0 \wedge y \leq 0 \end{cases}$$

The operators $\barwedge$ and $\barvee$ are read, respectively, "logical-linear and" and "logical-linear or". The intuition to understand the semantics of the operators is that they collect evidence *for* or *against* fitness. They carry two kinds of information during the computation: the sign of the value, that means fit or not fit, and the value itself that gives a measure of the magnitude of the quantity of proof accumulated for or against fitness. With the $\barwedge$ operator we want for instance to force both the operands to be positive for the final value to be positive; in case only one of the two values is positive, only the negative value will be kept; in case both are negative the result will be the summation of the two negative values. The operation performed by the $\barvee$ operator is symmetrically similar.

The operators have been used for edge detection and other forms of simulation of early vision mechanisms [IZ95, Ive93]: their advantage over other methods in the field of computer vision come from the fact that they provide a stricter framework for formalization of vision algorithms (contrarily for instance to the way the well-know Canny's algorithm is defined), a framework that supports both linear accumulation of

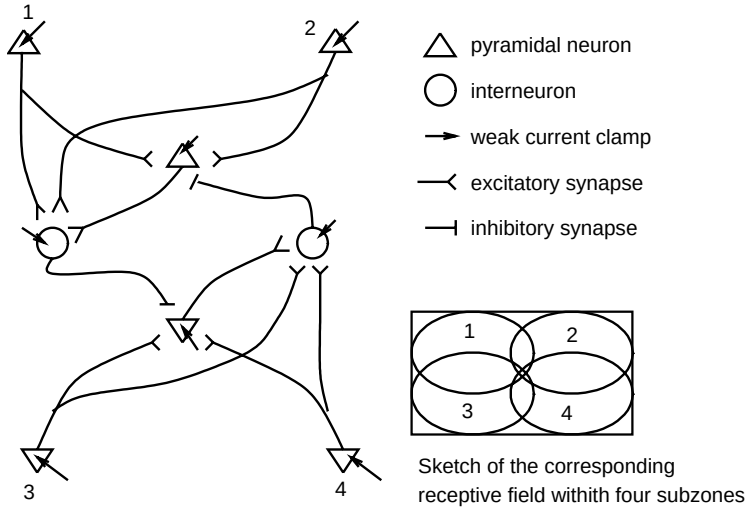

Figure 4.2: A neural circuit implementing a logical/linear operator.

support and non-linear combination of incompatibility. Additionally, there is biological proof that the neuronal cells involved in vision could perform these kinds of operations [BSHIZ03]: we report as an example a sketch of a neural circuit implementing a linear/logical operator in Fig. 4.2.

From our point the operators were interesting for the properties of the expressions of this language: it comprises a boolean algebra, and it is linear on certain subspaces of the entire domain of the vector space $\mathbb{R}^n$, where n is the number of the ϕ functions involved. The adapted proofs of these two properties are omitted here for simplicity but the originals can be found in Iverson's PhD thesis [Ive93]: please note that in the thesis the operators are called combinators and the functions expressed in the language $\mathcal{L}$ are called operators.

The parenthesization defined by the last rule of the grammar aims only at giving the order precedence of the operations; otherwise the $\wedge$ operator has precedence over the $\forall$ operator.

Note that the language $\mathcal{L}$ is all but Turing equivalent: in particular, it does not have recursion mechanisms. We presume that it is anyway sufficiently expressive thanks to the use of computable functions (the grammar terminals, that is the ϕs) as described above.

4.4.4 Adding constants to $\mathcal{L}$

We would like also to add constants to the language $\mathcal{L}$ to give more freedom in defining fitness functions. The resulting language is the following:

$$\begin{array}{rlll} \mathcal{L}' := & \phi_i^C(c) & | & c \in [C], C \in \mathcal{C}_{/\cong}, i \in I. \\ & \mathcal{L}' \curlywedge \mathcal{L}' & | & \\ & \mathcal{L}' \curlyvee \mathcal{L}' & | & \\ & a & | & a \in \mathbb{R} \\ & (\mathcal{L}') & & \end{array}$$

It is semantically equivalent to the previously defined language, as for every constant $a, -1 \leq a \leq 1$ we could substitute a constant function defined in the empty context of this kind:

$$\phi^{\perp}(c) = a$$

Of course, when a is not in the range $[-1; 1]$ we can just make use of the operators to produce the desired a value combining more functions like the one just described above.

As the two languages $\mathcal{L}$ and $\mathcal{L}'$ are equivalent, we will consider one or the other depending on which one is more practical for a specific need. If considerations on one language are not trivially transferable to the other language, we will also provide further details.

4.4.5 Properties of $\mathcal{L}$ and $\mathcal{L}'$

Bounding

We begin defining, by structural induction on $\mathcal{L}$, two operators that calculate the upper and lower bound of an expression.

Definition 4.4.9 (Upper bound)**.** *The structural induction rules to calculate the upper bound are:*

$$\frac{}{1}$$
$$\frac{}{\phi}$$

$$\frac{l_1 + l_2}{l_1 \curlywedge l_2} \qquad \frac{l_1 + l_2}{l_1 \curlyvee l_2}$$

It is easy to verify how these rules calculate the maximum value achievable by a function written in the language $\mathcal{L}$. Similarly, we define the lower bound.

Definition 4.4.10 (Lower bound)**.** *The structural induction rules to calculate the lower bound are:*

$$\frac{}{-1}$$
$$\frac{}{\phi}$$

$$\frac{l_1 + l_2}{l_1 \curlywedge l_2} \qquad \frac{l_1 + l_2}{l_1 \curlyvee l_2}$$

It is easy to prove from definitions 4.4.9 and 4.4.10 the following theorem.

Theorem 4.4.11 (Bounds of a function in $\mathcal{L}$)**.** *Given a function $l \in \mathcal{L}$ with n terminals (occurrences of ϕ functions), the evaluation of the function can only yield values in the closed range $[-n; n] \in \mathbb{R}$.*

Proof. By structural induction:

(base case) $l = \phi$: in this case the function is a single terminal and $n = 1$. From definitions 4.4.9 and 4.4.10 we derive that the lower bound is -1 and the upper bound 1, so the evaluation of l lies in $[-1; 1]$.

(inductive step) l is in one of the following forms:

$l_1 \text{ } \Delta \text{ } l_2$ We assume that l_1 has n_1 terminals and l_2 has n_2 terminals. By inductive hypothesis, the range of l_1 is $[-n_1; n_1]$, while the range of l_2 is $[-n_2; n_2]$. By definition 4.4.10, the lower bound of l is the sum of the lower bound of l_1 and of l_2, and it is then indeed $-(n_1 + n_2)$. Similarly, by definition 4.4.9, the upper bound limit is $n_1 + n_2$. The range of l is then $[-(n_1 + n_2); n_1 + n_2]$. But the number of terminals of l is exactly the sum of the terminals in l_1 and of the ones in l_2: $n = n_1 + n_2$. We have then that the range of l is $[-n; n]$.

$l_1 \text{ } \forall \text{ } l_2$ The same reasoning of $l_1 \text{ } \Delta \text{ } l_2$ applies here.

(l_1) The number of terminals of l is exactly the number of terminals of l_1 and thus the inductive hypothesis is sufficient to prove this case.

□

The results of theorem 4.4.11 could be used to normalize the value obtained by the evaluation of a function in $\mathcal{L}$, so that the possible values of such functions are all distributed in the same range. Usefulness and implications of this normalization are currently being investigated.

Partial evaluation

When having a function l written in $\mathcal{L}$, we can evaluate it for a candidate solution or, if the context is not a candidate solution, we can evaluate some of the terminals and operators where evaluated terminals occur. In this vision, it easier to consider language $\mathcal{L}'$, as an evaluated terminal will give a value that should be represented as a constant.

We formally consider a partial evaluation as an operator that, considering a context, transforms functions in $\mathcal{L}'$ to other functions in $\mathcal{L}'$.

Definition 4.4.12 (Partial Evaluation Operator)**.** *A partial evaluation operator is a function:*

$$\mathfrak{p} : \mathcal{C} \times \mathcal{L}' \to \mathcal{L}'$$

that evaluates all $\phi_i^C(c)$ when c is in the equivalence class $[C]$, and substitutes them with a constant terminal, and it evaluates all operators that have constants as children.

Formally, the semantics of the operator is defined by the following structural induction rules:

$$\frac{\phi_i^C(c)}{\phi_i^C(c)}\ {\scriptstyle c\notin[C]} \qquad \frac{\textit{value of } \phi_i^C(c)}{\phi_i^C(c)}\ {\scriptstyle c\in[C]}$$

$$\frac{c_1 + c_2}{c_1 \uparrow\!\!\!+ c_2} \qquad \frac{c_1 + c_2}{c_1 \forall c_2}$$

$$\frac{l_1 \uparrow\!\!\!+ l_2}{l_1 \uparrow\!\!\!+ l_2} \qquad \frac{l_1 \forall l_2}{l_1 \forall l_2}$$

Definition 4.4.13 (Partial Evaluation Step)**.** *As it is useful to do all the substitutions at once, we will consider a partial evaluation step as the fixed point of* $\mathfrak{p}_c$*. We will write:*

$$l_1 \xrightarrow{c}_{\mathfrak{p}} l_2$$

To consider the result of all the substitutions done by the operator $\mathfrak{p}$ *when evaluating in the context* c*.*

Of course, when the context is a candidate solution, a partial evaluation is supposed to give a final constant value.

4.4.6 Experimental implementation

To experiment with the language $\mathcal{L}$ a prototypical implementation has been done in OCaml [Ler95, RV97].

We will provide here the lines of codes worthy of attention. First of all, the definitions:

```
module Context = struct
  type t = ( string, string ) Hashtbl.t

  let make () =
    Hashtbl.create 50

  let set c key v =
    Hashtbl.add c key v

  let get c s =
    Hashtbl.find c s

  let is_world c =
    Hashtbl.length c = 0
end

type context = Context.t
```

```
type phi = context -> float option

type operator =
  | Constant of float
  | Phi of phi
  | Llor of operator * operator
  | Lland of operator * operator
```

A context is just seen as a bag of data (a hashtable), where the $\leq$ relation is the subset relation. A function ϕ checks if context has the required data, in which case it returns SOME result, otherwise it returns a NONE result.

The function that computes the bounds of a function in $\mathcal{L}'$ is straightforward from the inductive definition:

```
let rec bounds = function
  | Constant c ->
    ( c, c )
  | Phi f ->
    ( -1.0, 1.0 )
  | Llor ( op1, op2 ) ->
    let ( min1, max1 ) = bounds op1 in
    let ( min2, max2 ) = bounds op2 in
    (
      ( if min1 *. min2 >= 0.0
          then min1 +. min2
          else if min1 >= 0.0
                 then min1
                 else min2 ),
      ( if max1 *. max2 >= 0.0
          then max1 +. max2
          else if max1 >= 0.0
                 then max1
                 else max2 )
    )
  | Lland ( op1, op2 ) ->
    let ( min1, max1 ) = bounds op1 in
    let ( min2, max2 ) = bounds op2 in
    (
      ( if min1 *. min2 >= 0.0
          then min1 +. min2
          else if min1 <= 0.0
                 then min1
                 else min2 ),
      ( if max1 *. max2 >= 0.0
```

```
          then max1 +. max2
          else if max1 <= 0.0
                 then max1
                 else max2 )
    )
```

As it is straightforward the function to do the partial evaluation:

```
let pevaluate context op =
  let rec pevaluate = function
    | Constant v ->
      Constant v
    | Phi f ->
      ( match f context with
          | None ->
            Phi f
          | Some v ->
            Constant v )
    | Llor ( op1, op2 ) ->
      ( match ( pevaluate op1, pevaluate op2 ) with
        | ( Constant v1, Constant v2 ) ->
            if v1 *. v2 > 0.0
              then Constant ( v1 +. v2 )
              else if v1 >= v2
                then Constant v1
                else Constant v2
      | ( op1, op2 ) ->
            Llor ( op1, op2 )
      )
    | Lland ( op1,  op2 ) ->
      (
      match ( pevaluate op1, pevaluate op2 ) with
      | ( Constant v1, Constant v2 ) ->
            if v1 *. v2 > 0.0
              then Constant ( v1 +. v2 )
              else if v1 >= v2
                then Constant v2
                else Constant v1
      | ( op1, op2 ) ->
          Lland ( op1, op2 )
      )
  in
  pevaluate op
```

A simple ϕ function could then look like:

```
Phi ( function c ->
    try
      let failures = float_of_string
        ( Context.get c "SMEProfile.UsageData.Failures" ) in
      let calls = float_of_string
       ( Context.get c "SMEProfile.UsageData.TotalCalls" ) in
      Some ( failures /. calls )
    with
      Not_found -> None
)
```

4.5 Future of the mathematical framework

In the next phase of the project the mathematical framework proposed here will be studied more thoroughly from the theoretical point of view and will then be implemented into the existing simulator and ported to the EvE service version of the simulator. Probably in a real-world implementation, some parts will be revised forcing a new analysis of the theory behind for consistency. Some additional theorems on the properties of the operators should be formally proved and written down.

Most of the future work will probably be on the ϕ functions, maybe defining a basic set of them, or maybe giving primitives for basic operations. For instance we could define a set of ϕs to work on the abstract syntax description of requests and services.

Chapter 5

Simulator

> *As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.*
> (Albert Einstein, 1879 – 1955)

This first version of the fitness landscape simulator is to test and explore the possibility of optimising a service chain using evolutionary computing and genetic algorithms. The main focus of the work on the simulator was to see if it is a feasible approach to use a GA for the service chain optimisation problem. Therefore a first basic fitness function for evaluating the fitness is presented. The fitness function calculates a *relative* fitness, meaning that the fitness of a service chain is evaluated to a given service request (see Chapter 5.3 for details).

The decision for using a GA instead of other optimisation techniques is based on the work done by UBham, which is documented accordingly in their deliverable D8.1 [RM05]. Using UBhams work as basis, the simulator is also used to investigate the parameters of the GA. Even more, some research into performance issues regarding scalability has been made. Therefore, a brief report on results of this testing is also given (see Chapter 5.4).

The simulator currently does not make use of the DBE infrastructure what is mainly due to the fact that a stable release of the system has not been available. However, as the DBE computing infrastructure itself the simulator was implemented using the Java [GJSB00] programming language. This will allow an easy adaptation to the DBE framework and infrastructural services.

As mentioned above, the simulator optimises service chains using methodologies and algorithms from evolutionary computing. As WP9 focuses on fitness rather than on genetic algorithms, and the work on and with the simulator focuses on the *usage* and not on the *development* of a GA. Therefore, although the implementation of a genetic algorithm would be a straightforward engineering task, we decided to use an existing, well-proofed and documented evolutionary computing framework. The framework used by the simulator is the ECJ 12 [Luk02] Evolutionary Computing (EC) research system. Since this framework is also implemented in Java, it therefore could be adapted and integrated to fulfill the simulator's needs.

5.1 Service definition

The simulator uses a simplified service definition. A simulator-service has just two parameters, *type* and *value*. Currently, type is a character between A and K. The associated value is a integer between 1 and 10. To give a concrete example, service type could represent "Hotel", service value would then represent the category (3 star, 4 star, ...) of the hotel. In the terms of genetics, this single service represents a *gene*. For identification, each service needs a unique service id.

In order to build service chains, a *service pool* (a pool of services), as an equivalent to the gene pool, is needed. This service pool, which is therefore the source for the solutions build by the GA, can be pre-defined and imported using an XML file with the proper structure. If no pool is available, it is possible to create a new one with randomly[1] generated services. In terms of the DBE architecture, this pool can be seen as the equivalent (or simple representation) of the DBE Knowledge Base (see [Fer05]).

A service chain represents an *individual*. As for the service definition, we use a simplified service chain definition. First, a service chain is a *list* of services. Also, the simulator does not note the "glue" between services of a chain. It is defined that the output of one service in a chain is handled on to the next, acting as an input to its successor. Hence, the fitness evaluation in our simulation focuses just on *one* part of a service definition. If we stress the hotel example again, the fitness evaluation contains the *descriptive* part of a service (CIM), and not the matching of the *technical* interfaces. Beside this, the *position* of the services of a chain is of importance. A service chain "A1→B5" is not equal to a chain "B5→A1", although the *services* might be the same.

From the coding point of view, a service chain is implement as a integer vector. Each component of the vector holds an integer value, the service-id, which references the service. Therefore, a service-chain is not a composition of services but a composition of *references* to services.

The current version of the simulator supports a configurable but fixed length of service chain (genome). That is, the length of a service chain can be defined in the configuration of the simulator, but – for the period of a given simulation – it is fixed. If a request has the length of four, meaning a service chain is composed of four services, the GA will produce responses with a length of four.

5.2 Simulation

For the matter of handling, the simulator is coupled with a Java-Swing based Graphical User Interface (GUI) (see Fig. 5.1). In this screen-shot, a genome/service chain length of four is being used. The number of items will automatically change when modifying the corresponding configuration parameter. As can be seen in the figure, a request is formulated by selecting the service type and value for each component (service) of a service chain. In addition, the simulator provides a weighting factor – *severity* – via a slide for each service. This severity slide allows to express the importance and exactness

[1] As the Java Math.random() function is used, this is actually a *pseudo*-random generation

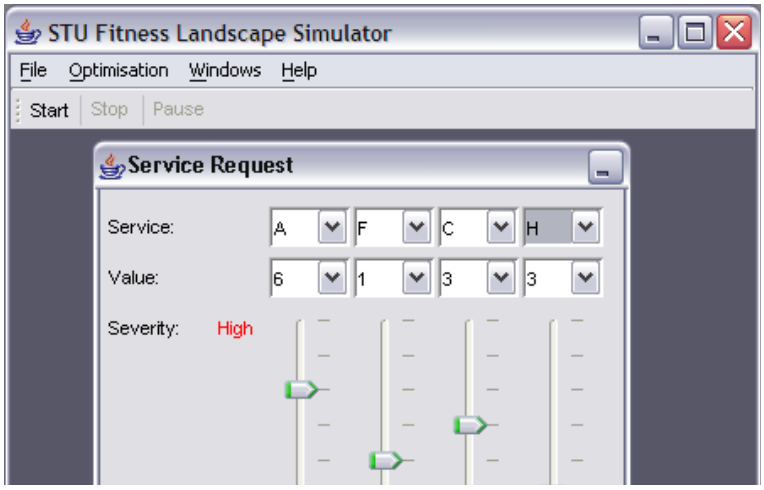

Figure 5.1: Fitness Landscape Simulator

for evaluating the fitness, directly impacting the fitness calculation for a response from the GA. If severity of a service is set to *high*, a high fitness value is only produced if service type is the same as in the request and the value for a service is also an exact match. Severity *low* means that a value that is close to the request also produces a high fitness given the same service type. To give an (again hotel) example: value=8 means a 4-star rated hotel. If severity is set to "high", than the user wants *exactly* a 4-star rated hotel. If severity is set to "low", the user *prefers* a 4-star, but "near" solutions (3-star, 5-star) are also welcome. The precise calculation of fitness and the influence of the severity is explained in detail in Chapter 5.3, Fitness Function. After formulating a request, a service chain can be searched by clicking the start button. While the simulation is running, several windows are being used to visualise the optimisation process (see Fig. 5.2).

The *Fitness Evolution* graph window prints the currently highest fitness value if this run, and the history of the run. Here, the fitness *trend* can be seen and tracked.

The second graph, *Generation History*, sketches the highest fitness value of each generation. As the effects of changes in the GA parameters can be well seen in this graph, the data used for plotting this information is the source for analysing the effort of parameter changes.

The third window, *Top 5 Individuals*, shows the by far fittest individuals found in this run. Also, the associated fitness values of each individual is presented.

All data presented in the graphs are written to appropriate log files to allow post processing and in-detail analysis of the data.

5.3 Fitness Function

The fitness function presented in this section was implemented before the idea of the mathematical fitness function *framework* (see Chapter 4) was conceived, and for this

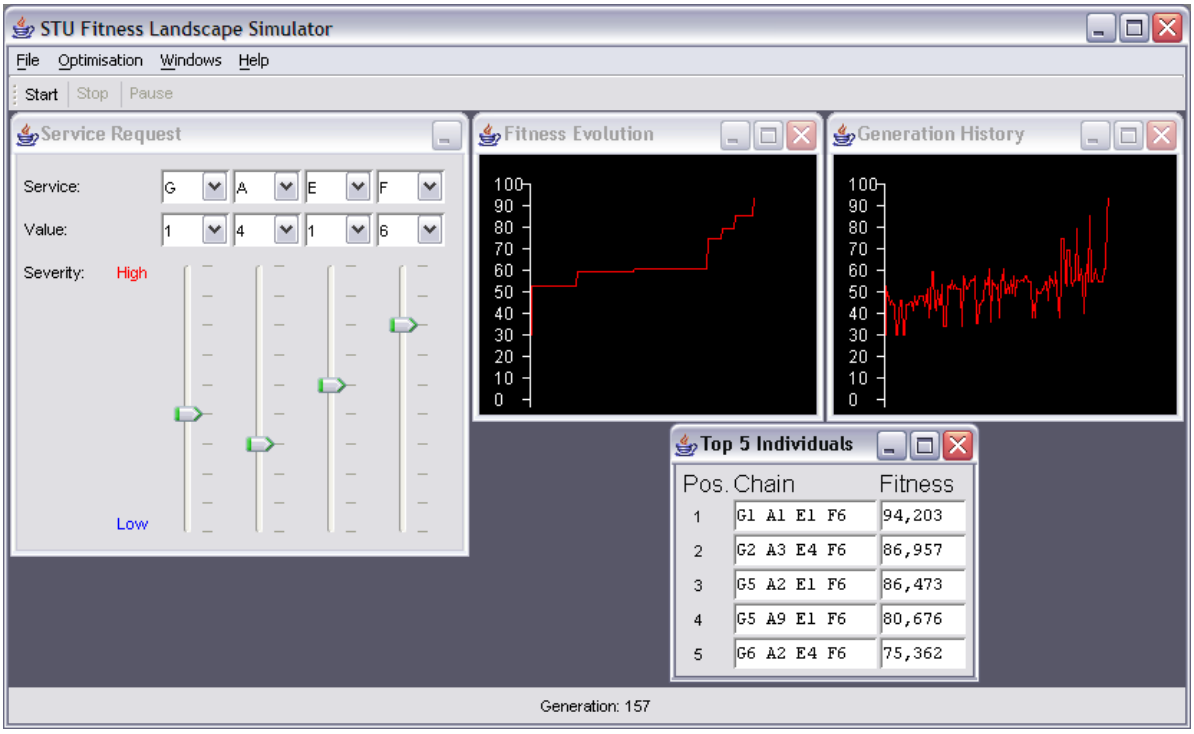

Figure 5.2: Simulation output

reason this simulator does not *implement* this framework. In fact, as said before, the aim of the simulator was to test the optimisation process and thus the mathematical framework implementation was out of scope. Nevertheless, the fitness function presented below can be expressed through the framework as we will show at the end of this section after the introduction of the fitness function.

The function that is used to evaluate the fitness is based on geometry. Like a species "sits" on a specific biological niche in an ecosystem, we see a service chain "sitting" on a specific point in a multi-dimensional space, where its exact position is fixed by its features and parameters. Service chains that are geographically *near* in such a space have similar features and parameters. Fig. 5.3 shows an example for a service chain in a 2-dimensional space, where its position is defined by its 2 services (Service 1, Service 2). Here, the effect of "severity" is also visualised: a low severity just "broadens" the niche of a service chain, making more far solutions also fit.

So, the fitness is calculated by measuring the *distance* between the point that is represented by the service request and the point represented by the currently evaluated service chain. The following formula shows the details of the function:

$$Alphabet\ A = \{1, .., 10\}\,, \quad \#A = 10$$

$$\vec{x}, \vec{y} \in A^d \quad d..dimension$$

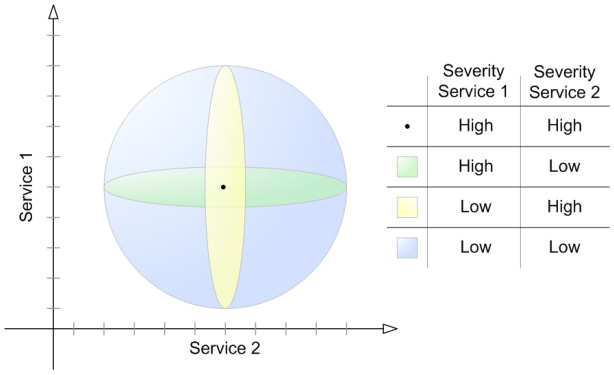

Figure 5.3: Severity

$$\vec{\omega} \in (0,1]^d \quad \vec{\omega} = (\omega_1, .., \omega_d) \quad \omega_i \in (0,1]$$

$$Distance\ \delta\left(\vec{x}, \vec{y}, \vec{\omega}\right) = \left(\frac{1}{(|A|-1) * \sum_{i=1}^{d} \omega_i}\right) * \sum_{i=1}^{d} \omega_i \left|x_i - y_i\right|$$

The Alphabet A defines that each service type has 10 possible values between 1 and 10. Vector x represents the values of the service request, vector y the current service chain. The dimension is equal to length of service chain.

The severity is represented with vector ω. ω can have a value between zero and one (excluding zero). A *high* severity adjustment in the user interface means $\omega = 1$.

The distance δ is calculated as the sum of all absolute values of x minus y weighted with ω. The denominator is a normalizing constant so that the final value is between zero and one.

This fitness function can of course also be expressed using the framework described in Chapter 4, as anticipated in the beginning. Note that for each i-th element in the vectors x and y we check if they are the same kind of service (in case they are not, the maximum distance $|A|$ is assigned) and calculate their difference $|x_i - y_i|$. We can then write the following ϕ functions:

$$\textsc{SameService}_i(y) = \begin{cases} \omega_i & \text{if } x_i = y_i \\ -\omega_i & \text{otherwise} \end{cases}$$

$$\textsc{Diff}_i(y) = \omega_i |x_i - y_i|$$

As these ϕ functions do not use any data but the $\vec{x}$ that is the candidate solution and $\vec{\omega}, \vec{y}$ that are the user request, they cannot do any kind of partial evaluation (see section 4.1.3). They can be combined to form the final fitness function in the following way:

$$\begin{array}{ll} (\textsc{SameService}_1(y) \mathbin{\barwedge} \textsc{Diff}_1(y)) & \forall \\ (\textsc{SameService}_2(y) \mathbin{\barwedge} \textsc{Diff}_2(y)) & \forall \\ \quad \ldots & \forall \\ (\textsc{SameService}_d(y) \mathbin{\barwedge} \textsc{Diff}_d(y)) & \end{array}$$

The normalization, if needed, can then be achieved automatically, using theorem 4.4.11: this kind of normalization slightly differs from the one proposed above as it is absolute, and not relative to the weights given by the user request, but is still effective for the purposes of the optimisation.

5.4 Tests

In order to investigate the nature of a genetic algorithm implemented in Java and to make assumptions on behaviour of such a system in a production environment, several tests have been performed. The tests performed were focusing on the following two main aspects:

1. Influence of GA parameters on finding a solution
2. Influence of size of search space on finding a solution

The next sections present the outcomes of the tests.

5.4.1 GA parameters

The simulator uses *steady state*[2] Genetic Algorithm (GA). Here, we investigated the following parameters:

- Selection Method;
- Crossover Method;
- Crossover Probability;
- Mutation Probability.

The parameter values have been set in order to have the least number of generations to obtain a most fit solution. To focus on the results rather than the details of implementation, we do not give a detailed overview and explanation on each parameter and possible values. Nevertheless, most parameters are well know from the literature on genetic programming (e.g. [Koz92, Gol89, BP02]). Their exact definition and meaning within the scope of the ECJ Framework can be seen in the ECJ documentation available at the product's home location [Luk02].

The parameter tests have been performed with a service pool with size $n = 10.000$, and a service chain length of four. The termination conditions are having an individual with $fitness > 0.94$ or exceeding 1.500 generations.

Selection Method: Best results have been achieved using "Tournament selection" which works like this: first, *size* individuals are chosen at random from the population. Then of those individuals, the one with the best fitness is selected (taken from [Luk02]). Even more, using the default $size = 7$ was best (see also [Koz92] for details).

[2]For details on Genetic Algorithms and parameters, see inter alia [RM05, BFKN98, BP02]

Crossover Method: Best results with "One Point Crossover". This means, that one crossover point is generated at random. The probability of creating a crossover point is defined by the next parameter.

Crossover Probability: Giving a high Crossover probability (near to 1) worked sufficiently well.

Mutation Probability: Although the results where not as clear as for the other parameters, we saw a good trend when using mutation probability between 0.05 and 0.17. Lower values resulted in needing significant more generations to get a result, higher values tend to need more. As we got no really clear and precise result here, we used a mutation probability of 0.05 for our further investigation, which produced satisfactory results.

5.4.2 Search space

The second test focuses on the influence of the size of the service pool (or search space) on the time used to find an optimal solution. The notion is to be able to make forecasts on the run-time behaviour of the system with respect to the amount of services available.

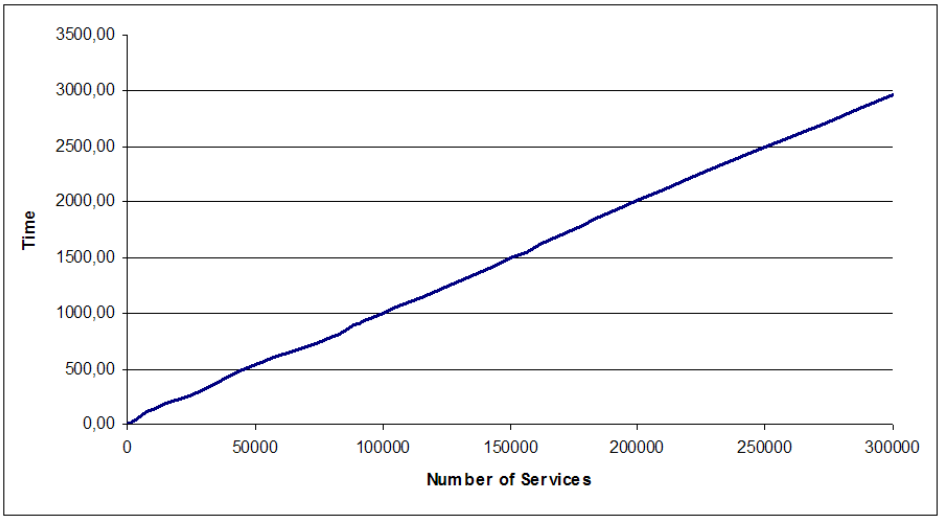

Figure 5.4: Search space test

As before, we used a service chain length of four. Then, different service pools have been generated randomly. In detail, service pools with the following number of service have been created: 10^2, 10^3, $5 * 10^3$, 10^4, $3 * 10^4$, $5 * 10^4$, $8 * 10^4$, 10^5, $1.5 * 10^5$, $2 * 10^5$, $2.5 * 10^5$, $3 * 10^5$. Even more, three pools per number of services have been created and used in order to flatten statistical outlier. Fig. 5.4 shows the result of the search space test. As can be seen, the system shows an almost linear behaviour.

This timing experiments have been made in a simulated, simplified environment. But the results can be extrapolated to the forthcoming EvE service implementation.

Although we currently do not know the amount of services that will be available in the DBE at a given time in future, we can make well-founded predictions on the time needed to find an optimum solution once we have a first system up and running.

Chapter 6

Conclusions and outlook

> *The future, according to some scientists, will be exactly like the past, only far more expensive.*
> (John Sladek, 1937 – 2000)

In this report, scientific and computational aspects of the concept of fitness for the utilization of the biological simile in the DBE project are covered. As the role of Salzburg University of Applied Sciences (STU) is to interface between scientifically focused and computationally focused partners in the project, the present work intends to translate concepts from science to computationally feasible approaches.

6.1 Capabilities of the fitness model

Fitness in nature and in the DBE exists as a relative concept hence fitness landscapes are graphical representations of relative fitness values. The biologically inspired concept of fitness can be applied in many places in the DBE. However, a complete one-to-one mapping of all known aspects of biological fitness to software seems to be not feasible at any given time. A reasonable starting point to put it operational is the request-depending optimization of software services by a GA. Real-world services and software services can be treated similarly when wrapping the former with a software-layer capable of mediating the services. Several aspects regarding optimization can then be treated similarly, there are, however, aspects that differ.

A myriad of fitness parameters could be suggested; we have picked a small subset for proof-of-concept as the final fitness assessment will be user-dependent and environment-dependent. With increase in time, an increasing amount of fitness data is expected thus an ever-improving utility of that data is probable. However, the issue of getting rid of obsolete data has to be solved in the near future.

On the long run, the evolution of services itself could be represented by principal sub-services that continuously are integrated into self-improving fully fledged services by application of the concepts detailed in this work.

As the FL deals with evolution of services to fit a specific request, the EvE deals with the ongoing evolution of the whole ecosystem. The EvE is the scientific abstraction that

allows to deal with such evolution within and between the Habitats. In the same way that the FL-Simulator is proving useful for investigating the FL, STU saw the necessity to go into further details of EvE by simulating its behaviour before implementing it in the runtime environment. The EvE-Simulator, which uses partly the already implemented DBE components and partly simulated components recreates parts of the EvE in a virtual environment and simulates its behaviour over time. The EvE-Simulator should be an extension of the FL-Simulator in the sense that the FL-Simulator itself is used internally to simulate the optimization of services and the EvE-Simulator is used to simulate the behaviour of the evolutionary process in the distributed DBE.

Following some possible capabilities of an EvE simulation are listed. Some of them will be implemented within the task *Simulator Evolutionary Environment (EvE)* of WP9:

- estimation of the critical mass (number of services, number of SMEs);
- performance testing and behaviour of different optimization algorithms;
- tests of the dynamic behaviour of the system for unexpected problems;
- usage as a marketing instrument for attracting new regions;
- organization of the level of distribution (centralized against fully distributed);
- exploration of emerging unknown properties of the system;
- etc.

The behaviour of the SMEs will be simulated or influenced by one or more users similarly to a *Monopoly game* to test for example self-learning capabilities of the fitness functions. Therefore we intend to use existing GA-based simulations of economic systems (see [Dav99]) and adapt them to the needs of an EvE simulation in the DBE.

When working on the definition of the EvE architecture together with ICL, a number of open issues and questions where documented. Although some of these questions are already answered, most of these questions are going to be answered during the definition and design of the EvE architecture. Nevertheless, we added all of them in Appendix A of this document for clarification and as a common base for discussion.

6.2 Capabilities of the mathematical model

A context-based fitness concept is presented in this work in the form of a Mathematical Fitness Function Framework (MFFF). It is in the form of a formal language for expression of fitness requirements and is intended as a common language to speak about fitness without misunderstandings. The computational implementation of this framework is expected to be straight-forward, the mapping to business stakeholders' needs is currently under development. Besides studying further theorems and properties of the model, the implementation of this framework in the EvE service is on top of the agenda as this should also be the starting point for a wide-ranging utilization of the model in the Open

Source Software (OSS) community. From this work it will be seen whether or not the mathematical framework has to be adapted for the situation of distributed environments.

It is also intended to study further how a distributed environment affects the fitness calculation, e.g. for clustering or determination of global fitness values, and to explore where else the mathematical framework is useful, e.g. regarding Quality of Service (QoS) or even inside a cluster pool. The idea of using the mathematical framework for evaluations related to the cluster is an expansion of the idea of partial evaluation and implies an evaluation of fitness of services for a cluster and not against a user request.

The vision here is to have the ϕ-function of the framework implemented as different DBE-services enabling auto-evolution of fitness functions. In the case that services can be automatically created in the future based on formal description, fitness functions can also be generated *from the scratch* like 'growing from nothing'. The fitness function could understand by itself what is to be considered *fit*. In addition, the related service description communicates that information transparently to the user(s).

6.3 Capabilities of the simulator

The fitness simulator discussed here is intended as demonstrator for applicability of the scientific vision to internal and external stakeholders of the DBE project, specifically from the business domain. It helps to benchmark the 'academic' optimization with real business needs by making visible add value potential for business partners.

It is intended to integrate the knowledge derived from the simulator in the forthcoming EvE service implementation and to widen the concept of EvE by taking input from the distributed intelligence to prove the global benefit for the whole system.

Then it can be used to *pre-tune* user requests in a way that the service 'knows' the client's (or clients') expectations regarding information, components, and services. It might also be possible to extend the concepts to application domains such as network management for studying visibility, latency and this like in a distributed service environment. On the long run, the exploration of EvE functionality as 'operating system functionality' is envisaged.

Bibliography

[AJL+02] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. *Molecular Biology of the Cell*. Garland Publishing, Inc., New York, NY, USA, fourth edition, 2002.

[Ave03] John Avery. *Information Theory and Evolution*. World Scientific Publishing, Singapore, 2003.

[Bak96] Per Bak. *How nature works*. Copernicus, New York, 1996.

[BD04] Gerard Briscoe and Paolo Dini. Evolutionary Environment Discussion Paper. Internal DBE Report, 2004.

[BFKN98] Wolfgang Banzhaf, Frank D. Francone, Robert E. Keller, and Peter Nordin. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., dpunkt.verlag, 1998.

[BHT96] Mike Begon, John Harper, and Colin Townsend. *Ecology: Individuals, Populations and Communities. Third Edition*. Blackwell Publishing, 1996.

[BP02] William B.Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer, Berlin Heidelberg New York, 2002.

[Bri04a] Gerard Briscoe. D6.1 - Self-organisation in Multi-Agent Systems. Version 1, July 2004.

[Bri04b] Gerard Briscoe. Science2Computing document. Internal DBE Report, 2004.

[BSHIZ03] Ohad Ben-Shahar, Patrick S. Huggins, Tomas Izo, and Steven W. Zucker. Cortical connections and early visual function: intra- and inter-columnar processing. *Journal of Physiology-Paris*, 97:191–208, March 2003.

[Dav99] Herbert David. *Adaptive Learning by Genetic Algorithms, Second, Revised and Enlarged Edition*. Springer, Berlin Heidelberg New York, 1999.

[DBE03] DBE Consortium. Digital Business Ecosystem, Annex I - Description of Work. Sixth Framework Programme, Priority 2.3.1.9., Networked Business and Governments, Proposal/Contract No. IST2002-507953, 2003.

[Die97] Reinhard Diestel. *Graph Theory.* Springer-Verlag, New York, 1997.

[Din04] Paolo Dini. D18.1 - Report on DBE-Specific Use Cases. Version 4, December 2004.

[Dow05] Jim Dowling. D23.1 - DBE Fitness Landscape. Version 1.0, January 2005.

[Fer04] Pierfranco Ferronato. D21.1 - DBE Architecture Requirements. Version 2.6, November 2004.

[Fer05] Pierfranco Ferronato. D21.2 - DBE Architecture Scoping Document. release A, January 2005.

[Gav04] Sergey Gavrilets. *Fitness Landscapes and the Origin of Species.* Princeton University Press, Princeton and Oxford, 2004.

[GJSB00] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification 2nd Ed.* Addison-Wesley, Boston, 2000.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison Wesley, 1989.

[HKM+04] Thomas Heistracher, Thomas Kurz, Claudius Masuch, Pierfranco Ferronato, Miguel Vidal, Angelo Corallo, Gerard Briscoe, and Paolo Dini. Pervasive Service Architecture for a Digital Business Ecosystem. *CoRR*, cs.CE/0408047:71–80, 2004.

[Ive93] Lee A. Iverson. *Toward discrete geometric models for early vision.* PhD thesis, McGill University, Montreal, 1993.

[IZ95] Lee A. Iverson and Steven W. Zucker. Logical/Linear Operators for Image Curves. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(10):982–996, 1995.

[Kau00] Stuart A. Kauffman. *Investigations.* Oxford University Press, New York, 2000.

[Koz92] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, Cambridge, MA, USA, 1992.

[Ler95] Xavier Leroy. Le système Caml Special Light: modules et compilation efficace en Caml. Rapport de recherche 2721, INRIA, nov 1995.

[Luk02] Sean Luke. *ECJ: A Java-based evolutionary computation and genetic programming system.* Available at `http://cs.gmu.edu/~eclab/projects/ecj/`, 2002.

[Mar05] Giulio Marcon. A Mathematical Framework for the Fitness Function. Internal DBE Report, January 2005.

[Mas05] Claudius Masuch. Service manifest conceptional model (internal dbe report). Version 0.9, 2005.

[MS02] Lynn Margulis and Dorion Sagan. *Acquiring genomes: a theory of the origins of species*. Basic Books, 1st edition, 2002.

[Nac02] Francesco Nachira. Toward a network of digital business ecosystems fostering the local development. Discussion paper, September 2002.

[PGT04a] Maria Petrou, Konstantinos Giannoutakis, and Duminda Thilakawardana. D12.1 - Model adopted. Version 2, July 2004.

[PGT04b] Maria Petrou, Konstantinos Giannoutakis, and Duminda Thilakawardana. D12.2 - Optimisation approaches and software optimising the cost function in a static space. Version 2, December 2004.

[RH04] Jonathan E. Rowe and Dzena Hidovic. An evolution strategy using a continuous version of the gray-code neighbourhood distribution. In Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund K. Burke, Paul J. Darwen, Dipankar Dasgupta, Dario Floreano, James A. Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andrew M. Tyrrell, editors, *GECCO (1)*, volume 3102 of *Lecture Notes in Computer Science*, pages 725–736. Springer, 2004.

[RM05] Jonathan E. Rowe and Boris Mitavskiy. D8.1 - Report on Evolution of High-Level Software Components. Version 0.9, April 2005.

[Row05] Jonathan E. Rowe. Population dynamics of genetic algorithms. In *Foundations of Learning Classifier Systems*. Springer, 2005. To appear.

[RV97] Didier Rémy and Jérôme Vouillon. Objective ML: A simple object-oriented extension of ML. In *Proceedings of the 24th ACM Conference on Principles of Programming Languages*, pages 40–53, January 1997.

[Sch24] Moses Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316, 1924.

[Tur36] Alan M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.

[Tur48] Alan M. Turing. Intelligent machinery. In *Machine Intelligence*, volume 5, chapter 1, pages 3–23. Edinburgh University Press, Edinburgh, UK, 1948.

[Tur50] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[Wri32] Sewall Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In D. F. Jones, editor, *Proceddings of the Sixth International Congress on Genetics*, volume 1 of *Lecture Notes in Computer Science*, pages 356–366, 1932.

List of Abbreviations

B2B	Business to Business
B2C	Business to Consumer
BML	Business Modelling Language
BNF	Backus-Naur Form
BPEL	Business Process Execution Language
CIM	Computation Independent Model
DBE	Digital Business Ecosystem
DNA	Deoxyribonucleic Acid
DSS	Distributed Storage System
EC	Evolutionary Computing
EvE	Evolutionary Environment
ExE	Execution Environment
FL	Fitness Landscape
FFF	Fitness Function Framework
FZI	Forschungszentrum für Informatik, Karlsruhe
GUI	Graphical User Interface
GA	Genetic Algorithm
ICL	Imperial College London
ISUFI	Istituto Superiore Universitario di Formazione Interdisciplinare, Lecce
MFFF	Mathematical Fitness Function Framework
P2P	Peer-to-Peer

OSS	Open Source Software
QML	Query Metamodel Language
QoS	Quality of Service
RNA	Ribonuclein Acid
SBVR	Semantics of Business Vocabulary and Business Rules
SF	Service Factory
SME	Small and Medium Enterprise
STU	Salzburg University of Applied Sciences
SUN	SUN Microsystems Iberia
TCD	Trinity College Dublin
TUC	Technical University Crete
UBham	University of Birmingham
UniS	University of Surrey
XML	eXtensible Markup Language

Index

Appendix A

Questions regarding the EvE architecture

1 Services, individuals and composition

1.1 Service manifests

1.1.1 Definition of data structures for service manifests and EvE services;

1.1.2 Is global and local usage data stored in the service manifest? If not, where else (DSS,)? Can it possibly and/or useful be split?

1.1.3 What kind of data is private?

1.1.4 How is access to private data handled?

1.1.5 Is the relation between Fada Proxy and Service Manifest one-to-many?

1.1.6 Is the relation between Service Manifest and EvE service one-to-many?

1.2 EvE composed services (service chains)

1.2.1 What are the possible structures of a service composed by the EvE? In other words, what do we optimise in the genetic algorithm? Sets, chains, trees, graphs or work flows of services?

1.2.2 What is the owner of a composition of services coming from the EvE?

1.2.3 Is it possible to access the structure of EvE composed services?

1.2.4 In case the structure is accessible, is it possible to change services?

1.3 Composition of Business Modelling Language (BML)

1.3.1 Which component composes/decomposes the BML?

1.3.2 Who defines how the composition/decomposition of BML can be done?

1.4 Composition and decomposition

1.4.1 Note: it was assumed during the meeting that the Automatic Composer takes the output of the EvE optimisation process and creates the final service (mainly creates the Business Process Execution Language (BPEL)).

1.4.2 What is going to happen after the BML composition?

1.4.3 What are the constraints for changing services after a composition and deployment to the ExE?

1.5 Service adaptation

1.5.1 Will SBVR come vital to the DBE, if yes, when?

1.5.2 Will the Service Factory (SF) comprise a SBVR Editor?

1.5.3 Is the ExE able to "run" SBVR?

1.5.4 What is the End-User Use-Case for adapting a 1) software service 2) real-world service?

2.1 Service life cycle

2.1.1 What is the current service life cycle from the computing point of view?

2.1.2 What is the EvE service life cycle?

2.1.3 How is the service manifest removal information notified to the EvE?

2.1.4 Has an EvE service pool to register in the Semantic Registry?

2.2 EvE user interaction

2.2.1 Which part of the EvE shows up to the user?

2.2.2 What are the use cases of this interaction?

2.2.3 What are the components of this interaction?

2.2.4 Where does this fit in the overall architecture?

2.3 Push mechanisms and visibility of migrated services

2.3.1 Do we want to offer SMEs services without requests in terms of profile focused advertising? (short term)

2.3.2 Do we want to offer SMEs services without requests which are evolved by a evolutionary computation? (long term)

2.4 Migration and clustering

2.4.1 What is the migration behaviour of new services?

2.4.2 What triggers migration?

2.4.3 Does migration implies moving or copying a service? Or both?

2.4.4 What are the consequences of migration on the EvE service definition?

2.4.5 What kinds of feedback does migration provide (note: necessary for clustering)?

2.4.6 What is the opportunity space? Is a cluster the opportunity space?

2.4.7 Impact of disconnected habitats (no migration) or total sharing (migration without migration)

2.5 Benefit of migration

2.5.1 Could the fitness function framework be used to measure the benefit of migrated services?

2.5.2 Can this be done without the GA?

2.6 Gateway definition and fitness function framework usage

2.6.1 Should a gateway be present for restricting entrance of migrating services into habitats?

2.7 Bootstrap of local service pool

2.7.1 How are the EvE local service pools bootstrapped in the beginning? (excessive service migration, single habitat for all habitats)

2.7.2 When a new habitat joins the EvE how it is itself bootstrapped? (copying of an existing habitat service pool with a similar profile).

3.1 User request format

3.1.1 In which language is the user request expressed 1) now and 2) in the future? (some example requests would be useful)

3.1.2 How is a user request passed to the EvE?

3.2 Comparison of user request to service description

3.2.1 How can we match a request to a service description? (e.g. growing fitness vector)

3.3 Connection to ontology

3.3.1 Clarification of connection between ontology and fitness function

3.4 Sources for optimization

3.4.1 Is the local service pool the only source for services used by the GA or also services from the Service Registry could be used?

3.4.2 How do you handle a request outside the user profile?

3.5 Input for fitness function (evolving population)

3.5.1 What are the inputs to the fitness function (user profile, individuals usage data, request, individual description)?

3.5.2 Which components are involved in providing such inputs?

2 Interfaces from EvE to other DBE components

4.1 Interfaces to ExE and SF

4.1.1 What are the components interacting with the EvE?

4.1.2 What are the interfaces to these components?

4.1.3 Where is the Habitat positioned in the architecture (inside a separate EvE component or as a service in the servent)? What are the impacts of the two solutions?

4.2 Interface to testing

4.2.1 Timeframes from testing are not viable for EvE (days to weeks)

4.2.2 What are the interfaces to testing long/mid/short term?

4.2.3 How is integrity of a EvE composed service evaluated?