Universidad de Granada

SERVICE ORIENTED ARCHITECTURE FOR ADAPTIVE EVOLUTIONARY ALGORITHMS: IMPLEMENTATION AND APPLICATIONS

Presented by

PABLO GARCÍA SÁNCHEZ

To apply for the

INTERNATIONAL PHD DEGREE IN
COMPUTER AND NETWORK ENGINEERING

Advisors

JESÚS GONZÁLEZ PEÑALVER
JUAN JULIÁN MERELO GUERVÓS
ALBERTO PRIETO ESPINOSA

Signed: Pablo García Sánchez

Mayo de 2014

Editor: Editorial de la Universidad de Granada
Autor: Pablo García Sánchez
D.L.: GR 2134-2014
ISBN: 978-84-9083-154-0

VISTO BUENO

El **Prof. Dr. D. Jesús González Peñalver**, Profesor Titular de Universidad, y los profesores **Prof. Dr. D. Juan Julián Merelo Guervós** y **Prof. Dr. D. Alberto Prieto Espinosa**, Catedráticos de Universidad, del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada,

CERTIFICAN:

Que la memoria titulada:

"Service Oriented Architecture for Adaptive Evolutionary Algorithms: Implementation and Applications"

ha sido realizada por **D. Pablo García Sánchez** bajo nuestra dirección en el Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada para optar al grado de **Doctor en Informática** .

En Granada, a 13 de Mayo de 2014.

Los Directores de la tesis doctoral:

Fdo. Jesús González Peñalver, Juan Julián Merelo Guervós y Alberto Prieto Espinosa

DECLARACIÓN

El doctorando Pablo García Sánchez y los directores de la tesis Jesús González Peñalver, Juan Julián Merelo Guervós, y Alberto Prieto Espinosa garantizamos, al firmar esta tesis doctoral, que el trabajo ha sido realizado por el doctorando bajo la dirección de los directores de la tesis y hasta donde nuestro conocimiento alcanza, en la realización del trabajo, se han respetado los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

Granada, Mayo de 2014

Pablo García Sánchez	Jesús González Peñalver, Juan Julián Merelo Guervós, y Alberto Prieto Espinosa

A mi madre, donde quiera que esté.

ABSTRACT

The objective of this thesis is to prove that the Service Oriented Architecture (SOA) paradigm can be used to create distributed, heterogeneous, dynamic and standards-based environments for Evolutionary Algorithms (EAs). SOA provides independence in programming language and transmission mechanisms, and also facilitates the dynamic component management.

A methodology to develop EAs in these environments is proposed. In this methodology, called SOA-EA, the SOA paradigm is proposed to develop Service Oriented Evolutionary Algorithms (SOEAs). The proposed methodology takes into account the requirement to develop services and EAs, and it provides the steps to identify, specify, implement and deploy the elements that conform a SOEA, and how to convert a traditional EA into a SOEA. To validate this methodology, it has been used to create a framework for SOEAs, called OSGiLiath, based in a public specification technology (OSGi). This framework provides mechanisms for dynamic component management and language and transmission independence.

OSGiLiath and SOA-EA have been used to carry out experiments in different areas to validate dynamic control and different and heterogeneous environments: uncentralized distributed EAs, other systems integration and different EA models.

RESUMEN

El objetivo de esta tesis es demostrar que el paradigma de Arquitectura Orientada a Servicios (AOS) puede usarse para crear entornos distribuidos, heterogéneos, dinámicos y basados en estándares para Algoritmos Evolutivos (AEs). AOS proporciona independencia en el lenguaje de programación y mecanismo de transmisión y facilita la administración de componentes de forma dinámica.

Se propone crear una metodología para desarrollar AEs en estos entornos. En esta metodología, denominada SOA-EA, se propone el uso del paradigma de SOA para desarrollar Algoritmos Evolutivos Orientados a Servicios (AEOS). La metodología propuesta tiene en cuenta los requisitos para desarrollar servicios y EAs y proporciona los pasos para identificar, especificar, implementar y desplegar los componentes que forman un AEOS,

y cómo convertir un AE tradicional a un AEOS. Para validarla, esta metodología se ha utilizado para crear un framework para AEOS, llamado OSGiLiath, basado una tecnología pública (OSGi). Este framework proporciona mecanismos para administración de componentes dinámicos, independencia del lenguaje y protocolo de comunicación.

Tanto OSGiLiath como SOAEA se han utilizado para realizar experimentos en distintos campos para validar el control del dinamismo y la heterogeneidad: AEs distribuidos no centralizados, integración con otros sistemas y distintos modelos de EAs.

SCIENTIFIC PUBLICATIONS

Some of the ideas, images and data exposed in this thesis have been previously published in the next references:

- Pablo García-Sánchez, J. González, Pedro A. Castillo, Maribel García Arenas, Juan Julián Merelo Guervós *Service oriented evolutionary algorithms*. Soft Comput. 17(6): 1059–1075 (2013).
- Pablo García-Sánchez, Maria I. García Arenas, Antonio Miguel Mora, Pedro A. Castillo, Carlos Fernandes, Paloma de las Cuevas, Gustavo Romero, Jesús González, Juan Julián Merelo Guervós *Developing services in a service oriented architecture for evolutionary algorithms*. In Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion. ACM, 2013. p: 1341-1348.
- Pablo García-Sánchez *A service oriented evolutionary architecture: applications and results*. In Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion. ACM, 2013. p: 1663–1666.
- Pablo García-Sánchez, J. González, Pedro A. Castillo, Juan Julián Merelo Guervós, Antonio Miguel Mora, Juan Luís Jiménez Laredo, Maribel García Arenas *A Distributed Service Oriented Framework for Metaheuristics Using a Public Standard*. In Proceeding of Nature Inspired Cooperative Strategies for Optimization. Studies in Computational Intelligence. Springer, 2010. p: 211–222.
- P. García-Sánchez, A. Fernández-Ares, A. M. Mora, P. A. Castillo, J. González and J.J. Merelo *Tree depth influence in Genetic Programming for generation of competitive agents for RTS games*. Applications of Evolutionary Computation, EvoApplicatons 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Proceedings. Springer, 2014. Lecture Notes in Computer Science (to appear).
- P. García-Sánchez, A. M. Mora, P. A. Castillo, J. González and J.J. Merelo *A methodology to develop Service Oriented Evo-*

lutionary Algorithms. Proceedings of 8th International Symposium on Intelligent Distributed Computing (IDC'2014) Springer, 2014. Studies in Computational Science (to appear).

This was a triumph!
I'm making a note here:
Huge success!
It's hard to overstate my satisfaction.
[...]
But there's no sense crying
over every mistake.
You just keep on trying
'til you run out of cake.
And the science gets done.
[...]
for the people who are
still alive.
[...]
Now, these points of data
make a beautiful line.
And we're out of beta.
We're releasing on time!
So I'm GLaD I got burned!
Think of all the things we learned!
for the people who are
still alive.
[...]
Still alive.

— Genetic Lifeform and Disk Operating System (GLaDOS), 2007.

AGRADECIMIENTOS

Hola.

Posiblemente estés leyendo esto conmigo en la misma habitación, y también posiblemente sea lo único que leas en esta tesis. Así que aquí me tienes, esforzándome en escribir esto en lugar de mejorar los capítulos siguientes. Pero bueno, uno no tiene oportunidad todos los días de leer una tesis, así que voy a explayarme.

En primer lugar, y como has leído en la página anterior, esta tesis se la dedico a mi madre. Si yo soy lo que soy es gracias a ella. Te quiero, mamá.

Y ahora seguimos con el orden estándar de agradecimientos de tesis. Primero, mis directores. A JJ, por haberme dado la mejor oportunidad que me han dado nunca y la que me ha mostrado mi vocación. A Jesús, por enseñarme a pensar antes de actuar y a

organizar lo que se debe hacer. Y a Alberto, por mostrarme que un aprendiz puede enseñar a alguien más sabio (aunque sean trucos de magia).

A Pedro y a Maribel, porque también son directores de esta tesis, aunque por motivos burocráticos no salgan en la portada. Al resto de miembros de GeNeura (Gustavo, José Luis, Carlos, Pedro G., Paloma, Antares y Javi) y en especial a Antonio y Juanlu, por aquellos desayunos míticos en La Bodeguilla.

Ahora vienen un montón de nombres, aviso.

Al resto de miembros del departamento ATC, por ser el departamento más enrollao que hay: Manolo, Julio, Alberto, Samuel, Antonio, Eva, Héctor, Mancia, Pedro, Ignacio, Encarni, Paco Illeras... ¡si es que no tengo queja de ninguno!

A todos los *hamijos* del CITIC: Fortuno, Fránsfuga (sé quien es el último cylon, ¡ja!), Quique (DQ), Leo (DL), Doresti, Nuria, Raquel-hada, Sara, Mr. Manu, Aída, Mr. Cliff, Nolo, Karl, Don Urq, Don Jpflorido, Ana (bueno, Belén), María, Ángel, el otro Juanlu Jiménez, el otro Pablo G. Sánchez, a los de BitStamina... Y a los novatos desayunantes de La Posada, claro. También a Alfonso Romero, por enseñarme lo que es de verdad ser un estudiante de doctorado (y por ponerme issues en el Github de esta tesis). A J. M. Palomares por dejarme usar su plantilla de tesis, que como veis, está muy chula. También, a la gente de mi sustitución en Ceuta, por enseñarme los sitios clave y tomar pizzas con gaviotas rondando: Jesús, Gabriel, Calixto, Natalia, Kawtar, Jose María, Bea, Rodrigo y Manolo. A la gente de la Fundación I+D del Software Libre y a los asistentes de los McDays, en especial a Rubén, José Carlos, Pedro, Miki, Mabel, Draxus y Catwoman. A los colegas de conferencias y proyectos: Carlos, Raúl y Antonio (Málaga), Anna y Anaís (Valencia), Paco y Lucas (Alicante) y los Pacos (Extremadura) por cantar Siniestro Total en las calles de varias ciudades de Europa Oriental. También a los que han colaborado en mejorar OSGiLiath en los hackathones de la Oficina de Software Libre de la UGR: Luis, Roberto, Fruela, Carlos, Daniel, Renato, Makova y Psicobyte.

To Gusz Eiben and his team at the Vrije Universiteit Amsterdam (Berend, Rob, Evert, Jean Marc, Giorgos, Eelco, Selmar, Luis and Willem): thanks for my amazing internship there. Nobody expected the Spanish Inquisition!. Y también al resto de ex-pats españoles: Rafa, Delia, Meri y Germán.

A los miembros de Poiosoft Corporation, por aquellas prácticas míticas de IS3: Iván, JP, DPP, Hose y Ñete. A los amigos de toda la vida, pues porque sí. Jesús, Nachete, Milín, Ignacio, Maria Luisa, Emilio (y sus paes!), Tania, Maricarmen, Sergio,

y Nacho. Por partidas al Uno, al Mario Kart de Game Cube o cervezacas en el mirador secreto.

A mi familia de Granada porque sois lo más. A mi familia de Sevilla porque sois la gente más buena que conozco. A mi hermana Rocío, porque yo estaba solo antes de que ella llegara (esto lo he copiado de la tesis de Juanlu, pero es que es verdad). A mi padre, Carlos, por enseñarme que para conseguir las cosas hay que esforzarse.

Bueno, y ya que estamos y hay espacio, a Batman (otra vez), al Doctor en cualquiera de sus regeneraciones, y a Ezio Auditore de Firenze, por salvar al mundo y todo eso.

Y a Ana. La persona más alucinante que jamás he conocido. Te quiero. Te quiero más que a nadie.

Menudo montón de gente. Si estás leyendo esto cerca mía y no te he nombrado es que soy un desastre y merezco que me des una colleja. En fin, supongo que uno es fruto de la gente con la que se relaciona y el conoceros a todos vosotros ha hecho que llegue a donde estoy y a lo que soy. Gracias a todos. Por cierto, si estás leyendo esto y no te conozco quiere decir que te interesa esta tesis, así que te la dedico a ti también, qué demonios. Lo pongo en inglés: if you are reading this and I do not know you, I also acknowledge this thesis to you, what the hell.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

API	Application Programming Interface
CPU	Central Processing Unit
GA	Genetic Algorithm

GP	Genetic Programming
EA	Evolutionary Algorithm
EC	Evolutionary Computation
EP	Evolutionary Programming
ES	Evolutionary Strategy
HeHa	Heterogeneous Hardware
HoHa	Homogeneous Hardware
HeSi	Heterogeneous Size
HoSi	Homogeneous Size
IDE	Integrated Development Environment
MMDP	Massive Multimodal Deceptive Problem
OSGi	Open Service Gateway Initiative
REST	Representational State Transfer
RTS	Real Time Strategy
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOEA	Service Oriented Evolutionary Algorithm
UDDI	Universal Description, Discovery and Integration
URL	Uniform Resource Locator
VM	Virtual Machine
WSDL	Web Services Description Language
XML	eXtensible Markup Language

Part I

INTRODUCTION

1 INTRODUCTION

Call me Ishmael.

— Herman Melville, Moby-Dick; or, The Whale

INDEX

1.1 GOALS OF THIS THESIS

The goal of this thesis is to create a methodology that adapts Evolutionary Algorithms (EAs) to heterogeneous, dynamic and standard-based distributed environments. This methodology proposes the use of Service Oriented Architecture (SOA) as a new paradigm to develop EAs. To validate this methodology, a framework that use all the advantages of this paradigm (dynamic binding, automatic distribution and publication of interfaces using standards) will be created. Finally, this methodology will be used to create Service Oriented Evolutionary Algorithms (SOEAs) in different scenarios, where these advantages will be reflected.

1.2 MOTIVATION

Evolutionary Computation is a scientific field that involves a large number of bio-inspired methods, problems and tools. One of these methods are the evolutionary Algorithms (EAs), a set of techniques of this field applied to optimization problems [38]. These algorithms imitate the process of natural selection, giving to fittest solutions (or *individuals*) more probability to mate with others to generate new solutions that inherit its information. Thus, iteratively, better individuals would recombine to form better solutions of the problem to solve.

Initially, the EAs were proposed as a fixed set of steps to be executed in a machine [38]. These steps can be combined to create new algorithms, or be used dynamically depending on some information during the run (for example, average quality of solutions). Therefore, they should be designed and developed as loose-coupled elements. With the advancement of Internet, new trends such as P2P, leads to a new paradigm where different software architectures, programming languages and transmission protocols collaborate to share computational resources and integration.

Because of this combination of different technologies and trends, a number of challenges in the EA area needs to be addressed. One of them is the lack of standardization and integration in EA software tools [119]. Many frameworks for EAs exist, but without the possibility of interoperation of their components. This would be desirable because it could save time and effort in development, reusing existent components to create new EAs. Establishing public and discoverable standards for computing elements not only can help in development and integration, but facilitate Open Science [52]. Finally, there are not mechanisms to deal with dynamism to manage operators (locally or remotely).

Service Oriented Architecture [118] is proposed in this thesis as a solution to address previous shortcomings. This paradigm defines the usage of loose-coupled and self-contained elements (services) based on public standards. Its aim is to facilitate the integration, interoperability and discovery in different software systems.

1.3 CHALLENGES IN EVOLUTIONARY ALGORITHMS

In the past decades much research has been conducted on Evolutionary Computation, and several challenges have been pointed out. The ones related with this thesis are shown next:

1.3.1 *Parameter Adaptation*

One of the greater challenges of the EC field is to find the appropriate values for EAs parameters, as claimed by Eiben *et al.* [36]. Researchers need to put effort on finding these values in order to attain significant performance in their EAs. Not only to adapt the numerical parameters (such as crossover rate), but also the elements that conform an evolutionary algorithm (different types of crossovers). The integration should be as easy as possible to allow researchers develop new algorithms easily.

1.3.2 *Dynamism and distribution*

As in the previous challenge, mechanisms to deal with dynamism in operators should be addressed. That is, not only the way to combine the operators that conform an algorithm, but also how to dynamically select among these available elements. Also, several authors have mentioned the problem of the limited dynamic and reflexive capabilities for loading algorithm elements (for example, problems and heuristics) in frameworks for EAs [119]. Thus, mechanisms to announce operators and automatically discovering and binding them should be used.

Dynamism should also be managed in distributed EAs. In the traditional parallelization models [6] issues such as fault-tolerance, security, churn, massive scalability or decentralization were not taken into consideration [4]. New trends on distributed EAs (such as P2P [92] or pool-based EAs [103]) are emerging, and classic programming paradigms (such as Object Orientation) do not provide mechanisms to deal with these issues.

1.3.3 *Adaptation to hardware*

Adapting algorithm parameters to available computational resources can improve performance [138]. For example, the population size in EAs is the key to obtain good performance, because it has effect on the quality of the solution and the time spent during the run [91]. This parameter has been studied as a fixed [72]

or adaptive parameter during runtime [46, 98], but without taking into account the computational power of each machine in a heterogeneous network of computers.

Also, EAs can be used in different hardware, such as mobile devices [63], "smart dust" [125] or inside robots [56], so EAs could benefit from the adaptation to the execution environment.

1.3.4 *Interoperability*

Interoperability is the ability of making systems to work together. In the past decades, many programming languages and distribution technologies have appeared. The integration of these technologies is an important problem to be addressed [118]. Although several development paradigms have been proposed (for example, the plug-in based programming [143]) to develop EAs, a recent survey in metaheuristic frameworks [119] shows that there are not any mechanism of integration in the 33 frameworks evaluated. As in the new trends previously described, researchers must also deal with heterogeneous hardware and different communication protocols, but also with dynamic, non-centralized or uncontrolled environments which expose different resources.

1.3.5 *Open Science*

It is in the field of Open Science [7] where the integration and standardization of the elements that conform an EA can take advantage, facilitating the re-use and access to existing software, systems, data, and results. Open Science is a movement that encourages the accessibility to all scientific research process open publicly to all citizens, based on free software, public licenses, open data, public scientific dissemination, and finally, well defined and publicly available services [52].

1.3.6 *Applications*

Evolutionary Algorithms have been applied to a wide number of applications from different fields. Various types of EAs, such as Genetic Algorithms, have been applied to optimize routing and inventory management [42], evolutionary art [62], evolution of robot behaviour [56], optimization of Neural Networks [18] among many others. Genetic Programming algorithms have been used for generation of agents for videogames [43] or document transformation [61].

Several yearly conferences, such as EvoApps or GECCO, present research in fields as diverse as economics [85], energy [79], videogames [107], design [30], image analysis [8], industrial environments [95], and security [73], among others. Therefore, this field is wide enough to allow the participation of researchers of many different areas and expertise, obtaining results applied to many academic, cultural and industrial fields.

1.4 OBJECTIVES

The objectives that this thesis tries to achieve are:

Objective 1: Prove that the Service Oriented Architecture paradigm can be used to create distributed, heterogeneous, dynamic and standards-based environments for EAs

EAs are a large area that deals with several fields of computer science: parallelism and distribution, parameter adaptation or development of applications and tools. Therefore, the first objective is to propose a new paradigm that can deal with some of the problems in this area: lack of standardization and mechanisms to facilitate integration, dynamism and interoperability of their components. First, the traditional classification of EAs and distributed EAs will be presented to clarify their common elements. Then, new trends in EAs (such as P2P or pool-based EAs) will be explained to show their advantages and deficiencies. Dynamic parameter adaptation and hardware adaptation works will be shown. Finally, several works on the development of EAs will be explained to extract the requirements to design EAs, and different EA frameworks will be studied, to extract their advantages and weaknesses. A new paradigm, the Service Oriented Evolutionary Algorithms (SOEAs) will be proposed to address the detected shortcomings.

Objective 2: Propose a methodology that is able to successfully adapt evolutionary algorithms to distributed, heterogeneous, dynamic, standards-based environments

This methodology will help to identify, specify, implement and deploy services to create SOEAs. This methodology will have to deal with the restrictions in the SOA and EA design identified in previous objective, to help in development, integration and dynamism.

Objective 3: Validate the methodology using a SOA technology

To validate the methodology of the previous objective, this methodology will be applied to create a framework (called OSGiLiath) using a specific SOA technology (OSGi). This framework will be used to solve some of the problems previously detected: dynamic binding of services, publication of service interfaces using public standards and save time without adding specific code for distribution. In this objective, different SOA technologies will be compared to select the most appropriate one for the problems we want to solve.

Objective 4: Prove that a SOA-based implementation of distributed, dynamic, standards-based evolutionary algorithms is able to solve efficiently different problems

The final objective of this thesis is to demonstrate that SOEAs can be used to obtain relevant scientific results in several fields. The methodology and framework will be used to perform experiments in dynamic parameter adaptation to hardware, and creation of competitive bots for video-games.

1.5 STRUCTURE OF THE THESIS

This chapter shows an introduction to this thesis, with the motivations and questions to address. The rest of the chapters are structured as follows:

The first step of this thesis is to prove that the Service Oriented Architecture paradigm can be used to create distributed, dynamic and standards-based environments for EAs (objective 1). The reason to use this kind of environments will be explained in Chapter 3, where new tendencies in distributed EAs, such as P2P or pool-based algorithms, require some mechanism to deal dynamic control of the nodes and heterogeneous architectures. Also, other shortcomings of this environments that can be addressed, such as the lack of integration of languages and transmission protocols, will be explained.

SOA will be explained in Chapter 4 as a possible way to develop this kind of distributed, dynamic, standard-based environments, as it offers mechanisms to facilitate standardization, integration, open science and dynamism. Different technologies and methodologies for SOA will be explained. Also, the requirements in SOA applied to the genericity of EAs and guidelines

to create services for EAs will be presented to accomplish the next objective: propose a methodology that is able to successfully adapt evolutionary algorithms to distributed, dynamic, standards-based environments. This methodology, called SOA-EA, will be presented in Chapter 5. SOA-EA proposes several steps to create Service Oriented Evolutionary Algorithms (SOEAs). Steps for identification, specification, implementation and development of services will be presented in that chapter, with some guidelines about how each element of the EA should be designed as a service.

To validate SOA-EA (objective 3), this methodology will be applied in Chapter 6 to create a framework (OSGiLiath) to develop SOEAs using a specific technology. To show the automatic binding of services, an experiment that adaptively enables and binds services to increase the performance of a SOEA will be presented. Different ways of exposing services publicly using standards will be shown, with a comparison of transmission time of different distribution technologies. Finally, a comparative study in development time with other frameworks for EAs will be presented.

OSGiLiath will be used in Chapter 7 to develop a new method to adapt a parameter of a distributed EA (population size) to the computational power of the nodes that execute the algorithm. To do this, SOA-EA will be applied to create automatic binding of services to create a decentralized island-based SOEA. This will be used to accomplish the objective 4: prove that a SOA-based implementation of distributed, dynamic, standards-based evolutionary algorithms is able to solve efficiently a problem of parameter adaptation to hardware.

In the next chapter (Chapters 8) OSGiLiath will be used to create SOEAs to generate competitive bots for RTS games. SOA-EA will be applied to create new dynamic services, obtaining relevant results.

Finally, chapter 9 will summarize the main contributions of this thesis and future lines of research.

Figure 1.1 shows the methodology applied to prove all the objectives of this thesis.

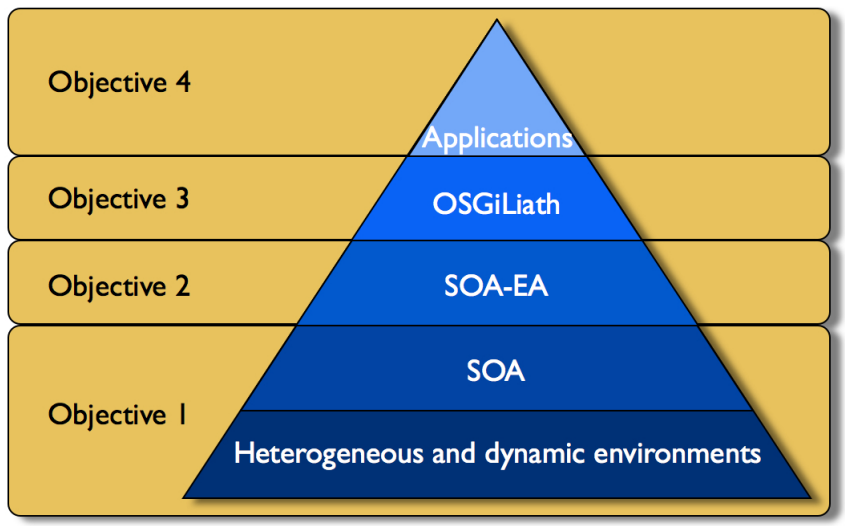


Figure 1.1
Summary of the objectives of this thesis.

2 INTRODUCCIÓN

Index

2.1 OBJETIVOS DE ESTA TESIS

El objetivo de esta tesis es crear una metodología para adaptar Algoritmos Evolutivos (AEs) a entornos dinámicos, heterogéneos y basados en estándares. Esta metodología propone el uso de Arquitectura Orientada a Servicios (AOS) como un nuevo paradigma para desarrollar AEs. Para validar esta metodología se usará para crear un *framework* que use todas las ventajas de este paradigma (enlace dinámico y distribución y publicación de interfaces utilizando estándares). Finalmente, esta metodología se usará para crear Algoritmos Evolutivos Orientados a Servicios (AEOS) en diferentes escenarios, donde se reflejarán estas ventajas.

2.2 MOTIVACIÓN

La Computación Evolutiva es un campo científico que abarca un gran número de métodos bio-inspirados, problemas y herramientas. Uno de estos métodos son los Algoritmos Evolutivos (AEs), un conjunto de técnicas aplicadas a problemas de optimización [38]. Estos algoritmos imitan el proceso de la selección natural, dando a las soluciones (o *individuos*) más adaptadas más probabilidad de cruzarse con otras para generar nuevas soluciones que hereden su información. Así, iterativamente, los mejores individuos se recombinan para formar mejores soluciones al problema a resolver.

Inicialmente, los AEs fueron propuestos como un conjunto fijo de pasos para ser ejecutados en una sola máquina [38]. Estos pasos pueden combinarse para crear nuevos algoritmos, o escogerse dinámicamente dependiendo de alguna información durante la ejecución (por ejemplo, la calidad media de las soluciones). Por lo tanto, deberían diseñarse y desarrollarse como elementos débilmente acoplados. Con el avance de internet, nuevas tendencias como el P2P traen un nuevo paradigma donde distintas arquitecturas software, lenguajes de programación y protocolos de transmisión colaboran e interoperan para compartir recursos computacionales.

Debido a esta combinación de diferentes tecnologías y tendencias, se necesitan abordar nuevos retos. Uno de ellos es la falta de estandarización e integración en las herramientas software para EAs [119]. Existen muchos frameworks para AEs, pero sin la posibilidad de interoperación entre sus componentes. Esta integración es deseable, ya que podría ahorrar tiempo y esfuerzo en desarrollo, reutilizando componentes existentes para crear nuevos EAs.

El establecimiento de estándares públicos y accesibles para elementos computacionales no sólo puede contribuir en desarrollo e integración, si no que también facilita la adopción del principio de Ciencia Abierta [52]. Finalmente, no existen mecanismos que permitan lidiar con el dinamismo en operadores (local o remotamente).

La Arquitectura Orientada a Servicios [118] es la solución propuesta en esta tesis para abordar las limitaciones previamente mencionadas. Este paradigma define el uso de elementos (llamados servicios) autocontenidos y con bajo acoplamiento. Este tipo de arquitectura aspira a facilitar la integración, la interoperatividad y el descubrimiento en diferentes sistemas software.

2.3 DESAFÍOS EN ALGORITMOS EVOLUTIVOS

En las últimas décadas, se han realizado muchas investigaciones en el campo de la Computación Evolutiva, y numerosos retos han aparecido. Los retos a abordar en esta tesis son los listados a continuación:

2.3.1 *Adaptación de parámetros*

Uno de los principales retos en el campo de la CE es el de encontrar los valores apropiados para los parámetros de los AEs, tal y como se reivindica en Eiben *et al.* [36]. Los investigadores invierten un gran esfuerzo para encontrar el valor adecuado de estos parámetros, con el objetivo de obtener un rendimiento adecuado de sus algoritmos. Sin embargo, no sólo se trata de adaptar los valores numéricos (como la tase de cruce), sino también alguno de los elementos que componen un algoritmo evolutivo (diferentes tipos de cruce). La integración debería ser lo más fácil posible para permitir a los investigadores el desarrollo fácil de nuevos algoritmos.

2.3.2 *Dinamismo y distribución*

Al igual que en el reto anterior, es necesario proveer mecanismos para gestionar el dinamismo de operadores. Esto es, no sólo la forma en la que los operadores que componen el algoritmo se combinan, sino cómo seleccionar dinámicamente entre los elementos disponibles. Además, diversos autores han mencionado el problema del limitado dinamismo y la falta de capacidades reflexivas para cargar elementos de los algoritmos (por ejemplo, problemas y heurísticas) en frameworks para AEs [119]. Por tanto, es necesario ofrecer mecanismos para la publicación de operadores y el descubrimiento y enlace automático.

El dinamismo también tiene que ser gestionado en AEs. En los modelos tradicionales de paralelización [6] no se tienen en cuenta aspectos tales como tolerancia a fallos, seguridad, caída de nodos, escalabilidad masiva o descentralización [4]. Nuevas tendencias en algoritmos evolutivos distribuidos (como redes P2P [92] o basados en *pool* [103]) han surgido en los últimos años y los paradigmas tradicionales de programación (como la orientación a objetos) no ofrecen mecanismos para gestionar los aspectos anteriormente señalados.

2.3.3 *Adaptación al hardware*

La adaptación de los parámetros de los algoritmos evolutivos a los recursos computacionales disponibles puede mejorar su rendimiento [138]. Por ejemplo, el tamaño de las poblaciones en algoritmos evolutivos es una de las claves para obtener un buen desempeño, puesto que afecta tanto la calidad de la solución ofrecida como el tiempo de ejecución del algoritmo [91]. Este parámetro ha sido estudiado como un parámetro fijo [72] o cómo un parámetro adaptativo durante la ejecución [46, 98]. Sin embargo, no se ha sido estudiado teniendo en cuenta la potencia computacional de cada máquina en una red heterogénea.

Asimismo, AEs pueden ser ejecutados en distintos tipos de hardware, como por ejemplo dispositivos móviles [63], "smart dust" [125] o en robots [56]. Los AEs pueden beneficiarse también de la adaptación al entorno de ejecución.

2.3.4 *Interoperabilidad*

La interoperabilidad es la propiedad que permite que diferentes sistemas trabajen juntos. En las últimas décadas, han surgido muchos lenguajes de programación y tecnologías de distribución. La integración de estos tecnologías es un importante dificultad a superar [118]. A pesar de que varios paradigmas de desarrollo has sido propuestos para el desarrollo de AEs (por ejemplo, la programación basada en plug-ins [143]), un reciente estudio in frameworks para metaheurísticas [119] muestra que ninguno de los 33 frameworks evaluados incluye mecanismos de integración. Como el surgimiento de nuevas tendencias como las mencionadas anteriormente, los investigadores no sólo tienen que enfrentarse con hardware heterogéneo y diferentes protocolos de comunicación, sino también con entornos dinámicos, no centralizados o no controlados que ofrecen distintos tipos de recursos.

2.3.5 *Ciencia Abierta*

En el campo de la Ciencia Abierta [7], la integración y estandarización de los elementos que componen un AE pueden ofrecer gran utilidad: facilitar la reutilización y el acceso a software, sistemas, datos y resultados ya existentes. El movimiento de Ciencia Abierta incentiva el acceso abierto al proceso de investigación científica para todos los ciudadanos, el uso de software libre,

datos abiertos, difusión de la ciencia y, finalmente, servicios bien definidos y públicos [52].

2.3.6 *Aplicaciones*

Los Algoritmos Evolutivos has sido aplicados a un amplio número de problemas en distintos campos. Distintos tipos de AEs, como Algoritmos Genéticos, han sido utilizados para la optimización de rutas y la gestión de inventario [42], arte evolutivo [62], evolución del comportamiento de robots [56] y optimización de Redes Neuronales [18], entre otras muchas posibles aplicaciones. Algunos algoritmos de Programación Genética has sido utilizados para la generación de agentes en videojuegos [43] o transformación de documentos [61].

Numerosas conferencias anuales, como EvoApp o GECCO, presentan investigaciones en campos tan diversos como economía [85], energía [79], videojuegos [107], diseño [30], análisis de imágenes [8], entornos industriales [95] y seguridad [73]. Por tanto, este campo es suficientemente amplio para permitir la participación de investigadores en diferentes áreas de estudio, obteniendo resultados aplicables a diferentes campos, ya sean académicos, culturales o industriales.

2.4 OBJETIVOS

Los objetivos que esta tesis quiere validar son los siguientes:

Objetivo 1: Probar que el paradigma de la Arquitectura Orientada a Servicios puede ser utilizada para crear entornos para AEs distribuidos, dinámicos y basados en estándares

Los AEs definen un área en la que se aplican distintas ramas de las ciencias de la computación, como el paralelismo y sistemas distribuidos, la adaptación de parámetros y el desarrollo de aplicaciones y herramientas. En consecuencia, el primer objetivo propone un nuevo paradigma que permita tratar con algunos de los problemas presentes en este área: falta de estandarización y mecanismos para facilitar la integración, el dinamismo y la interoperatibilidad de sus componentes. En primer lugar, la clasificación tradicional de AEs y AEs distribuidos será presentada para clarificar sus elementos comunes. A continuación, se explicarán las nuevas tendencias en AEs (como P2P y basadas en *pools*), mostrando sus ventajas y deficiencias. Un nuevo paradigma, los Algoritmos Evolutivos Orientados a Servicios (AEOSs)

será propuesto para tratar de paliar las desventajas detectadas en otros paradigmas.

Objetivo 2: Proponer una metodología que sea capaz de adaptar exitosamente algoritmos evolutivos en sistemas distribuidos, dinámicos, heterogéneos y basados en estándares

Esta metodología deberá facilitar la identificación, especificación, implementación y despliegue de servicios para crear AEOS. Esta metodología también tendrá que tratar con las restricciones propias de los AE y SOA identificadas en el objetivo previo, para facilitar el desarrollo, integración y dinamismo.

Objetivo 3: Validar la metodología usando una tecnología SOA

Para validar la metodología presentada en el objetivo anterior, ésta será aplicada para la creación de un framework (llamado OSGiLiath) utilizando un tecnología SOA específica: OSGi. Este framework será utilizado para abordar algunos de los problemas previamente detectados: enlace dinámico de servicios, publicación de interfaces de servicio usando estándares públicas y el ahorro de tiempo, eliminando la necesidad de escribir código específico para la distribución. En este objetivo, distintas tecnologías SOA se compararán para seleccionar la más apropiada para cada problema a resolver.

Objetivo 4: Probar que una implementación basada en SOA de algoritmos evolutivos distribuidos, dinámicos y basados en estándares tiene la capacidad de resolver de forma eficiente distintos problemas

El objetivo final de esta tesis es demostrar que los AEOSs pueden ser utilizados para obtener resultados de relevancia científica en distintos campos. La metodología y el framework serán usados en una serie de experimentos en adaptación dinámica de parámetros al hardware y la creación de bots competitivos para videojuegos.

2.5 ESTRUCTURA DE LA TESIS

Este capítulo ofrece una introducción a esta tesis, incluyendo su motivación y las preguntas a abordar.

A continuación se expone la estructura del resto de capítulos:

El primer paso de esta tesis es probar que la Arquitectura Orientada a Servicios puede ser utilizado para crear entornos para AEs que sean distribuidos, dinámicos y basados en estándares (Objetivo 1). La razón para utilizar este tipo de entornos será explicada en el Capítulo 3, donde se muestra cómo las nuevas tendencias en AEs distribuidos, como P2P y algoritmos basados en *pool*, requieren ciertos mecanismos que proporcionen control dinámico de los nodos y arquitecturas heterogéneas. Además, serán explicados otros defectos de este tipo de entornos que pueden abordarse, como la falta de integración de los lenguajes y protocolos de comunicación.

SOA se explicará en el Capítulo 4 como un posible paradigma para el desarrollo de este tipo de entornos distribuidos, dinámicos y basados en estándares, al ofrecer mecanismos que facilitan la estandarización, integración, la ciencia abierta y el dinamismo. Diferentes tecnologías y metodologías para SOA serán explicadas. Por otra parte, se presentarán los requerimientos en SOA aplicados a la genericidad de los AEs, así como las directrices para la creación de servicios para AEs que permitirán cumplir con el próximo objetivo: proponer una metodología que pueda adaptar con éxito algoritmos evolutivos a sistemas distribuidos, dinámicos y basados en estándares. Esta metodología, llamada SOA-EA, será presentada en el Capítulo 5. SOA-EA propone varios pasos para crear Algoritmos Evolutivos Orientados a Servicios (AEOSs). Los pasos para la identificación, especificación, implementación y desarrollo de servicios serán también presentados en este capítulo, junto con algunas directrices acerca de cómo deberá ser diseñado como un servicio cada elemento de un AE.

Para validar SOA-EA (objetivo 3), esta metodología será aplicada en el Capítulo 6 para crear un framework (OSGiLiath) para el desarrollo de AEOSs utilizando una tecnología específica.

Con el objetivo de mostrar el enlace automático de servicios, se presenta un experimento en el cual se activa y enlaza de forma adaptativa servicios para incrementar el rendimiento de un AEOS. Se mostrarán diferentes formas de publicar servicios utilizando diferentes estándares, comparando los tiempos de transmisión de cada tecnología de distribución. Finalmente, se presenta un estudio comparativo de los tiempos de desarrollo con distintos frameworks para AE.

OSGiLiath se utilizará en el Capítulo 7 para desarrollar un nuevo método para la adaptación de parámetros de un AE distribuido (tamaño de la población) a la potencia computacional de los diferentes nodos en los que se ejecuta el algoritmo. Para efectuar este experimento, SOA-EA será utilizado para diseñar

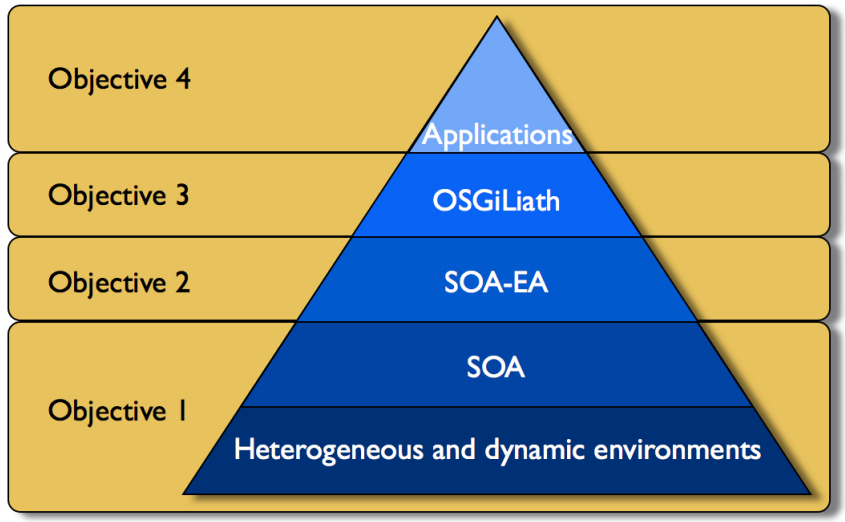


Figure 2.1
Resumen de los objetivos de esta tesis

servicios para la creación de un AEOS descentralizado y basado en islas. De esta forma, se alcanzará el objetivo 4: probar que una implementación basada en SOA de algoritmos evolutivos distribuidos, dinámicos y basados en estándares tiene la capacidad de resolver de forma eficiente distintos problemas.

En el siguiente capítulo (Capítulo 8), se utilizará OSGiLiath para crear AEOSs que generan bots competitivos para juegos RTS. SOA-EA será aplicada para la creación de servicios dinámicos, obteniendo resultados relevantes.

Finalmente, el Capítulo 9 resumirá la principales contribuciones de esta tesis y las futuras líneas de investigación.

La Figura 2.1 muestra cómo la metodología se aplicará para probar todos los objetivos de esta tesis.

3 EVOLUTIONARY ALGORITHMS

I have called this principle, by which each slight variation, if useful, is preserved, by the term of Natural Selection.

— Charles Darwin, The Origin of Species (1859), Chapter III

INDEX

Evolutionary Algorithms are a set of bio-inspired techniques applied to optimization problems [38], based on the process of natural selection [24]. In this kind of algorithms, a POPULATION of codified solutions (called INDIVIDUALS) is created. The most adapted individuals have more chances to be selected for reproduction, so their offspring could inherit their genetic material. The level of adaptation of each solution is measured using a FITNESS function, that usually models the problem to solve.

Initially the EAs were proposed as a fixed set of steps with different operators and data structures. Then, several ways to parallelize these algorithms were also presented, with new classifications and operations. Nowadays, new emerging trends that deal with new technologies (such as P2P or Cloud Computing) and new algorithmic methods (such as parameter adaptation) are being used. These trends require a new way to develop EAs taking into account some shortcomings, such as integration of heterogeneous elements or dynamic resources, and also to deal

with fault-tolerance, churn, massive scalability or decentralization.

As previously said in the introduction, the aim of this thesis is to facilitate the development, standardization, integration and dynamism in EAs. Therefore, in this chapter the classic classification of EAs is presented to clarify their common elements, together with the most extended models to parallelize these algorithms. This classification can be used as a base for the creation of new algorithms, establishing their similarities, to facilitate the *development*. Then, the new trends in EAs are also presented to summarize their benefits, but also their deficiencies and problems that should be addressed (such as *dynamism* of computing nodes or in the elements that form the EA). Finally, some of the most used frameworks to develop EAs are listed and analysed, to understand their capabilities and deficiencies (such as the lack of *standardization* and *integration*). This way, in following chapters, the design and development of interoperable, standardized and dynamic services for EAs will have a solid base to start.

3.1 TYPES OF EVOLUTIONARY ALGORITHMS

The general scheme of an EA, extracted from the work of Eiben and Smith [38] is described in Figure 3.1. Although most of the EAs follow the scheme shown in this Figure, they have differences depending on the representation of the solutions, the problems to solve, and other features. A possible classification is presented in this chapter, taking into account the new approaches that cannot fit in the traditional taxonomy. This classification would help to clarify the elements that distinguish an algorithm from another (for example, the operators), and the existing similarities and differences, in order to establish a good starting point for designing services for EAs.

Figure 3.1 General scheme of an evolutionary algorithm in pseudo-code

```
BEGIN
 INITIALISE population with random candidate solutions;
 EVALUATE each candidate;
 REPEAT UNTIL (TERMINATION CONDITION is satisfied) DO
   1 SELECT parents;
   2 RECOMBINE pairs of parents;
   3 MUTATE the resulting offspring;
   4 EVALUATE new candidates;
   3 SELECT individuals for the next generation;
 OD
END
```

3.1.1 *Classic classification of EAs*

This subsection explains the traditional variants, according to the book of Eiben and Smith [37]. These authors clarify that the features of an EA are:

- EAs are population based.
- EAs mostly uses recombination to generate new individual from the existing ones.
- EAs are stochastic.

These are the elements that distinguish the EAs from another meta-heuristics (such as the Local Search [1], for example).

GENETIC ALGORITHMS These kind of algorithms were proposed by Holland [78], and also studied by Goldberg [65] and Michalewicz [106]. In this kind of EA, the representation of the solution is a string of numbers (usually binary), called CHROMOSOME (and sometimes GENOME). The individuals are selected proportionally to their fitness, and then RECOMBINATION and/or MUTATION are applied to generate new individuals that will be introduced in the population. These algorithms have been used in different areas, such as function optimization [106], combinatorial optimization [42], artificial intelligence in videogames [48], or generative art [62], among others.

EVOLUTION STRATEGIES The Evolution Strategies (ES) are used to solve problems whose solution is included in the domain of real numbers. Their main difference with GAs is the inclusion of self-adaptation of the mutation rate, being coded in each individual [33]. Also, the parent selection is performed randomly. ES have been applied in fields such as Evolutionary Robotics [56].

EVOLUTIONARY PROGRAMMING In Evolutionary Programming (EP), the representation of the solution depends on the nature of the problem being solved, for example, neural networks [18] or Radial Basis Functions (RBFs) [68] have been used as individuals.

GENETIC PROGRAMMING The objective of this technique is to create functions or programs to solve determined problems. Individual representation is frequently in the form of a tree, formed by operators (or *primitives*) and variables (*terminals*). These sets are usually fixed and known. The genome size is,

therefore, variable, but the maximum size of the individuals is commonly fixed, to avoid high evaluation costs. GP has been used to evolve LISP programs [87], or XSLT scripts [61], among others.

LISP: *LISt Processing*
XSLT: *eXtensible Stylesheet Language Transformations*

3.1.2 *Other models*

Other EAs that do not match in the previous classification have been proposed. For example Differential Evolution (DE) [137], Estimation of Distribution Algorithms (EDAs) [93] or Bayesian Optimization Algorithms [121].

Combining elements of all the previous algorithms with other heuristics is the base of Memetic Algorithms (MAs). These algorithms are based on the concept of *meme* proposed by Dawkins [26]. In this context, a meme is domain specific knowledge coded by a computational representation to the effective solving of a problem.

This kind of algorithms can be seen as an hybrid combination of the population-based evolution methods previously explained, coupled with some kind of local search. Initially, the hybridization was made just combining two o more methods with some kind of problem knowledge. For example combining a genetic algorithm with Simulated Annealing (SA) [124]. The local search can be performed before, during, or after the evaluation. New trends, such as adaptive MAs [116] lead to the usage of several memes for searching and deciding dynamically which meme should be applied to each individual. For example, Cowling *et al.* proposed the term hyperheuristic [20] as the strategy to choose the meme to be applied depending on the time and the region of the search space (or, in a brief, heuristic to choose memes). Other examples are the work of Krasnogor *et al.* [88], where the inclusion of the memetic codification inside the individual chromosome (Multimemes) to select which meme to use is applied; or codify a set of rules, as proposed by Smith [132] (Co-evolving MAs).

In brief, the memetic algorithms can be seen as a composition of other algorithms. The integration should be as easy as possible to allow researchers develop new algorithms easily. Moreover, not only the way to combine the algorithms that conform the memetic one, but also how to dynamically select among these available memes. Thus, mechanisms to announce memes and automatically discovering and binding should be applied.

3.2 PARALLEL AND DISTRIBUTED EVOLUTIONARY ALGORITHMS MODELS

Evolutionary Algorithms are inherently parallelizable, since each individual can be considered as an independent unit [4]. Thus, there exist several ways to parallelize: for example, fitness evaluation can be distributed into several slave machines, or the population can be distributed among different nodes to be evolved at the same time. In 2002, two main types of parallelization models for EAs were classified by Alba and Tomassini in [6]: GLOBAL PARALLEL EAs and SPATIALLY STRUCTURED EAs. However, with the aim of new technologies and architectures, such as P2P, new ways to parallelize EAs have been proposed. In this section, the differences of the existing classification with these new trends are explained, and the requirements of each one are presented.

3.2.1 *Traditional parallelization classification*

As this classification was established in 2002, issues such as fault-tolerance, churn, massive scalability or decentralization were not taken into consideration. The number of computational nodes are fixed during the whole execution and both the network and the nodes are reliable and trustworthy.

Global parallel evolutionary algorithms

In this model, also called FARMING MODEL, MASTER-SLAVE or CENTRALIZED EA, the parallelism is applied at evaluation level, where a central node coordinate several slave nodes. The central node executes the EA in a sequential way, but distributes the individuals of the population to the slaves just for being evaluated. Figure 3.2 depicts this situation.

Spatially structured algorithms

The parallelism is performed at population level, that is, dividing the population among the different computing elements. Depending on how the distribution is performed we have:

COARSE-GRAINED APPROACH One of the most usual approaches is the ISLAND MODEL, where a number of nodes executes simultaneously the EA, working with different sub-populations at the same time. Each certain number of generations some individuals are interchanged (migrated) between populations. Figure 3.3 shows this model with a ring topology.

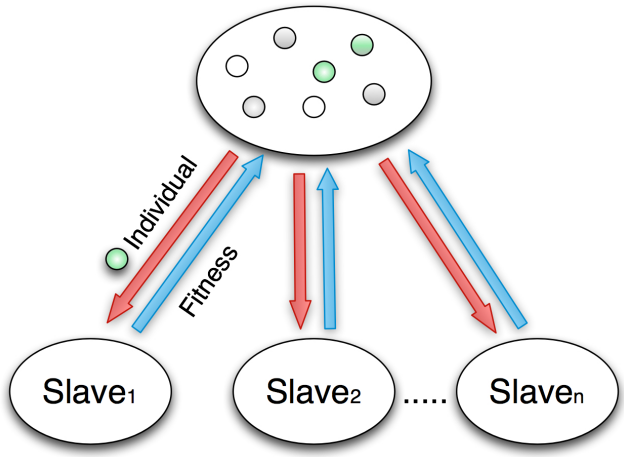


Figure 3.2
Master-slave model.

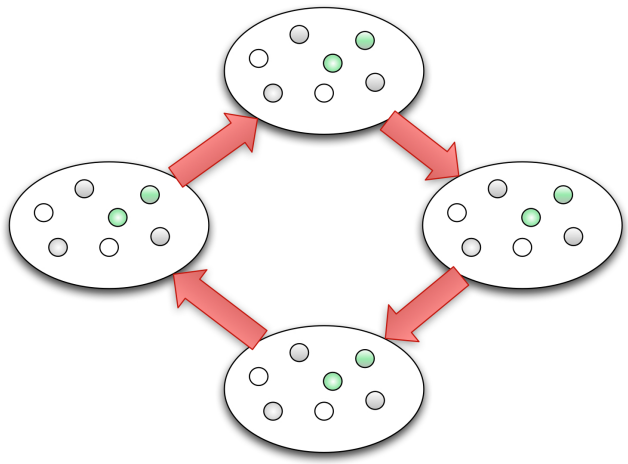


Figure 3.3
Island model scheme using a neighbourhood ring topology.

FINE-GRAINED APPROACH In this approach, also called CELLULAR EA (CEA), each node has one individual of the population, and selection and reproduction are limited to the individuals of the neighbourhood of the node [31]. Usually a bi-dimensional grid is used as topology, such as the one showed in the Figure 3.4.

3.2.2 *New trends on parallel EAs*

When developing distributed and parallel EAs (or any distributed system in general), we should deal with the *Fallacies of Distributed Computing*, proposed by Peter Deutsch and then explained by Rotem-Gal-Oz in [126].

1. The network is reliable.
2. Latency is zero.

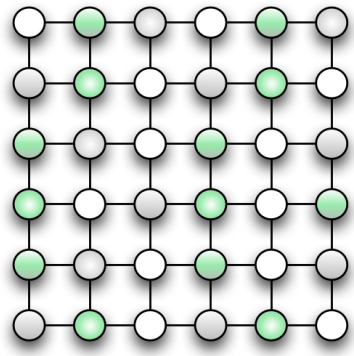


Figure 3.4 Cellular Evolutionary Algorithm.

3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

These fallacies were not taken into account in the previous classification. However, with the advent of new technologies, such as cloud computing, P2P networks, or the usage of heterogeneous hardware, new approaches have been proposed. Distributed EAs can be executed in other computing elements different than the classic computers. For example, in mobile devices [63], "smart dust" [125] or inside robots, learning from the environment in an on-line manner [56]. But with the advancement of the Internet, where millions of nodes can co-operate, and whose behaviour is not totally controlled or predicted, is when new distributed approaches have become more evident.

P2P: *Peer-to-peer*

P2P systems are parallel infrastructures composed by a large number of resources, without any central server [136]. In practice, the resources in these networks can appear or disappear dynamically. These platforms can be used to execute large instances of problems, taking advantage of the massive scalability that these systems potentially offer. An example of one EA that has been designed to take advantage of these systems, is EvAg, proposed by Laredo *et al.* [92]. This algorithm uses a decentralized population, where each peer has a single individual, and new individuals are created combining the ones in their current neighbours. To solve the problem of the dynamic topology,

EvAg: *Evolvable Agent*

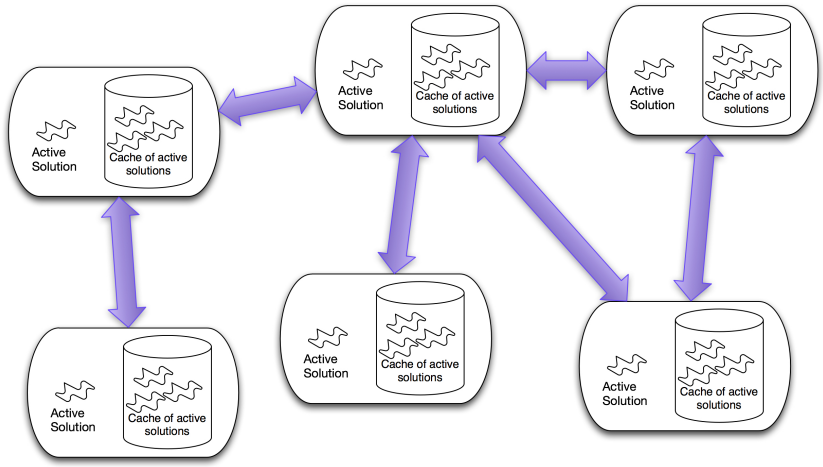


Figure 3.5
P2P Evolutionary Algorithm: EvAg.

the population structure is maintained using the NEWCAST protocol [82]: each node has a cache of neighbours that can be interchanged and combined. Figure 3.5 shows this algorithm and its population structure. Results show that this algorithm outperforms tuned GAs, using less links that a panmictic (i.e. fully connected) population.

Technologies such as non-relational databases lead to POOL-BASED EAs [69], where the computational nodes exchange individuals using a shared pool. Although this can be seen as a farming model, there are differences in how the data flow of individuals is managed. This allows massive scalability with a heterogeneous underlying structure. This pool can be used as the global population, and the nodes asynchronously read and evaluate the individuals. It can also be used to share individuals among islands. This can lead to automatic load-balancing and synchronization, allowing the addition and removal of nodes. Different technologies can be used. For example, in the work of Meri *et al.*, the pool used is based on the Dropbox™ or SugarSync™ file storage services. Other authors propose the use of non-relational databases, such as the work of Merelo *et al.* [69], using FluidDB™. In [103] the same authors improve the design, proposing an asynchronous, fault-tolerant, and scalable dEA, based on the object store CouchDB™. The results show that adding clients could not scale, but increase the fault tolerance. Also, their experimentation shows a good methodology for designing EAs in heterogeneous distributed systems, which have the impossibility of analytic performance prediction.

Other systems, such as Grids [21] are distributed computing systems that allow sharing geographically distributed resources

to solve large scale problems. Several works that describe EAs being executed in these systems have been presented [80, 96, 111]. VOLUNTEER COMPUTING [9] proposes the creation of infrastructures to allow people donate CPU cycles for a combined computational effort. Its main difference with grid computing is the presence of un-trusted resources: for example, some nodes could return intentionally wrong results, so it requires the possibility of replication of the results to be validated. Also, the control of the participating nodes is not maintained by the experiment launcher. EAs have been executed in these systems, using several techniques, such as virtualization [140] or "parasite" computing [70].

Summarizing, with these new trends in parallel EAs, researchers must deal with heterogeneous hardware and different communication protocols, but also with dynamic, non-centralized or uncontrolled environments.

3.3 PARAMETER ADAPTATION IN EVOLUTIONARY ALGORITHMS

One of the greater challenges of the EC field is to find the appropriate values for EAs parameters [36]. Usually, these parameters are established by convention or after several test runs, for example. However, practitioners need to put effort on finding these values in order to attain significant performance in their EAs, even taking into account other variables, such as the computational power of the machines used.

3.3.1 *Parameter Control and Parameter Tuning*

There are two different approaches for algorithm parameter setting in EC: *parameter control* and *parameter tuning* [39]. The first one refers to setting up a number of parameters for the Evolutionary Algorithm (EA) and changing them in running time. The parameter tuning consists in establishing a good set of parameters *before* the run (and not changing these parameters during runtime).

Eiben, in [35] proposes the next taxonomy for the parameter control, according to *how* the are changes made:

- Deterministic methods: changes to a parameter are triggered by a deterministic rule (for example, increase mutation rate after certain number of generations).

- Adaptive methods: parameters change depending on some behaviour (for example, increase mutation rate if the average fitness of the population stagnates).
- Self-adaptive methods: parameters are encoded within the chromosome of the individuals of the population (for example, mutation rate can be a value in the chromosome).

Other classifications for control techniques in EAs are presented in that work. For example, regarding to *what* is changed:

- Representation
- Evaluation function
- Variation of operators and their rates
- Selection operators
- Replacement operator
- Population

Finally, a third classification can be obtained according to what evidence is available:

- Absolute evidence: The parameter changes if a rule is activated when an specific event occurs. For example: increase mutation rate when diversity drops. In this case, human knowledge is necessary to model these rules.
- Relative evidence: Parameters are compared with the fitness of their produced offspring and the best values are rewarded. It is not deterministic.

These classifications can be used to establish a way to manage or update the parameters in the development of algorithms. For example, it should be necessary to allow a direct access to each parameter and sources of information of the EA, independently of the actual state of the algorithm. A possible solution to manage the dynamic parameters, and the information of the elements that conform the EA, is proposed in next chapters.

3.3.2 *Adaptation in heterogeneous hardware*

Adapting algorithm parameters to available computational resources leads to improved performance (see the work of Hamadi and Schoenauer [138]). An easy way to take advantage of the available resources is balancing the workload [54] to distribute it

across multiple nodes. However, assigning the same tasks to each node on heterogeneous clusters may result in suboptimal performance (as shown by Bohn and Lamont [14]). Parameters of an algorithm could also be adapted to increase the performance of the whole system. For example, the population size in EAs is the key to obtain good performance, because it has effect on the quality of the solution and the time spent during the run [91]. This parameter has been studied as a fixed [72] or adaptive parameter during runtime [46, 98], but without taking into account the computational power of each machine in a heterogeneous cluster.

The adaptation of an algorithm can be useful to leverage different hardware environments. One of the problems in parameter adaptation in heterogeneous clusters is the representation of the computational load. It depends on the algorithm, size of the problem, programming language, compiler or hardware characteristics, and the results obtained from artificial benchmarks (such as Linpack [41]) should not be extolled as identificative of the system performance [29]. For example, in the work of Garamendi *et al.* [54], a small benchmark was executed in all nodes at the beginning of the algorithm in order to distribute individuals of an Evolutionary Strategy, following a master-slave model. However, the computational load by artificial benchmarks may not accurately represent the correct load of the algorithm, so, information about the algorithm itself should be used for calibration.

In other works, there is no direct relation between the algorithm parameters and computational resources of the nodes. For example, Domínguez *et al.* [28] divided the available devices into "faster" and "slower" nodes to create a distributed hybrid meta-heuristic that combines two different EAs: Genetic Algorithms and Simulated Annealing. Their system executes the heavy (in computational terms) algorithms (GAs) in the faster nodes (computational devices), and simpler meta-heuristics (SA) in the slower ones, obtaining better results than other configurations. Gong *et al.* in [67] also ordered the nodes by their computational power to test different topology configurations in a distributed EA. Besides, ordering the nodes taking into account only their previously known computational resources, the results of the previous works were not compared with executions on a homogeneous cluster to validate if the adaptation takes advantage of the heterogeneity of the cluster.

The heterogeneous computational performance of nodes or network speed can affect the performance of an algorithm. In [5], Alba *et al.* compared a distributed Genetic Algorithm (dGA), one

of the sub-types of EAs, on homogeneous and heterogeneous clusters. Super-linear performance in terms of iterations was obtained in the heterogeneous ones, being more efficient than the same algorithm running on homogeneous machines. However, the parameter setting was the same in both clusters and they did not adapt the parameters to the machines used.

Adapting algorithm parameters to different nodes derives in heterogeneous parameter sets. These sets can improve the results in homogeneous hardware, for example, setting a random set of parameters in each homogeneous node can also increase the performance of a distributed Genetic Algorithm, as explained by Gong and Fukunaga in [66]. That model outperformed a tuned canonical dGA with the same parameter values in all islands. Also, adapting the migration rate produced better results than homogeneous periods, as explained by Salto and Alba in [128]. This indicates that heterogeneous parameters may lead to an increase of performance, so it is necessary to validate if the performance is due to the parameter set or to the heterogeneous devices combination. The way to access the elements that conform an EA (including their parameters) proposed in this thesis will be used in next chapters to address this issue.

3.4 DEVELOPMENT OF EVOLUTIONARY ALGORITHMS

One of the aims of this thesis is to facilitate the standardization, integration and development of EAs.

This is due to the existing large number of frameworks for Evolutionary Algorithms. Practically every programming language has its own implementation of the basic elements that form an EA. This implies a large effort made in each one, giving that these elements are not compatible among them. It is also difficult to migrate the code from a framework to another, mainly due to design choices, such as the existence of hidden global variables or language specific features (such as functional programming [22]). This is important for example, to integrate and reuse the elements of the frameworks in other systems (such as enterprise servers, for example). However, is in the field of Open Science [7] where the integration and standardization of the elements that conform an EA can take advantage, facilitating the re-use and access to existing software, systems, data, and results.

In this section, the genericity, communication and features of the development of EAs and existing frameworks are explained and compared to evaluate their benefits and shortcomings.

3.4.1 *Design of EAs*

The work of Gagné and Parizeau [53] established six criteria to qualify the genericity of a framework for EAs.

- *Generic representation*: independence of the structure of the individuals.
- *Generic fitness*: Individual fitness should be as independent as possible from the selection operators and individual representation. That means that the fitness evaluation should be outside of the implementation of the individuals (for example, not implementing the fitness in the class that implements the individual).
- *Generic operations*: operations should be used in conjunction with others and have minimal side effects. For example, a multi-parent reproduction (crossover with more than two parents, as presented by Eiben et al. [34]) requires separation of the concept crossover with only two elements, so the abstract concept of crossover should accept a list of individuals. Also, new operations can be created without affect the existing ones, allowing the interaction of a non-limited number of operators. Finally, the granularity of the design of the operators should be equilibrated, not being neither too coarse (may limit the flexibility to create new operators) nor too fine (could be difficult to integrate all the operators).
- *Generic evolutionary model*: As explained in Section 3.1.1, there exist different ways to model an EA leading to different algorithms (for example, a steady-state GA versus a generational GA, or a GA versus ES). Operators should be independent of the evolutionary model, being possible the change from a model to another.
- *Parameter management*: parameters, such as the population size, may be modified during runtime. Also, a good framework should accept the addition of new parameters.
- *Configurable output*: the output should be configurable. This is due to different statistics that could be used depending on the EA: for example, the tree depth in GP. Also, different outputs (console, files) should be managed.

These issues should be accomplished to develop new algorithms or operators, with independence of the programming language used. However, as will be explained in next chapters, new

Table 3.1 Comparison of EA frameworks. OO=Object-Oriented, SO=Service Oriented, PO=Plug-in Oriented

Name	Design	Language	Distribution	License	Last version
ECJ	OO	Java	Sockets	Academic Free Lic.	2013
MALLBA	OO	C++	MPI	Freeware	2010
jMetal	OO	Java	N/A	GNU/LGPL	2013
DREAM	OO	Java	DRM	GNU/GPL	2003
ParadiseEO	OO	C++	MPI	CeCILL	2012
HeuristicLab	OO/PO	.NET	Web-Services	GNU/GPL	2013
METCO	OO	C++	MPI	N/A	2009
JCLEC	OO	Java	N/A	GNU/GL	2013
Algorithm::Evol.	OO	Perl	N/A	GNU/GPL	2013
gridUFO	SO	Java	Web Services	N/A	2010

models of design and development can extend the previous criteria.

3.4.2 *Frameworks for EAs*

Over the large number of available frameworks (see [119] for a complete survey) a representation of them has been selected to explain their shortcomings, that will be addressed in this thesis.

Object Oriented programming is used in several frameworks, such as Algorithm::Evolutionary [104], METCO [94], JCLEC [141] or jMetal [32]. Users implement specific interfaces of these frameworks (*individual* or *crossover*, for example) and they group them in the source code. For example, creating an operator object that groups several operators. However, these frameworks are not compatible among them. For example, the operators created in JCLEC can not be used directly in jMetal (despite both are programmed in Java).

METCO: *Metaheuristic-based Extensible Tool for Collaborative Optimization*
JCLEC: *Java Class Library for Evolutionary Computation*

Parallelism and distribution are possible in other frameworks, such as MALLBA [2], DREAM [10] or ECJ [100], but using external libraries (such as MPI or DRM), so the code that uses these libraries is mixed with the algorithm's code.

DREAM: *Distributed Resource Evolutionary Algorithm Machine*
ECJ: *Evolutionary Computation in Java*
MPI: *Message Passing Interface*
DRM: *Distributed Resource Machine*

Even being distributed, these frameworks can not communicate with each other. This implies an extra effort to combine the capabilities that a framework can offer to other frameworks or other programs (for example, a web server). HeuristicLab [142] is one of the few plug-in and service oriented frameworks. It uses web services for communication, but only to distribute the load, after consulting a central database of available jobs. Finally, gridUFO is a service oriented framework [110], but it only allows the modification of the objective function and the addition of whole algorithms, without combining existing services. Table 3.1 shows a summary of the previous frameworks.

In brief, although these frameworks follow the six criteria for genericity proposed by Gagné and Parizeau previously explained,

they present some shortcomings when it is needed to develop or add new features: the user is forced to modify the source code or stop the execution to add new functionalities (like load balancing, dynamic control of operators, or an user interface).

Other frameworks, not focused on EAs, can be used to deal with some of the issues, such as dynamism in nodes. BOINC is one of the most used frameworks in volunteer computing. This middleware follows a master-slave architecture, where the server is in charge of hosting the project experiments and the creation and distribution of jobs [140]. Clients ask the server for works, download information, compute data and upload the results. EAs have been used in BOINC, such as the work of Fernández *et al.* [140]. Other authors imitate this architecture using a browser-based scheme [70] to distribute fitness evaluations among clients without installing any other software. Previous systems have the possibility of task distribution among the nodes, following a master-slave model, but without interaction between clients. Also, these systems does not count with automatic discovering of operators.

BOINC: *Berkeley Open Infrastructure for Network Computing*

3.5 CONCLUSIONS

Evolutionary Algorithms, and their subtypes (GAs, ES or GP, among others) follow a number of common steps: initialization, evaluation, selection, recombination, mutation, replacement and stop criterion. There exist many variations of these steps, and the different combinations can specify one algorithm or another. Memetic Algorithms also include extra elements that can be applied, and different heuristics can be combined during the algorithm's run.

Distributed EAs can improve the algorithmic and computational performance over the non-parallel versions of the algorithms. Classic parallelization approaches, such as the master-slave or island-based models, have been updated with the usage of new trends such as P2P or pool-based EAs. These new approaches manage with computational nodes entering and exiting during the experiment runtime and heterogeneous architectures.

Other research lines, such as the parameter adaptation can imply the existence of some kind of dynamism involving the parts that compose an algorithm: for example, different recombinators or mutators working at the same time. Moreover, there exist several lines of parameter adaptation in dynamic and heterogeneous environments, where different computational elements are working at the same time.

Finally, there exist a large number of different (and incompatible) frameworks for EAs, each one using different languages, technologies and communication protocols. As Parejo *et al.* suggest in [119], a standardization of the presented (and other) frameworks should be carried out. Moreover, it is difficult to access, in a public way, to available public systems to execute existing EAs to validate experiments and save time, encouraging Open Science.

Next chapter will explain a possible technological solution to cope with the previous issues that will be addressed in this thesis: development, integration, standardization and dynamism in EAs.

Part II

MATERIALS AND METHODS

4

SERVICE ORIENTED ARCHITECTURE: TECHNOLOGIES AND RESTRICTIONS FOR DESIGNING SERVICES FOR EAS

Service with a smile?

— The Joker. The Last Laugh. Batman: The Animated Series.

INDEX

The previous chapter has explained several shortcomings in the development of EAs in some contexts, mainly related with the integration of different frameworks, distributed programming and heterogeneity of computational environments, among others. This chapter explains the concept SERVICE ORIENTED ARCHITECTURE (SOA), with several associated technologies and methodologies, as a possible solution for these issues.

Research in SOA [118] is a growing field, as can be seen in Figure 4.1, obtained from the search terms *"service oriented OR service-oriented"* in the Scopus [1] database. Each year more papers about the topic are published. This area seeks to promote services usage and adoption, and to improve the way to use them. For example, solving a problem combining existing services in an automatic way [109]. Not only in the academic world, but also in the industry, with more than a seventy percent of adoption and satisfaction [75], adding significant value to the enterprises [74].

1 http://www.scopus.com

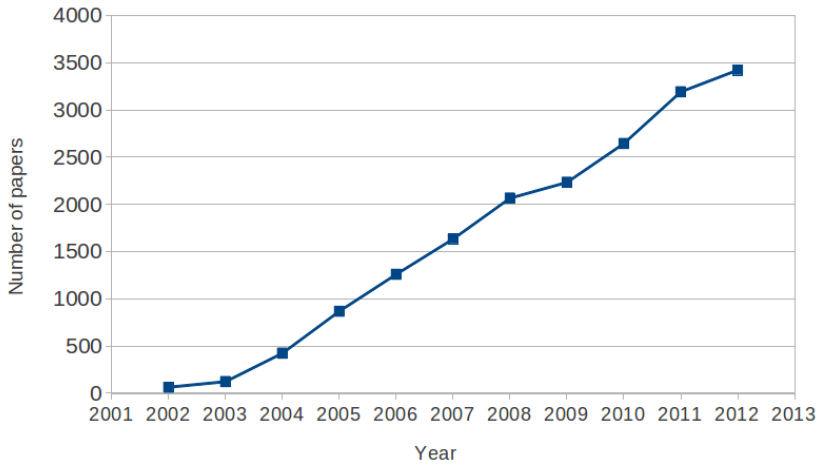


Figure 4.1
Number of published papers (per year) about SOA (obtained from Scopus database).

Service Oriented Architecture (SOA) is a computational paradigm where agents interact with each other using loosely coupled, coarse-grained, and autonomous components called *services* [127]. A service is a distributed entity (such as a node, program or function), used to obtain a result, increasing the integration of heterogeneous systems (several operating systems, protocols or languages) due to this multi-platform nature. The service users do not need to know the language used to implement the service, and they are not forced to use a specific technology to access that service. For example, an evolutionary algorithms researcher could have access to a fitness function made publicly available by another researcher at the other side of the world without even knowing which programming language has been used to implement it.

Also, with the advancement of the Internet, new scientific communities, based on interoperable and distributed platforms are emerging. These communities allow scientist to collaborate on their research, sharing data and remote access to their programs. To achieve this, they use SOA, obtaining the benefits of the standards it offers. Users publish and use flexible, interoperable and configurable services. These services can be created from scratch or by leveraging existing software [12].

Foster [52] defines the term "Service Oriented Science" as the pursuit of scientific research using distributed and interoperable networks, being the uniformity of these interfaces the key to success. Thanks to it, researchers can discover and access the services without developing specific access for each data source, or program. Therefore, this paradigm has the potential to increase the scientific productivity due to these public and dis-

tributed services, and also to increase the data analysis automation in computing. There are many examples that attempt to boost this paradigm, like Open Science Grid [7] and GLOBUS [51]. These projects include scientific communities and globally distributed infrastructures that support scientific and integrated applications of different domains.

It is necessary to remark that the technology for implementing services is not the key challenge in SOA, but to increase the effort to migrate the existing work and to change the mind of researchers and practitioners. This is, therefore, one of the aims of this thesis: to present to EA researchers the benefits of adopting this paradigm, describing the SOA concepts, restrictions and methodologies, but also to present the most used technologies to chose the most adequate one.

In this chapter, the usage of SOA is proposed to facilitate the shortcomings in the EAs presented in the previous chapter: development, integration, standardization and dynamism. First, several concepts to undarsted SOA are explained. Then the most used technologies and methodologies are presented and how can help with the previous shortcomings. After that, the benefits of using SOA in EAs are explained, but also the restrictions of SOA are listed in order to be considered when creating a methodology to develop evolutionary algorithms in this paradigm.

4.1 WHAT IS A SERVICE?

A service can be seen as a function call which can be executed locally or remotely, and which is independent of the programming language or running platform. As previously said, services have well defined interfaces, which depends on the desired technology to implement SOA. That means that the service users do not need to know the language implementation of the service or the operating system, and they are not forced to use a specific technology to access to that service.

Figure 4.2 shows the basic interaction among services. First, the SERVICE PROVIDER exposes the service, publishing its interface in the SERVICE BROKER (or SERVICE REGISTRY). The SERVICE CONSUMER (or REQUESTOR) finds a service in the broker to be used and receives its interface. Then the request is performed by the CONSUMER (which uses or consumes the service).

According to Valipour [139], services must follow these characteristics:

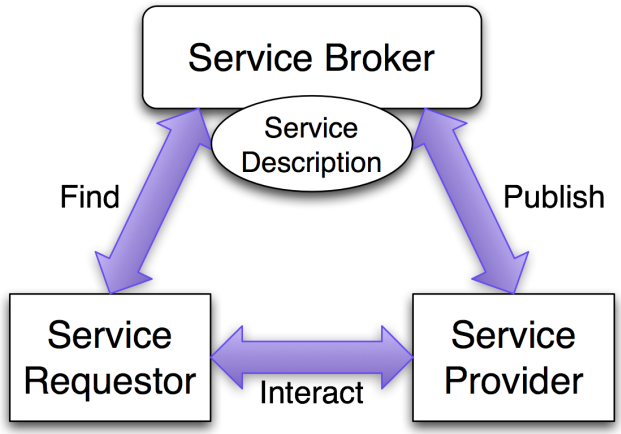


Figure 4.2 Service interaction schema. The service provider publishes a service description that is used by the consumer to find and use the service.

- *Discoverable and Dynamically Bound*: Services must be discoverable. Thanks to the service registry, a service consumer can discover a service to be use at runtime.
- *Self-Contained and Modular*: All functions in SOA are services. This means that every component in SOA must be modelled as a service, or as an aggregation of services. The services are well-defined: the interface of the service must be fixed, and it can not change in time, because the consumers or implementations of this interface should be modified with it. Services are, therefore, *encapsulated*: only the interface should be used to consume a service.
- *Interoperability*: Consumers do not need to know how the service implementation performs their function, as services behave as a "black box". This is, elements such as the programming language or distribution protocol are independent.
- *Loose Coupling*: Services should be designed to need only a few number of well-known dependencies.
- *Location Transparency*: Services must be indistinguishably local or remote, being independent of the protocol to establish the connection.
- *Composability*: Developing applications in SOA means to aggregate different existing services. Services are designed to be *re-usable*.

Moreover, several implementations of a specific service can exist (in one or several machines). The broker can choose which one

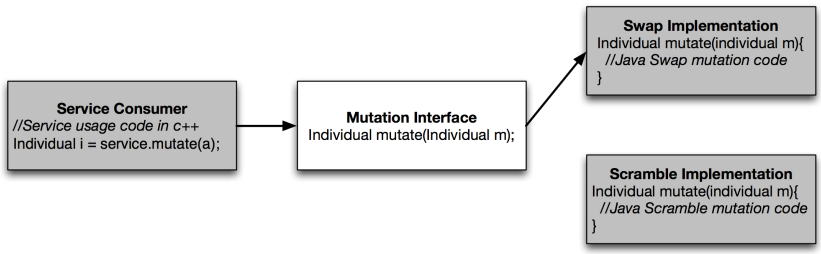


Figure 4.3 Example of usage of a service implementation.

to use each time, or offer another if a service is temporarily unavailable. Implementations may also have a different behaviour, so the researcher can take advantage to create an auto-adaptive algorithm to select different implementations according to some criteria. Figure 4.3 shows this special interaction, where two different implementations of an operator interface exist (even using different languages) and the broker has chosen one of them.

The service broker in a SOA can be implemented in several ways and have different behaviours: for example, the implementations of the services to be used can be defined in a text file (if the services do not change in execution time). However, the broker can also assign implementations to interfaces in an automatic way, or using several rules. For example, in the context of EAs, to select a better operator if the current one is not working properly.

An important SOA capability is that it is not focused on a specific implementation, but offers a set of guidelines to help the developers. In [11] these guidelines and good practices, and also the differences between SOA and Object Oriented Programming (OOP) are explained: the main difference between SOA and imperative programming or OOP is the order of service execution. This order is not necessarily static, because the services are designed to be used in a non-established and configurable order. Furthermore, another important difference is that services can be dynamically discovered and used (while in OOP a function/method must be previously known and can not change during execution), being also one of the most important capabilities the (optional) distribution in a network. Finally, in OOP the programming language must be the same for each method call.

4.2 IMPLEMENTATION TECHNOLOGIES

Despite the fact that the concept of service is independent of the technology used, there exist several ways to use and imple-

EBXML: *Electronic Business XML*
OSGI: *Open Service Gateway Initiative*

ment services, being Web Services, REST, EBXML and OSGI the most extended.

4.2.1 *Web Services*

One of most popular services implementations are WEB SERVICES [118]. A web service is a service available over Internet, that uses any standardized XML (eXtended Meta Language) [146] messaging system, and it is not tied to any specific language or operating system [19]. As SOA proposes, web services should be self-describing (using a standardized grammar) and self-discoverable.

Messaging system

There are several alternatives to the messaging system, as SOAP or XML-RPC. SOAP (Simple Object Access Protocol) is a standard protocol proposed by the W3C [15] which extends the XML remote procedure call (XML-RPC) standard. It is a complete and mature protocol that allows performing remote method calls to distributed routines (services) based on an XML interface.

SOAP clients can access objects and methods that are residing in remote servers, using a standard mechanism that makes the details of implementation transparent, such as the programming language of the routines, the operating system or the platform used by the provider of the service. At the moment, there exist complete implementations of SOAP for Perl, Java, Python, C++ and most modern languages. Unlike other remote procedure call methods, such as RMI (Remote method invocation, used by the Java language) or XML-RPC, SOAP has two main advantages: it can be used with any programming language, and it can use any type of transport (HTTP, TCP, SMTP and other protocols). In this way, SOAP constitutes a high level protocol, making easy the task of distributing objects among different servers, and avoiding the difficulties derived of defining the message formats, nor the explicit call to remote servers.

HTTP: *HyperText Transfer Protocol*
TCP: *Transmission Control Protocol*
SMTP: *Simple Mail Transfer Protocol*

Self-description

The interfaces of the methods of web services that can be accessed are specified by a Web Services Description Language (WSDL) [145]. The WSDL of a web service consists in an XML description of its interface, i.e., it is a file that describes the name of the methods, their parameters (number and type) and their type of response.

Self-discovery

UDDI (*Universal Description, Discovery and Integration*) [114] is a technical specification for describing, discovering and integrating web services [19]. This specification includes APIs for the storage and retrieval of information (also in an standardized XML format).

Other standardizations

One of the advantages of using web services is that the application stack is growing with the WS-Extensions. That is, the basic specifications of Web Services (such as SOAP) can be extended with transactions, security or messaging, for example. The most used are [118]:

- WS-Addressing (authentication)
- WS-Security , WS-SecureConversation and WS-Trust (authorization and secure messaging)
- WS-Policy and WS-Metadata Exchange (policy mechanisms for interactions)
- WS-Reliable Messaging and WS-Transaction (add-on mechanisms for the communication channel)

Also, functional extensions, such as WSRF [112], allows the discovery, inspection and interaction with stateful resources in standard and interoperable ways. Finally, BPEL (*Business Process Execution Language*) [84] is an XML-based language to control the invocation of different Web services with added business logic to help large-scale programming.

The main advantage of using Web Services in research is their public discovering and usage, thanks to the security extensions. Several studies about e-science taking advantage of web services can be found in bibliography [25, 99, 115, 122].

4.2.2 *REST*

Representational State Transfer (REST)[2] is an alternative method to build web services. This architectural style was proposed and defined by Fielding in [49].

In a REST-style architecture, a client sends requests to the server, who processes them and returns responses to the client. Requests and responses represent resources that can be addressed

2 http://en.wikipedia.org/wiki/Representational_state_transfer

by a Uniform resource identifier (URI). Usually, resources are documents or programs the client need to access.

REST usually works over the HTTP protocol. However it can be based on other protocols that provide the appropriate mechanisms to send requests and return responses.

In a REST environment, while servers are not concerned with the client state, clients only take about their own state and how to address resources on the server using URIs. Moreover, clients can cache responses to improve performance. As the client-server communication is stateless, servers are simpler and more scalable. Taking this into account, if the REST interface is not altered, servers and clients can be modified independently. Finally, servers can customize the functionality of the clients by sending their logic (code) to be executed.

REST web services are simple and lightweight (as no extra XML markup is needed), their message format is readable by humans, they are easy to build, and finally, developments achieve a high performance [23]. The main differentiating factor is that Web Services using SOAP tend to be operation-based, while REST services are resource-based. This is one of the reasons REST is replacing SOAP on the web [102].

4.2.3 *ebXML*

ebXML defines a set of standards that allows the enterprises negotiate their products through the Internet. It is based on a well-defined documents interchange using a contract-based approach [120], providing a specification for messaging, registry/repositories and business processes description, and unlike other approaches, it is an horizontal standard (it is not oriented towards a specific industry sector). On the contrary, Web Services expose any kind of applications to the Web, so anyone can call them (service approach). Another significant difference between Web Services and ebXML is that the former is based on BPEL, which can only describe the scenario inside a company, due to it has not all the information about the services being orchestrated, while the latter can be used to model a global choreography among several companies. Due to this, and because it is mainly focused to commercial and business processes, this technology is not going to be addressed in this thesis.

4.2.4 *OSGi*

OSGi was proposed by a consortium of more than eighty companies in order to develop an infrastructure for the deployment of services in heterogeneous network of devices, mainly oriented to domotics [57, 59]. Nowadays it defines a specification for a SOA for virtual machines (VMs). It provides very desirable features, like packet abstraction, life-cycle management, packaging or versioning, allowing a significant reduction of the building, support and deployment complexity of the applications. These features can be useful in the field of EAs, as suggested by Wagner *et al.* [143].

OSGi technology allows dynamic discovery of new components (or services), to increase the collaboration and to minimize and manage the coupling among modules. Moreover, the OSGi Alliance has developed several standard component interfaces for common usage patterns, like HTTP servers, configuration, logs, security, management or XML management among others, whose implementations can be obtained by third-parties.

These advantages are not so costly as can be thought: on one hand the OSGi framework can be implemented in a *jar* file[3] of about 300KB, and on the other hand, and differing from the normal usage of Java, each class pre-charges only the other classes it needs, not all. Also it is non-intrusive: the code to be executed in OSGi can be executed without it. Finally, from its specification in 1998 has been widely used as base in big projects: the Eclipse IDE is built over OSGi, and also big application servers (Glassfish[4] or IBM Websphere [5]) or residential gateways [57], among other examples.

IDE: *Integrated Development Environment*

In OSGi all services can be distributed using the OSGi features, simply setting which service is distributable and which is the distribution technology that provides service discovering and data transmission.

4.3 METHODOLOGIES FOR DEVELOPING SOA

Regardless of the chosen SOA framework, the processes of the platform must be analysed and modelled. So it is necessary to use a consistent and well-defined methodology to design a model based on a machine-readable description [58]. *Business-Centric Methodology (BCM) for Enterprise Agility and Interoperabil-*

3 A jar file is a file that groups some compiled Java files.
4 http://glassfish.java.net
5 http://www.ibm.com/software/websphere/

ity [113] is a roadmap for the development and implementation of procedures to create effective, efficient, and sustainable interoperability mechanisms. It has been developed by OASIS, the same consortium that created BPEL or UDDI, among others, and it is complementary to other existing architectures and technologies designed to build business oriented services, like ebXML or Web Services. BCM is formed by a set of model layers with a step-guide process, and an information pyramid to align the semantic information of partners. This allows the participation of business experts and the creation of a very large documentation repository. Nevertheless, this methodology has some disadvantages: it has a very large learning curve and it is not very extended yet.

UN/CEFACTs: *United Nations Center for Trade Facilitation and Electronic Business*
B2B: *Business to Business*

UN/CEFACTs *Modelling Methodology (UMM)* [77] is an approach to model the business services that each partner must provide in order to perform a B2B collaboration. It has a complete metamodel about business processes and business information, including a process analysis methodology. It is interesting to show that UMM provides and supports components to capture the knowledge about the business processes, and that it is independent of the underlying implementation technology (ebXML, Web Services, CORBA or EDI). Furthermore, because UMM extends UML, we could say that this methodology is more easily adaptable, due to the high development, acceptance and maturity of UML [58]. In fact, a survey of B2B modelling languages show that UMM is the most complete approach [50].

CORBA: *Common Object Request Broker Architecture*
EDI: *Electronic data interchange*
UML: *Universal Modelling Language*

Finally, *SOMA* (Service Oriented Modelling and Architecture) [11] is an architecture proposed by IBM to model service oriented processes. It lets the identification, specification and implementation of the services, flows and components inside the SOA paradigm. To achieve this tasks, it proposes a top-down modelling oriented to intra-enterprise services (service-oriented instead of business-oriented). It is more agile than the previous ones and it is not focused in enterprise environments.

4.4 BENEFITS OF USING SOA IN EVOLUTIONARY ALGORITHMS AREA

SOA has been previously used in the EA area. García-Nieto *et al.* proposed *Remote Optimization Service* [55], a client/server environment for launching different algorithms programmed in several languages. Although it uses XML and DTD to define inputs and outputs of the services, only the whole algorithm is exposed

as a service. Also, this system does not allow dynamic discovering or combination of available operators.

Web Services have been used in the grid area for optimization problems, as can be seen in the works of [21, 83, 133, 134], where services are defined using WSDL interfaces and other transmission mechanisms (such as Remote Procedure Call [76, 147]).

Although EAs are executed in grids [80, 96, 111]), no information about how to design these services for EAs has been provided in previous works.

In the previous chapter several shortcomings in the Evolutionary Algorithms area were presented, such as the new trends of distributed programming where nodes enter and exit in runtime, or the incompatibility between frameworks, for example. All these facts motivate the creation of a proper way to define services for evolutionary algorithms. The elements that combine an EA are candidate to be designed as services, as they can behave as input/output functions. Also, SOA solve the problems previously addressed:

- *Development*: there exist several methodologies to model and design services. Also, as services are re-usable, they can be combined in different ways to create the different types of EAs. Moreover, existing technologies, also facilitate the development, using techniques such as versioning, packaging or life-cycle control.

- *Integration*: Services are independent of the programming language. For example, services implemented in Java may use services implemented in C++ and vice-versa. Also, services allow distribution transparency: it is not mandatory to use a specific library for the distribution, or modify the code to adapt the existing operators. Existing EA frameworks could also be adapted to be accessed as services, providing their interfaces.

- *Standardization*: Interfaces of services use public standards (such as WSDL [145] or OSGi [117]). The service interfaces for EAs should be abstract enough to avoid their modification. Furthermore, as Foster claims [52], SOA is the key to develop Open Science.

- *Dynamism*: Services are not aware of the order of execution, so this paradigm can fit with new parallel approaches for EAs, where the control of the nodes is not centralized. Also, SOA provides techniques for automatic discovering of services. For example, new operators in different nodes can be

bound and used during the run of an algorithm. Also, there should be easy to add and remove elements to achieve self-adaptive mechanisms.

4.5 RESTRICTIONS IN SOA DESIGN FOR EAS

To allow these benefits the services for EAs should match with the next technological restrictions:

- The services can be dynamically bound to change the needed EA aspects.
- The source code of the basic EA services should not be re-written or re-compiled to achieve this task. That means that the design must be as abstract as possible.
- New services can be added in execution time.
- No specific source code for a distribution must added, neither the existing source code of the services should be modified for this purpose (that is, changing distribution libraries must not add extra code in existing services).

One of the main restrictions in SOA, apart from focusing on the development of abstract services, is the stateless and unordered nature of services. Therefore, services must follow the next guidelines.

First, as services are unaware of each other, there should not be global variables in any part of the code. Services are listening, and waiting to be executed. For example, a fitness service with a counter that is increased each time is called (to stop the algorithm if a limit is reached, for example). If several (and different) algorithms were working in parallel, and calling this function concurrently, the counter could not distinguish between algorithms, giving erroneous results. However, a service that maintains some kind of state is allowed, for example, a statistics service that reads events from all the algorithms being executed at the same time, but this should be managed to avoid errors.

Also, a service should not be distinguishable from local or remote running in other node in the network. Every stage in the algorithm should be treated as a service to be executed in local or in remote, even the *Population* or the *Parameters*. Mechanisms to ensure the correct data-sharing should be provided. Also, many implementations of the same service could exist at the same time (different implementations of *Crossover*, for example) and they should be correctly managed and used.

Moreover, a service is always a request-response function. For example, the fitness calculation should not be a method of the *Individual* implementation, but a function that receives a list of individuals and returns a list of the calculated fitness of that individuals. This allow, for example, remote fitness calculation and distributed load balancing, impossible to perform if the fitness is a method of the *Individual* class.

Thinking as abstract as possible requires to separate concepts such as the order of recombination, and the crossover itself. Usually, after parent selection, individuals are crossed in order. However, if we need a different mechanism for mating (for example, using more than two parents, or parent selected several times) a duplication of effort is needed. That is the reason to separate the concept *recombine* from *crossover* (and also, following the top-down design proposed in SOMA).

Finally, no assumptions should be taken about services previously executed or being executed next. For example, the *Mutation* service could be applied before the *Recombination* or the fitness could be calculated in the middle of the generation. Usually this step is performed in the last stage of the generation, but if we require the individuals for other tasks: for example, a Local Search or a statistics collector to guide the algorithm.

4.6 CONCLUSIONS

Even as SOA is used extensively in software development area, it is not widely accepted in the main EA software, as the survey of frameworks by Parejo et al., presented in Section 3.5 claim. The authors of these frameworks should improve their frameworks adding SOA technologies in order to facilitate the communication and integration among them, without duplication of effort to re-program all the EA elements, and therefore, saving time. Therefore, the benefits of using SOA in development, integration, standardization and dynamism (presented in Section 4.4) could be applied. Although all the approaches described Section 3.2 are focused on the implementation of distributed EAs, the abstraction level of each alternative can be quite different, as shown in Figure 4.4. As SOA is a paradigm and not a technology, areas such as Evolutionary Robotics, or EA classic frameworks can use SOA to be designed and developed. Implementation technologies, such as Web Services, can fill the gap between SOA (abstract) and grid (infrastructure) where interfaces are designed using SOA principles (dynamism, visibility, loose-coupling and heterogeneity). Finally, cloud computing can be seen as a com-

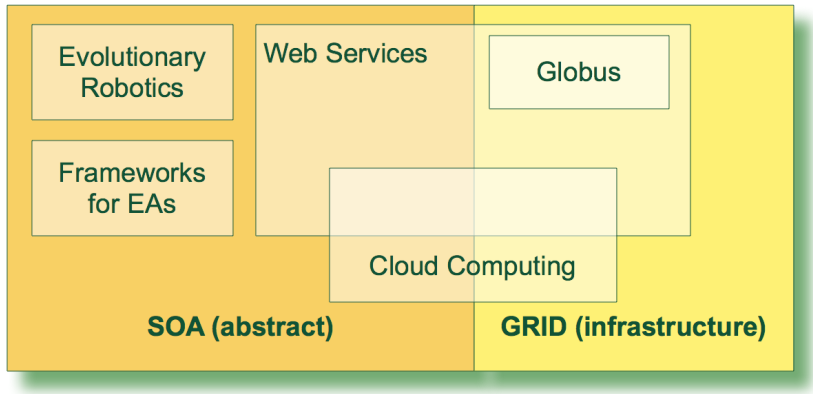


Figure 4.4
SOA as abstract paradigm to develop EAs in different areas. Using specific technologies such as Web Services allows grid integration. This figure has been updated from the one presented in [81].

bination that extends SOA adding the scalability of the grid, ad suggested by Jamil [81].

In this thesis the SOMA guidelines (identification, specification and realization of the services, flows and components) are going to be used, because it is the methodology more flexible and less focused on commercial purposes. Next chapter will present a methodology to use the design principles of SOA for developing services for EAs. Then, in later chapters, a specific SOA technology will be used to develop an implementation of a service oriented architecture for EAs.

5

A METHODOLOGY FOR DEVELOPING SERVICES FOR EAS

I don't claim to be a methodologist, but I act like one only because I do methodology to protect myself from crazy methodologists.

— Ward Cunningham (2004) Geek Noise

INDEX

SOA provides a good set of solutions to solve some of the problems in the EA area, such as the lack of integration, standardization and dynamism control, as presented in Section 3.5. It also allows ease of development in dynamic, distributed and heterogeneous systems, like the ones presented in Section 3.2.2.

SOMA methodology, presented in previous chapter (Section 4.3), establishes that the phases of the SOA design are *identification*, *specification*, *implementation* and *deployment* of the services and flows. Although SOMA is more focused on business environments (where other phases exist), the ideas that it offers are used to develop a methodology for the design of services for EAs. In this chapter the SOA-EA (Service Oriented Architecture for Evolutionary Algorithms) methodology is presented. SOA-EA is an abstract methodology to develop SERVICE ORIENTED

EVOLUTIONARY ALGORITHMS (SOEAs), that is, evolutionary algorithms whose elements are services, independently of the technology to be used. It is formed by several phases to identify the services that compose an EA and specify some of their possible behaviours, taking into account the restrictions presented in Section 4.5. This methodology will fulfil the OBJECTIVE 2 of this thesis: Propose a methodology that is able to successfully adapt evolutionary algorithms to distributed, heterogeneous, dynamic, standards-based environments.

5.1 STEPS FOR DESIGNING SERVICES FOR EAS

This section presents all the steps to design and implement SOEAs using SOA-EA. As in SOMA, the phases are not linear, but they are iterative and incremental, that is, the designer can move back to a previous step if necessary. For example, new services can be discovered during the specification phase or changes on the specification could appear in the deployment phase. Figure 5.1 shows the steps of the proposed methodology.

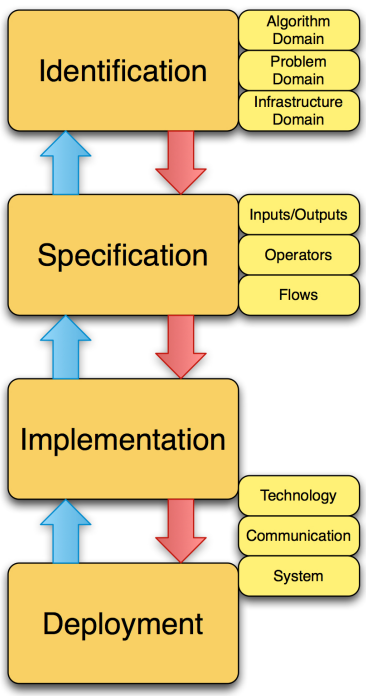


Figure 5.1 Methodology to develop services for Evolutionary Algorithms.

5.2 IDENTIFICATION

This phase is focused on the identification of the three constructs of SOA: services, components and flows. So, at the end of this step, the developers have a complete list of the services to be designed.

First, the developers should ask themselves the following questions to facilitate the identification:

- Which problem do I need to solve?
- What elements are needed by my EA?
- Has somebody else programmed this before?
- Which operators do I need?
- Is my algorithm going to be extended in the future?
- How can I parametrize my algorithm?

Solving the previous questions is the first step to identify the services. The next step is to classify the services in one of the three different domains that are proposed: *Algorithm domain*, the *Problem Domain* and the *Infrastructure Domain*.

5.2.1 *Algorithm domain*

Services in this domain are those that conform the EA. For example, operators of individuals, stop criterion, or populations. Also, getting the values of the parameters can also be a service, thus the EA developer obtains two advantages over using parameters only as variables: it is not mandatory to distribute the parameters among all services, and also they can be dynamically modified in execution time from an external service, facilitating self-adaptation.

5.2.2 *Problem domain*

In this domain, the user defines the services to address the elements of the problem. An example is the fitness function. The fitness function is a clear EA element that can be designed as a service. Each problem should implement an interface of the fitness service that receives the individual, allowing the distribution of this service (instead of being a method in the *individual* class, for example). There are also other services that depend on the problem, such as an initializer of individuals.

5.2.3 *Infrastructure domain*

Services in this domain are the ones that deal with the specific infrastructure that will be used to execute the algorithm. For example, services for user control, load balancing or logging. The design of many of these services is out of the scope of the EAs, but all them have to interact with the previous domains in some way. Depending on the environment where the EA is going to be developed, other services need to be modelled. For example, user control in cloud environments, different mechanisms for logging (in console, GUI...) or interconnection with other systems (such as external databases).

5.3 SPECIFICATION

Once the services have been identified, the next step of the methodology establishes the inputs and outputs of the services of the SOEA. The questions to solve prior to this phase are:

- Which are the inputs of the services?
- Which are the operations of each service?
- How are the individuals representation?
- How are the services going to be used?
- Which is the order of execution of the services?
- Is only one type of service required?
- Are the services going to be adapted to computational power of the machines?

All the characteristics of genericity for the design of an EAs, presented in Section 3.4.1, should be taken into account when designing elements for EAs. However, requirements are also aligned with the requirements for designing services, explained in previous chapter (Section 4.5). It is important to remark that in the future these services could be extended, so they should be designed taking into account this possibility.

5.3.1 *Specifying the operators*

When specifying operators (such as *recombinator* or *mutator*) they do not have to be modelled to receive one or two individuals,

since not all EAs have the same behaviour. They should receive a list of individuals to be crossed or mutated each generation. Almost all services in an EA (like *mutation* or *selection*) will accept individuals as input data and produce/modify these individuals. Due to many kind of individuals may exist, the operators should be as abstract as possible to work properly. Therefore, services must accept interfaces of individuals as inputs, not concrete implementations, such as vectors or lists (generic representation).

5.3.2 *Specifying the population*

The population should not be a list of individuals: it should be a service to access the individuals and allow the variation of its structure (for example, a change from an unique list population to a cellular model) without affecting the rest of the services of the algorithm. So, other services external to the EA could consult the *population* state and act accordingly to some rules.

5.3.3 *Specifying the fitness*

As previously stated in Section 4.5, the fitness should not be calculated within a method of an *Individual* class. To be less coupled, it should be implemented as an external service that receives a list of individuals (facilitating the load balancing). That way, the service is as abstract as possible.

5.3.4 *Specifying the parameters*

The parameter set should be a service for the same reason, allowing the possibility of performing experiments related to parameter control or tuning [35] in an efficient way (being separated from the code of the existing operators).

5.3.5 *Specifying the flow of the services*

A SOEA can be seen as a service flow. Flows should be designed to reduce the impact of potential future changes. An example of service flow would be an implementation called *Evolutionary Algorithm* with all the steps common to all EAs and with independence of the implementations of these steps (generic evolutionary model). This allow the adaptation of the evolutionary model. The user can manually select the services to be combined to create a Genetic Algorithm or an Evolution Strategy, for example.

Furthermore, to accomplish with the genericity presented in Section 3.4.1, the parameters and operators should be added dynamically. This is done with the SOA service binding. Users can specify the operators they need in several ways, for example, in a configuration file, or in an intelligent manner (an algorithm), taking into account the classification about evidence in parameter control presented in Section 3.3 . It is important to remark that these "pieces" do not need to be modified and compiled again, because the loose coupling and the dynamic binding of SOA. Without SOA this behaviour is very difficult to achieve or maintain, as will be explained in next section.

Also, mechanisms to allow adaptation of computational power to the machines should be addressed in this step. As this new paradigm may deal with dynamism and heterogeneity of the resources, services to control these issues need to be defined: for example, adapting parameters values or different operators depending some metric or benchmark, or to control nodes that disappear during running time.

5.3.6 *Specifying the infrastructure services*

The infrastructure services function is to manage the system. For example, output mechanisms (GUI or logging services) that should be independent of the services of the other domains. In this step, also the control of the system should be described (for example, user control or load balancing control).

5.4 IMPLEMENTATION AND DEPLOYMENT

Once the services have been identified and specified, a SOA technology should be used to implement and publish them. These two steps of the methodology are explained together because the decisions about the technological solution to be used is bound to both phases.

The questions to solve in these steps are:

- Are services going to be used locally or remotely?
- How the interfaces are going to be exposed?
- Are the services public?
- How are going the changes in service dynamism to be managed?
- How must be the overload of the messages?

- Which are the advantages of the chosen technology?
- Which are the specific considerations about security, persistence, benchmarking and monitoring?

5.4.1 *Select the technology to expose the interface*

As presented in Section 4.2, there exist several technologies to implement services. Depending on the use of the services, one technology should be chosen over other. For example, a service that is going to be used remotely and publicly from any programming language should export its interface with WSDL publicly available with an URL, to allow users to automatically generate the client for that service. On the other side, interfaces could be previously known, and it is not necessary to export them to the public. This is the case of OSGi, where the interface is exposed only to the OSGi service registry. Other mechanisms could be used to publish and share the interfaces of the services (for example, using a *newcast* protocol).

5.4.2 *Select the communication mechanism*

Services are also independent of the transmission mechanism, so this issue must be considered depending on the system to deploy the services. In the case of EAs, where the performance is important, usually the most efficient transmission mechanism should be preferred. However, sometimes other transmission mechanism can be used. For example, SOAP (explained in Section 4.2), includes extra information in headers [17], producing more network overload. However, this information is easier to manage for other systems, or easier to configure to be used remotely (as it uses a standard HTTP port).

5.4.3 *Deploy in the system*

Once the services have been implemented they have to be deployed in the desired system. Examples of environments to deploy services are application servers (such as *Apache Tomcat*[1] or *Oracle GlassFish*[2]), service containers (*Equinox*[3] or *Felix*[4]), BPEL engines (for example *OpenESB*[5]) or as a stand-alone system. In

1 http://tomcat.apache.org/
2 https://glassfish.java.net/
3 http://www.eclipse.org/equinox/
4 http://felix.apache.org/
5 http://www.open-esb.net/

this step, issues related with testing, user control, security and persistence should be taken into account. The implementations of the infrastructure services deal with the chosen system.

5.5 VALIDATION OF THE SERVICES

As previously remarked, during all the steps of the methodology it must be validated if the created services for EAs accomplish the requirements of the development of services (Section 4.5) with the genericity of developing EAs (Section 3.4.1). Therefore, at the end of the application of the methodology, all services must accomplish next restrictions:

- All elements of the EA should be designed as loose-coupled, stateless services.
- The services for EAs should operate with independence of the structure of the individuals (generic representation).
- The operators of the EAs should be designed to be used in conjunction with others (for example, aggregation) and have minimal side effects (generic operations).
- The services for EAs should work with independence of the evolutionary model (generic evolutionary model).
- Services must be discoverable and dynamically bound.
- Services must provide a standard-based interface.
- Services can be added or removed in execution time.
- Services must be indistinguishable of being executed locally or remotely.
- No specific code should be added in the implementation of the services to specify the distribution mechanism.
- In relation with previous requirement, the distribution mechanism can be modified without affect the existent code.

5.6 CONCLUSIONS

Chapter 3 presented several shortcomings in the Evolutionary Algorithms area, such as the incompatibility between frameworks or how to handle with new trends of distributed programming, where nodes enter and exit in runtime, for example. All

Table 5.1 Summary of migration from traditional EA programming to SOA

Element	Current EAs development	Using SOA	Reason to migrate
Programming language	Just one for all elements of the algorithm	Any	Services are independent of the programming language. Only the interface is required to use services
Operators	Methods or functions	Services	Services allow the selection of a specific implementation during the algorithm execution, and also different programming languages or distribution models
Operators behaviour	Methods applied to a single individual	Services that receive individual lists	It allows load balancing and distribution, and also to modify the operators in execution time
Operator selection	Modifying the source code	In a flexible way outside the source code	It is not mandatory to recompile the source code to integrate new operators
Fitness	Method that evaluates an individual	Service that evaluates an individual list	It allows the distribution, load balancing and addition of new fitness calculators in real time
Population	Array or individual list	Population service	It allows to change the population type and topography, by selecting the service implementation
Self-adaptation	Modifying source code for a specific experiment	Self-adapting service that selects specific operator implementations	It does not modify the created services and brings more flexibility in the dynamic adaptation
Distribution	Libraries like MPI	SOA mechanisms	SOA technologies allow changing the transmission protocol and using extra technologies without adding extra code

these facts motivate the creation of a proper way to define service oriented evolutionary algorithms (SOEAs) to facilitate the development, integration, standardization and dynamism.

In this chapter the requirements in EA design (genericity in representation, fitness, operations, model, parameters and output) presented in Section 3.4.1, with the requirements in SOA (genericity in interfaces, language independence, distribution and dynamism) explained in Section 4.5, have been taken into account to propose a methodology to model the services that compose a service oriented EA, and several guidelines about the design of these services have been explained. This methodology proposes 4 iteratively and incremental phases: identification, specification, implementation and deployment. A number of questions has been proposed to be answered in each phase to help in the development and validation of the created elements. This methodology can been used to create a service-oriented evolutionary algorithm that takes advantage of the SOA capabilities, such as loose-coupled services and automatic binding of new operators.

Table 5.1 shows the advantages to design the elements of the EA as services.

Next chapter will present a complete example of development and will explain how to modify services to change from a model to another, adding transparent distribution and load-balancing or dynamic adaptation of the parameters.

Part III

EXPERIMENTAL RESULTS

6

DEVELOPMENT OF A SERVICE ORIENTED ARCHITECTURE FOR EVOLUTIONARY ALGORITHMS

If the river is taken, if the garrison at Osgiliath falls,
the last defence of this city will be gone

— Gandalf The White. The Lord of the Rings.

INDEX

Previous chapter has presented SOA-EA, a methodology to develop SOEAs. This chapter validates the use of SOA-EA to create services using the steps proposed. Identification and specification of a complete example of a SOA will be presented, solving the questions to identify a number of services and their behaviour.

Then, for the implementation and deployment steps, a specific SOA technology (OSGi) will be used to implement and deploy all the services and examples shown in previous sections, and how to accomplish the requirements in the development of EAs and SOA, taking advantage of the capabilities of SOA. As this

is an iterative and and incremental methodology, new services can be discovered or removed in further steps (for example, infrastructure services). The result will be a framework to facilitate the development of SOEAs, called OSGiLiath (*OSGi Laboratory for Implementation and Testing of metaHeuristics*). Finally, several experiment to demonstrate the ease in integration and development will be carried out to fulfil the **Objective 3** of this thesis: validate the methodology creating a framework using a SOA technology that solves the problems addressed.

6.1 EXAMPLE OF CREATING A SERVICE ORIENTED EVOLUTIONARY ALGORITHM

In this example a basic SOEA will be designed. Then, to illustrate the iterative process of the proposed methodology and the capabilities of using SOA this example will be extended. First, a NSGA-II algorithm [27] will be also designed using the existing services and adding new ones. Finally, new services to add distribution and self-adaptation will be developed.

6.1.1 *Identification*

As stated in in Section 3.1, a basic EA is formed by several steps. Solving the questions in Section 5.2 and the considerations about the design of services (Section 4.5) and the genericity of EAs (Section 3.4.1) a number of abstract services have been identified. In the *algorithm domain*, the Algorithm, Population, Parent Selector, Recombinator, Mutator, Mutation, Replacer, Stop Criterion and Parameters. In the *problem domain*, the Fitness Calculator and Initializer. Finally, in the *infrastructure domain*, a Launcher service to start the algorithm.

6.1.2 *Specification*

Concrete implementations are defined in this step: for example, *N Tournament* and *Roulette* are implementations of parent selectors and *Optimum Found* the desired stop criterion. Also, to address the problem to be solved, implementations of the Problem domain should be created: services such as *OneMax Fitness Calculator* or *Binary Initializer*. The Initializer is the service that establish the representation of the individuals (for example, a list of elements or a tree), depending of the problem to solve. In

this case, a *ListIndividual* implementation of *Individual* interface is created.

As we need a fixed set of steps, a *Evolutionary Algorithm* service is created to model the flow of services. The discovered services in previous step have been specified to accept a list of individuals.

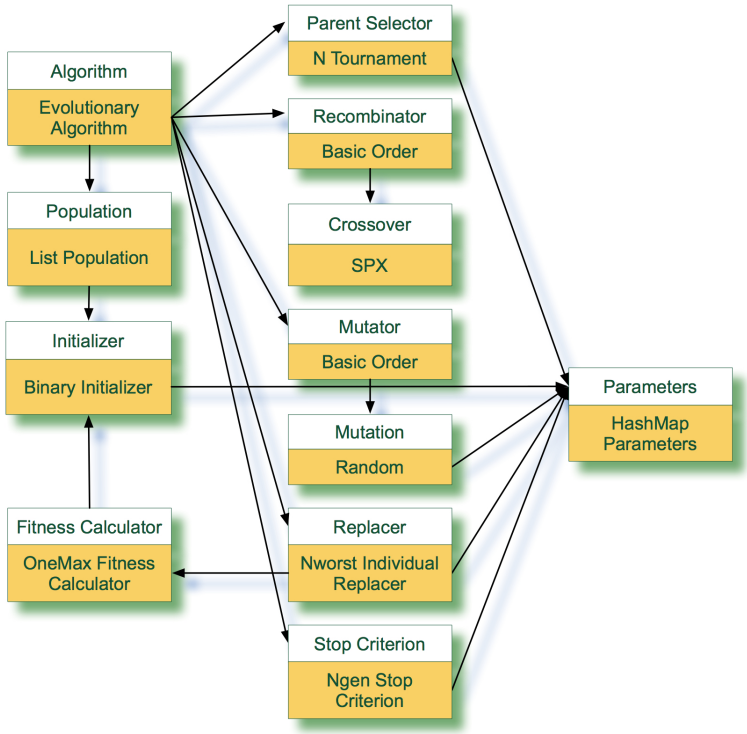


Figure 6.1 Diagram of a basic genetic algorithm. White blocks are interfaces and orange blocks are implementations. In this case, we are using specific implementations to solve the OneMax problem.

Figure 6.1 shows the diagram of a complete service oriented genetic algorithm, taking into account the proposed ideas. In this figure (and in the following ones) white blocks are the service interfaces. Orange blocks are specific implementations of these interfaces (that is, the source-code of the service), and arrows indicate how a service implementation can make use of other services via their interface. For example, almost all implementations access to the *Parameters* service using its interface. Service implementations (orange blocks) can be selected in a configuration file or be automatically bound when they are available (among other options).

The change from a problem instance to another is quite simple. It is only necessary to notify the algorithm a change in the implementation of the service *Fitness Calculator*. Because some algorithms need to calculate the fitness every time an individual is modified (and not only at the end of a generation) the service *Fitness Calculator* may be used inside the implementations that mod-

ify individuals (*Initializer*, *Mutator* or *Recombinator*). This should be considered in this stage, but could be changed dynamically. Moreover, each service can be in the local machine or distributed on the Internet, having the same behaviour.

6.1.3 *Extending the example to create a NSGA-II*

As this is an iterative and incremental approach, other services can be discovered and designed in this step. For example, the difference between the previous version of a GA and the well known NSGA-II lies in the selection operator. Therefore, to change from the basic GA to NSGA-II, the mutator and crossover are kept and new selection operators are added. Figure 6.2 shows the diagram of the service oriented version of NSGA-II algorithm, where the new implementations are marked with a thick border. The problem has also been changed to Multi-Objective Knacksack problem [148]. New auxiliary services have been added, like *Crowding Distance Assignator* or *Pareto Assignator*. As these services may be used in other algorithms in the future, they should be designed as abstract as possible. These new services are called from the implementation (code) of the services *NSGA-II Replacer* or *Binary Crowding Distance Selector* (black arrows indicate an interface call).

6.1.4 *Extending the example to add distribution*

As every service must keep the same behaviour, independently of the machine that hosts it, distribution services for load balancing of a specific service can be easily created. For example, notifying the algorithm to use a distributed implementation for that service, instead a local one. As previously stated, the service *Fitness Calculator* receives a list of individuals to calculate their fitness, so, in this example, the new fitness implementation (*Basic Fitness Distributor*) binds with every fitness service available (in the same machine or in a network). The source code of this basic implementation simply distributes the list of individuals among the bound services and waits for their termination. Although more complex implementations probably will be more efficient, the objective of this section is to show how to distribute services, thus, this basic implementation is sufficient. Figure 6.3 shows the modification from a sequential fitness calculator to a distributed one. Thanks to SOA, the number of distributed fitness calculators is not fixed: calculators can be added or removed in real time without stopping the system. As can be seen in the figure,

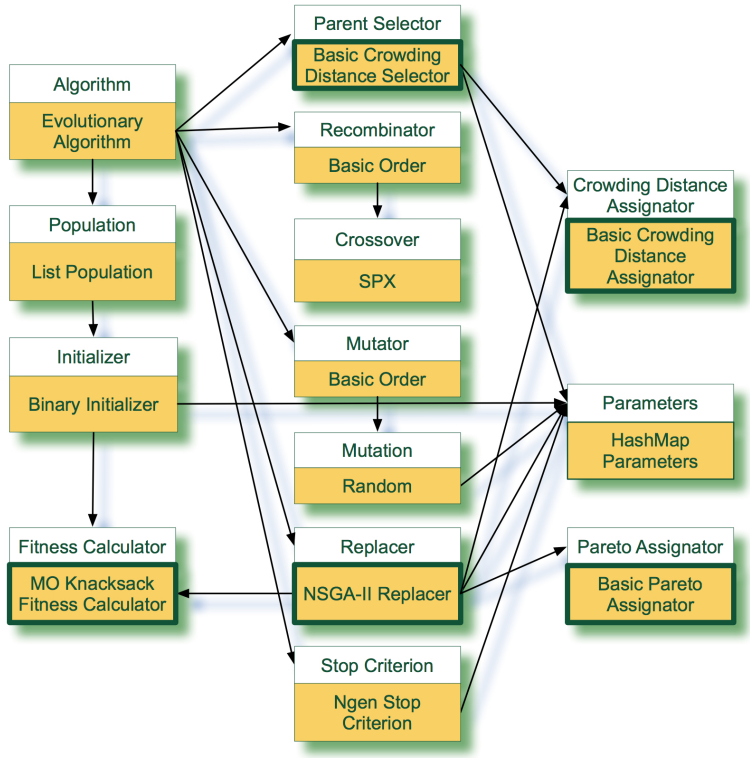


Figure 6.2 Modification of the basic GA adding new service implementations (orange blocks with thick lines).

if one of the nodes is a cluster, it could also implement another fitness distributor (block with thick lines in the Figure). This easy example can be adapted to more complex necessities depending on the infrastructure or the problem to be solved. More complex distribution services can be created, for example, taking into account communication latencies or computation capabilities of the nodes.

As explained in Section 3.2 other way to parallelize EAs is the island model. Using SOA-EA, the *Population* service implementation can be modified to become a distributed population. Each certain time, this population could exchange individuals with other populations modified by other algorithms. These populations should be added or deleted in execution time without affecting the algorithm execution. Figure 6.4 shows this example, where a *Replacer* implementation maintains a list of references to other *Population* interfaces (which can be local or remote). If one of these population services drop, the others can continue working. The topology of these islands can also be managed from services (such as *Basic Replacer* service, or another). The modification and dynamism of the population structure is difficult to apply in existing frameworks without using SOA because it is necessary to create mechanisms to modify the population be-

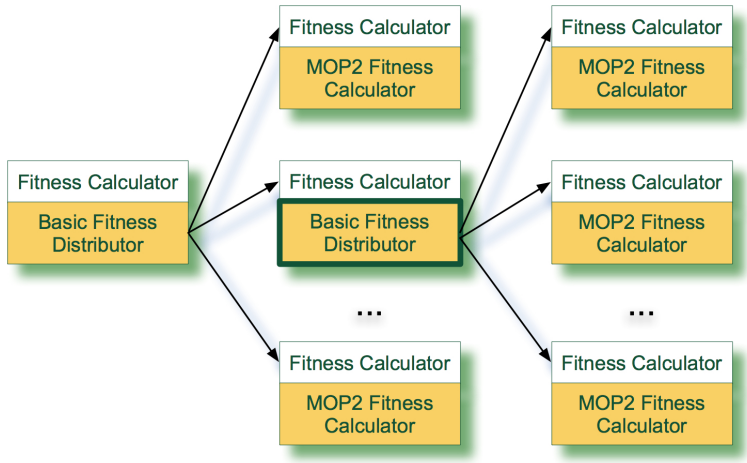


Figure 6.3
Fitness distributor. The thick line implementation also re-distribute the individuals.

haviour, the operators to modify it, the data structures, and also the code to manage all. With the usage of SOA, and due to the capability of accessing to a population via its service interface, it is not necessary to modify the source code to modify the population and its behaviour. Also, to avoid bottlenecks in distributed executions, asynchronous communication must be provided to avoid idle time. This kind of communication offers excellent performance when working with different nodes and operating systems, as demonstrated by [5].

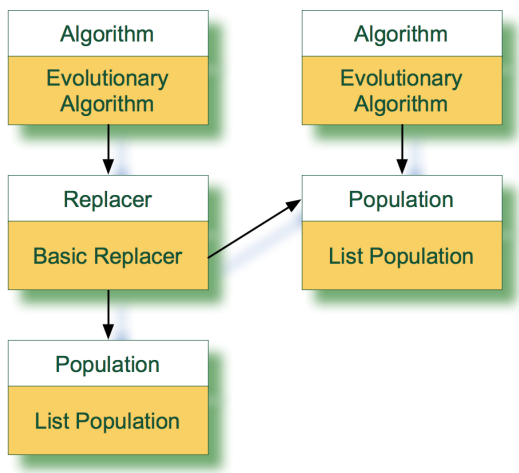


Figure 6.4
Island model. From time to time, the Basic Replacer Implementation could send or receive individuals from other islands.

6.1.5 *Self-adaptation of the services*

There are several ways to create self-adaptable algorithms using SOA. For example, creating a service that modifies the parameters in the *Parameters* service, or activating and de-activating

operators in real time. An easier way is to create a service that manages all available services of the same kind. For example a *Mutator* service that binds all the available mutation implementations and use the most adequate one depending on some rules during the execution [130]. This idea can also be extended to create a service that implements several interfaces and selects the most adequate implementation for each interface respect to some criteria, as can be seen in Figure 6.5, where thick lines represent the implementations used at the current moment (they vary as time passes).

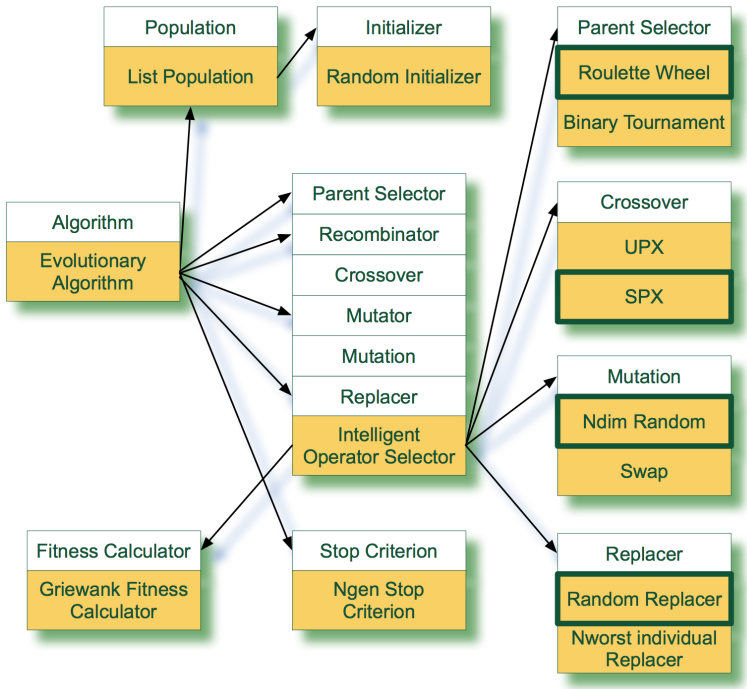


Figure 6.5 Self-adaptable Algorithm. The *Intelligent Operator Selector* selects which service implementation is used each time.

Finally, another important usage of EAs is its hybridization with other metaheuristics, to obtain more effective search algorithms [124], increasing the performance of intensification and diversification mechanisms. With traditional frameworks this task can be difficult, mainly because the source code for each metaheuristic must be modified. Nevertheless, using SOA a combination of loosely coupled services could be used.

Summarizing, the questions presented in Section 5.4 have been answered to obtain the next restrictions for the desired framework. Initially, the services can be executed locally, they must be dynamically bound and no extra code should be added to allow the distribution of the created implementations. This implementation should allow asynchronous data sending/receiving, without the need to implement specific functions in the source code,

like MPI or other distribution mechanisms. EAs developers can use the existing distribution services or create new ones, if they want. New improvements can be added without modifying the existing modules, that is, adding or modifying only the affected service implementations without modifying the source code of the other services.

6.2 IMPLEMENTATION AND DEPLOYMENT

In this section, the previous examples are going to be implemented and deployed. The aim of this section is to create OSGiLiath, a framework to facilitate the creation of SOEAs, taking into account the benefits of SOA, to provide a number of interfaces to implement and mechanisms for the dynamic binding of services. The previous examples will be implemented and added to the framework as a base for new developments and creation of new services for the rest of this thesis. First, a SOA technology will be selected, analysing their benefits and shortcomings. Then, the steps to implement and deploy the services using the chosen technology are explained.

6.2.1 *Select the technology to use*

Of the existing technologies for SOA (presented in Section 4.2) Web Services and OSGi have been considered to compliment with the restrictions previously addressed.

OSGi has been selected because it is faster than Web Services, because it was designed for lightweight devices [97]. Therefore, it can be used in embedded devices, like Evolutionary Robotics [56]. On the contrary, as explained in Section 4.2 Web Services were created to integrate complex data interchange among different companies. This is related with the transmission protocol in Web Services, SOAP, which implies the transmission of an XML, as explained in Section 4.2.1. This file is usually too large (for example, a complete list of workers in a company). EAs often need to send minimal information, but a large number of times (for example, the fitness of several individuals), so a complex transmission protocol is not recommended. OSGi includes many mechanisms for data transmission, allowing more flexibility depending on the execution environment of the algorithms (for example, in a machine, in a local network, over the Internet, or even in more lightweight devices).

Unlike Web Services, OSGi includes a blackboard event-manager, that is, services can inform what they are doing without

indicating any receiver. Other services can filter this information and actuate accordingly, so the synchronization is easier. For example, it is not mandatory to create a variable to count the number of times that the *Fitness Calculator* service is executed: an external service can track this number.

Other reason to use OSGi is the separation between OSGi and the source code of the services, so the code of OSGi-based applications can be used in other Java-based applications without OSGi. For the same reason, frameworks written in Java can be migrated into services easily.

Finally, OSGi includes other features that, although they are not related to SOA, facilitate the service development and deployment: version and package control, security and life-cycle management of the used components, that can be useful in the development of EAs (as explained by Wagner *et al.* [143]). These advantages can be used by the EA developers if they work in a team collaboration.

To distribute the OSGi services, the OSGi 4.2 Remote Services standard[1] is proposed. The *Eclipse Communication Framework*[2] (ECF) has been chosen because it is the most mature and accepted implementation of this standard (claimed by Petzold *et al.* [123]), and it also supports the largest number of transmission protocols, including both synchronous and asynchronous communication.

ECF includes a number of protocols for service discovery and service providers:

- Service Discovery API: Includes protocols to announce and discover remote services: Zeroconf, SLP/RFC 2608, Zookeeper, file-based and others [3].
- Remote Service API: Includes protocols to establish the communication (data streams, formats and others): R-OSGi, ActiveMQ/JMS, REST, SOAP, XMPP, ECF Generic [4]. This allows communication with systems that do not use OSGi or Java.

More information about the application of OSGi in other areas, with good practices, benefits and lessons learned is provided in [57].

1 http://www.osgi.org/Release4/Download
2 http://www.eclipse.org/ecf/
3 http://wiki.eclipse.org/ECF_API_Docs#Discovery_API
4 http://wiki.eclipse.org/ECF_API_Docs#Remote_Services_API

6.2.2 *Implementing the services*

In OSGi, the services are formed by the next elements:

- *Service interface*. It is a Java interface. The user just needs to specify the operations that the service will perform.
- *Service implementation*. The programmer just writes the code of the interface methods.
- *Service description*. It is an XML file that indicates which interface is being implemented and which other services need to be activated.

These elements accomplish with the restrictions of the separation between interfaces and implementations, explained in Section 4.5. More information about these concepts are explained in detail in Appendix A.

In this step the interfaces of the services identified in Section 6.1, such as *Algorithm*, *StopCriterion*, *Population* or *Recombinator* are implemented in Java. Concrete examples are *TPXCrossover* or *List Population*. These interfaces are grouped along other interfaces that do not need to be a service. For example, the interface of the object *Individual*. This interface is used in the *Recombinator* interface, which receives a list of *Individual* objects to be recombined, and returns another list with the recombined ones. Also, several implementations are included, such as the rest of services explained in previous sections, like the services for NSGA-II.

Once the services are implemented, the flow of execution must be implemented. In previous chapter the usage of a Evolutionary Algorithm service implementation was proposed (Section 5.3.5). The source code of the method that executes the algorithm in the class *EvolutionaryAlgorithm* (implementation) is shown in Figure 6.6. It includes methods to bind the six references to the service implementations that are needed: `Population` (*pop* in the code), `StopCriterion`, `ParentSelector`, `Recombinator`, `Mutator`, and `Replacer`.

6.2.3 *Deploying the services*

The services are deployed inside an OSGi container (such as Equinox[5] or Felix [6]). The container has mechanisms to list, start and stop services. To automatically bind the service implementations with the service interfaces the *Service Description* is used.

5 http://www.eclipse.org/equinox/
6 https://felix.apache.org/

Each implementation of a service has an XML file indicating which interface is being implemented, and also other properties. This file is used by OSGi to automatically bind the services. The service descriptor of the EA is shown in Figure 6.7. This file describes that the `EvolutionaryAlgorithm` class is an implementation of the `Algorithm` interface, and that it needs implementations of the interfaces `Population`, `Mutator`, `ParentSelector`, `Replacer`, `StopCriterion` and `Recombinator` to be activated.

It should be noted that this file usually can be modified using a friendly GUI, or from an assistant in Java IDEs, such as NetBeans or Eclipse (so, users do not have to care about its XML structure). The user interface to create this file in Eclipse is shown in Figure 6.8. The interface being implemented is set in the lower part (*Algorithm*). The necessary services to activate this implementation are indicated in the upper part (with the cardinality and functions to set and unset the service implementations in the implementation source code).

This XML file is read by the OSGi execution environment, which is the responsible to bind the available services to this implementation. For example, if a *ParentSelector* is activated, it is automatically bound to the variable `parentSelector` through the function `setParentSelector`. The `cardinality` (explained in Appendix A) is also set in the file, in this case, only one implementation is necessary (not multiple). This file can be modified in execution time, so it is not required to re-compile the Java code to use and set new services.

In brief, each implementation of a service (<`implementation`>) indicates the interface to being implemented (<`provide interface`>), and the other services this implementation needs (<`reference`>).

Moreover, each service can provide properties to be used by other services to obtain more information and filtering. For example, in this case only the *Replacers* whose property `replacerName = nsga2` are used.

6.2.4 *Managing services: implementing the NSGA-II from the canonical GA*

Following the development example shown in Section 6.1.3, the extra services have been developed to convert the basic GA into a NSGA-II and new implementations also have been added to OSGiLiath to be available for users.

There exist many options for the implementation of the EA to pick up the appropriate service. The first of them is modifying the source code of the implementations. Obviously this is

not recommended, because the service would not be loosely coupled due to the specific OSGi code, and this is not a good SOA practice. The following ways makes the service usage not code-dependent:

- De-activating the implementation *Binary Tournament* from the OSGi administration console, and activating the implementation *Crowding Distance Selector* (that is, manually). This technique is not recommended, because all services are then managed by hand, and this is very difficult with a large number of services. However, the OSGi console allows modifying services in execution time, so it can be used in some cases (for example, to stop the service in a machine while another big task is being executed, and activate it again when this task is over).

- Modifying the Service Descriptor of the *Evolutionary Algorithm* implementation to filter the desired implementations (for example, the attribute `target="(selectorName=nsga2)"` in Figure 6.7). This option is used when the algorithm is fixed and does not need to be modified in execution time, and the number of operators and types are known in advance. However, as previously stated, new services can be added in execution time (for example, if the cardinality is set to multiple).

- Using an external service that activates or de-activates desired implementations or modify their status. This technique must be used when self-adaptation properties are used in the algorithm, and it is presented in next subsections.

None of these options needs to modify the source code of the existing services: they just indicates which services uses each time.

6.2.5 *Making it distributed*

As previously stated in Section 4.5, services should be undistinguishable of being local or remote, and should not add extra code for distribution. Therefore, all services can be distributed using the OSGi features. In this case, the distribution is performed using the service descriptor to set which service is distributable and which is the distribution technology that provides service discovering and data transmission.

As explained in Section 6.2.1, OSGi allows several implementations for the service distribution. This specification uses the

OSGi service registry to expose remote services to other machines (being indistinguishable from the local ones). ECF also separates the source code from the discovery and transmission mechanism, allowing users to apply the most adequate technology to their needs, and providing the integration with existing applications. For example, the lines of Figure 6.9 can been added to any service descriptor to distribute it in the local network.

In this case, it is only necessary to set the properties that ECF uses to identify the services being distributed in the network, indicating that all implemented interfaces are distributable (`service.exported.interfaces`). Also, the communication technology to be used is established (`ecf.generic.server`, although another kind of protocol could be used), and finally, the service URL (`ecf.exported.containerfactoryargs`). As previously stated, the service properties can be modified from other services, so these properties can be added outside the XML. It should be noted that the source code of the services has not been modified to distribute them (as would happen if MPI, or other middleware, had been used to perform the distribution, for example).

Figure 6.6 Java code of the class *Evolutionary Algorithm*. This class implements the *Algorithm* interface, which defines the operation *start()*

```
//References to the implementations to use
Population pop;
ParentSelector parentSelector;
Recombinator recombinator;
Mutator mutator;
Replacer replacer;

//Example of the method to obtain an implementation
//of the ParentSelector interface
//(one function per reference)
void setParentSelector(ParentSelector sel){
  this.parentSelector = sel;
  //now sel is a reference to an implementation
        //of ParentSelector
}

//Implementation of the start() method of the
//Algorithm interface
public void start(){
  pop.initializePopulation();
  actualIteration = 0;
  do{
    //SELECT parents
    List<Individual> parents =
    parentSelector.select(pop);

    //RECOMBINE parents
    List<Individual> offspring =
    recombinator.recombine(parents);

    //MUTATE offspring
    List mutatedOffspring =
    mutator.mutate(offspring);

    //SELECT new population.
    //pop is modified here
    replacer.select(pop, parents,
    offspring, mutatedOffspring);

    actualIteration++;

  }while(!stopCriterion.hasFinished());

}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
enabled="false" immediate="true" name="OsgiliathEvolutionary" >
<implementation
class="es.ugr.osgiliath.evolutionary.EvolutionaryAlgorithm"/>
<service>
<provide interface="es.ugr.osgiliath.algorithms.Algorithm"/>
</service>
<reference bind="setPopulation" cardinality="1..1"
interface="es.ugr.osgiliath.evolutionary.elements.Population"
name="Population" policy="static" unbind="unsetPopulation"/>
<reference bind="setMutator" cardinality="1..1"
interface="es.ugr.osgiliath.evolutionary.elements.Mutator"
name="Mutator" policy="static" unbind="unsetMutator"/>
<reference bind="setParentSelector" cardinality="1..1"
interface="es.ugr.osgiliath.evolutionary.elements.ParentSelector"
name="ParentSelector" policy="static" target="(selectorName=nsga2)"
unbind="unsetParentSelector"/>
<reference bind="setReplacer" cardinality="1..1"
interface="es.ugr.osgiliath.evolutionary.elements.Replacer"
name="Replacer" policy="static" target="(replacerName=nsga2)"
unbind="unsetReplacer"/>
<reference bind="setStopCriterion" cardinality="1..1"
interface="es.ugr.osgiliath.evolutionary.elements.StopCriterion"
name="StopCriterion" policy="static" unbind="unsetStopCriterion"/>
<reference bind="setRecombinator" cardinality="1..1"
interface="es.ugr.osgiliath.evolutionary.elements.Recombinator"
name="Recombinator" policy="static" unbind="unsetRecombinator"/>
<property name="algorithmName" type="String"
value="EvolutionaryAlgorithm"/>
</scr:component>
```

Figure 6.7 Service descriptor of the Evolutionary Algorithm implementation. Figure 6.8 shows the friendly user interface to automatically create this file using the Eclipse program

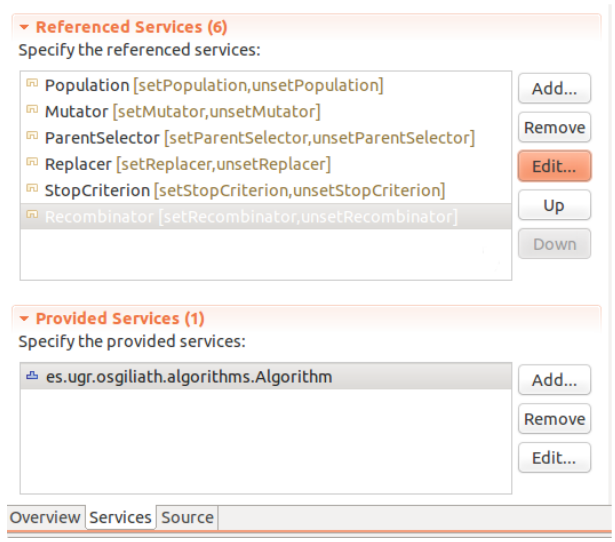


Figure 6.8 Graphic user interface in Eclipse that generates the Service Descriptor of Figure 6.7

```
<property name="service.exported.interfaces" type="String" value="*"/>
<property name="service.exported.configs" type="String"
value="ecf.generic.server"/>
<property name="ecf.exported.containerfactoryargs" type="String"
value="ecftcp://localhost:3787/server"/>
```

Figure 6.9 Lines added to the service descriptor of Figure A.5 to be discovered by other services in a network (this can also be done in the GUI)

6.3 EXPERIMENTS

In this section, several experiments to confirm some of the advantages of using SOA and OSGi, explained in previous chapters, are performed using OSGiLiath. First, a comparison of time of using OSGi and the services as normal classes is presented to demonstrate that OSGi does not adds extra overhead (as explained in Section 4.2.4). Experiments to demonstrate the automatic adding of new operators during runtime, and integration with other systems without modification of the existing source code are shown. Finally, a comparison of Lines of Code (LoCs) with other existing frameworks for EAs (presented in Section 3.4.2) is performed.

6.3.1 *Comparing overhead of using services*

One may think that working with services usually implies an overhead. This is true when communication protocols like SOAP are used, because the transmitted XML must be generated and parsed. However, as SOA is independent of the implementations, services also can behave as normal method calls in the same machine.

The basic GA implemented in Section 6.2.2 is also executed outside of the OSGi framework, and a normal Java class has been used to integrate the interfaces and implementations "as is". The population has been set to 64 individuals, parents have been selected using Binary Tournament, and the mutation rate has been fixed to 0.1. Worst individuals (parents and off-spring combined) are replaced, and the stop criterion has been set to 200 generations. Each experiment has been launched 30 times to solve the OneMax problem [129]. OneMax is a simple linear problem that consists in maximising the number of ones in a binary string. That is, maximize the expression:

$$f_{OneMax}(\vec{x}) = \sum_{i=1}^{N} x_i \tag{6.1}$$

Results of Table 6.1 show that time of services of OSGiLiath and numerical results are not affected by the OSGi framework: times are almost identical to the integration with Java code (p-value $<<$ 0.05).

Name	Average solution	Average Time (s)
OSGiLiath	612.36 ± 6.05	0.19 ± 18.21
OSGiLiath (without OSGi)	613.36 ± 4.50	0.19 ± 22.74

Table 6.1 Comparison of tested EA frameworks in time and development.

6.3.2 *Adding operators in runtime for self-adaptation*

Previous sections remarked that using SOA benefits are also related to self-adaptation. A simple example is presented here to demonstrate how easy is to convert a basic evolutionary algorithm into a self-adaptive one in OSGiliath.

To demonstrate that services can be managed and deployed during runtime, a simple experiment is proposed. An external service to the algorithm (called *Asynchronous Enabler*) runs in parallel consulting the population from time to time. If the best individual has not been improved in a number of times, this service automatically enables another implementation of the service *Parent Selector*. This new implementation is automatically bound to the the *Selector Gatherer* service and starts to use it. Figure 6.10 shows this configuration.

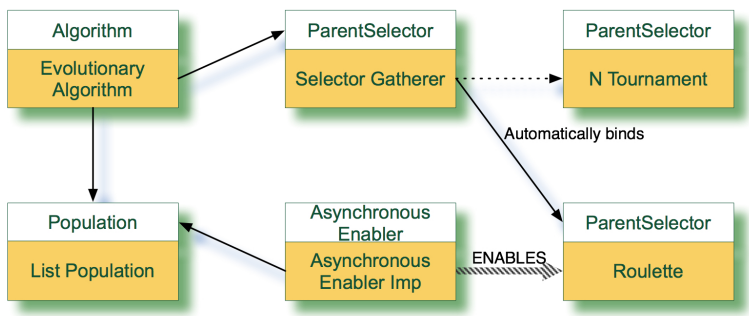


Figure 6.10 Service that enable automatically an operator to be used during runtime.

This enabler does not affect the code of the existing services (such as *Population* or *Evolutionary Algorithm*). The gatherer also does not need specific code to acquire all operators in execution time: it is done automatically thanks to OSGi.

Two versions have been compared: a non-adaptive version that only uses a *Binary Tournament* implementation for *Parent Selection* service, and an adaptive one, which automatically enables a *Roulette* implementation when a local optimum is found. The parameters used in this comparison (accessed from the *Parameters* service) are a population of 64 individuals, selector rate of 0.5, TPX crossover, bit flip mutation, and individual length of 60 genes. The *Roulette* selector service is enabled when the best individual of the population has not changed in 10 seconds (checked every 2 seconds). According to the classifications of parameter

Table 6.2 Basic deceptive bipolar function (s_i) for MMDP.

Unitation	Subfunction value
0	1.000000
1	0.000000
2	0.360384
3	0.640576
4	0.360384
5	0.000000
6	1.000000

Table 6.3 Results obtained using the Asynchronous Enabler to solve the MMDP problem.

	Non-adaptive	Adaptive
Generations	219403,10 ± 141692,16	167166,66 ± 93594,37
Evaluations	14041926,40 ± 9068298,82	10698794,66 ± 5990039,68
Time	68766,40 ± 45073,04	51710,40 ± 29329,21

control presented Section 3.3, this method is *adaptive*, the population is the key change, and it is based in *absolute evidence*.

The problem to solve is the MMDP (Massively Multimodal Deceptive Problem) [64]. The MMDP is designed to be difficult for an EA, due to its multimodality and deceptiveness. Deceptive problems are functions where low-order building-blocks do not combine to form higher order building-blocks. Instead, low-order building-blocks may mislead the search towards local optima, thus challenging search mechanisms. MMDP it is composed of k subproblems of 6 bits each one (s_i). Depending on the number of ones (unitation), s_i takes the values shown in Table 6.2.

The fitness value is defined as the sum of the s_i subproblems with an optimum of k (Equation 6.2). The search space is composed of 2^{6k} combinations from which there are only 2^k global solutions with 22^k deceptive attractors. Hence, a search method has to find a global solution out of 2^{5k} additionally to deceptiveness. In this work $k = 25$.

$$f_{MMDP}(\vec{s}) = \sum_{i=1}^{k} fitness_{s_i} \tag{6.2}$$

Table 6.3 shows the results obtained from the 30 executions of the two configurations tested. As it can be seen, automatic and adaptive enabling of selection operators has allowed an increase of performance, reducing time and evaluations (both significantly with a p-value<0.05 of a Wilcoxon test). It must be remarked that the aim of this experiment is not the numerical results obtained. This example has been used to demonstrate that

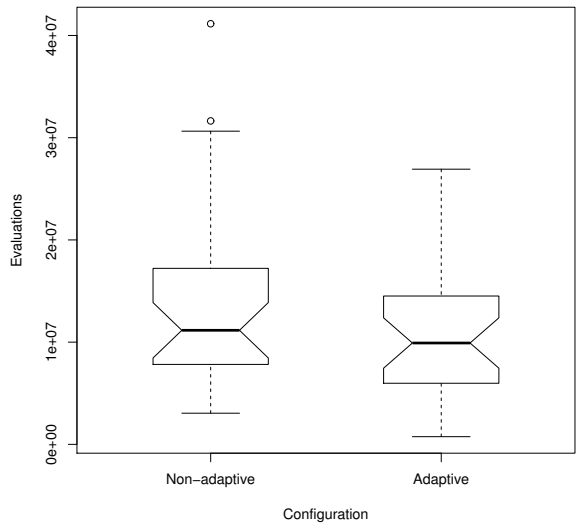


Figure 6.11 Boxplot of the number of evaluations in each configuration.

applying a methodology to develop loose coupled services that can be dynamically bound, without modification of the existing services, can be used to achieve better results.

6.3.3 *Increasing interoperability with other systems*

As previously stated, another advantage of SOA is the programming language independence with respect to the service interfaces. Although OSGi is a kind of SOA, it does not include the capability of interoperability with other kind of services by default. However, adaptation services can be added to transform OSGi interfaces into other SOA interfaces, such as Web Services (presented in Chapter 4) without modifying the existing code. So, services that are not written in Java, neither OSGi-based, could use services implemented in OSGiLiath (and vice-versa).

For example, using ECF all OSGi service interfaces are transformed into WSDL interfaces (explained in Section 4.2.1) automatically. Thus, these services could be used from other systems, that do not need to know the implementation language of the services in OSGiLiath. An example where an OSGi interface is transformed into a WSDL interface is shown in Figure 6.12. The computation node A, based on OSGi, uses the OSGi interface of the computation node C to calculate the fitness. Node B uses the WSDL interface to do the same task. It is not necessary to modify existing services source code to convert an OSGi interface into a WSDL interface. This transformation is bi-directional: given an

WSDL interface, it can also be transformed into a service to use inside OSGi.

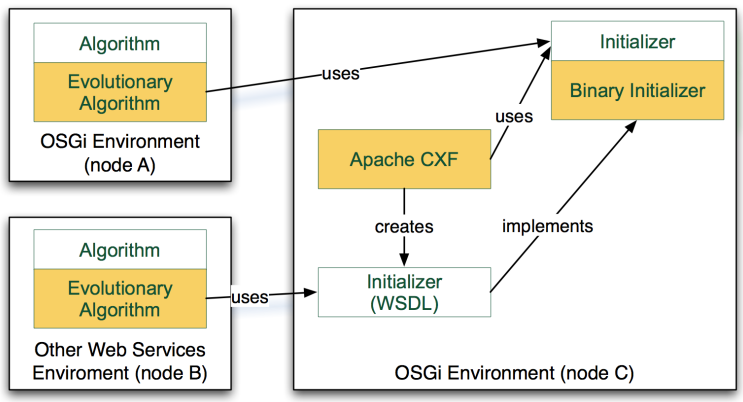


Figure 6.12 Communication with other kind of services. Apache CXF service automatically creates WSDL interfaces for the OSGi interfaces to be used from other environments

To test the differences in the communication protocol two configurations are going to be used. The first one (*OSGi configuration*) expose a distributed OSGi service to be automatically bound by another node, being undistinguishable from a local one (based in the OSGi Remote Specification standard). The second one (*SOAP configuration*) automatically generates a WSDL interface without modification of the code and it is accessed from a different programming language (PHP 5). The service are the implementation *Binary Initializer*, which receives the number of individuals to generate with the size given in the *Parameters* service. The individual size is 10 genomes and each configuration and number of individuals requested has been executed 100 times. Two Ubuntu 11.10 Intel(R) Core(TM)2 Quad CPU Q6600 machines in the same network have been used.

PHP: *PHP Hypertext Pre-processor*

Results are shown in Table 6.4 and plotted in Figure 6.13, which show the average time in seconds for each individual number and configuration. As expected (and explained in Section 6.2.1), the transmission time is higher in the SOAP configuration, because the extra markup added in the SOAP requests and responses for each individual transmitted. However, using the WSDL interface and SOAP transmission mechanism have advantages with respect to OSGi Remote Specification: language independence and public exposition of the interfaces for automatically create remote clients.

6.3.4 *Comparing with other Frameworks*

Since the OSGi framework adds features to the implementation of the algorithm that are similar (and even superior) to those offered by several of the frameworks described in Chapter 3, the

Number of individuals	OSGi configuration	SOAP configuration
250	0.015 ± 0.007	0.08 ± 0.001
500	0.018 ± 0.001	0.16 ± 0.003
1000	0.029 ± 0.001	0.326 ± 0.007
2000	0.052 ± 0.008	0.641 ± 0.012

Table 6.4 Transmission time (average ± std. dev.) for each configuration and number of individuals.

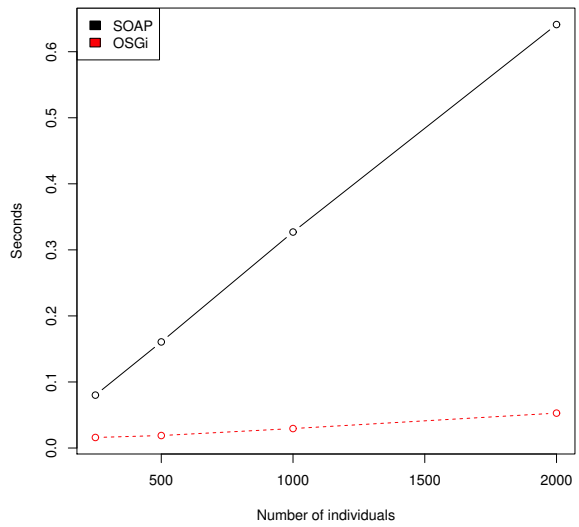


Figure 6.13 Transmission time for OSGi and SOAP configurations.

same algorithm of Section 6.3.1 (with the same operators and parameters) has been coded using several well-known frameworks, such as Mallba (C++), Algorithm::Evolutionary (Perl), and ECJ (Java). Table 6.5 shows the execution time achieved, average solution, and Lines of Code (LoC) needed to integrate the algorithm for each framework. All the algorithm implementations have been executed on the same computer, an Ubuntu 12.04 Linux machine with Intel Core2 Quad CPU Q8200 @ 2.33GHz, 4 GB RAM, without any distribution mechanisms. The LoC have been calculated using *sloccount* program.

Note that, although the services are developed under SOA, and bound in runtime, they are not distributed. Algorithmically, all frameworks behaves the same, and results are not quite different. The differences among frameworks are produced because the different implementations of random generators, operators or logs, for example. In the work of Merelo *et al.* [104], these different behaviours are also justified.

Regarding LoCs, MALLBA has the higher number: this is because every algorithm is created as a "skeleton" and a duplication of code exist for each algorithm and problem to execute. This is produced because many operations affect global variables:

Table 6.5 Comparison of tested EA frameworks in time and development.

Name	Average solution	Average Time (s)	LoC
OSGiLiath	612.36 ± 6.05	0.19 ± 18.21	10
OSGiLiath (without OSGi)	613.36 ± 4.50	0.19 ± 22.74	103
MALLBA	578.76 ± 7.48	0.16 ± 0.0003	2073
ECJ	602.76 ± 6.08	1.40 ± 0.03	5
Algorithm::Evolutionary	617.60 ± 12.92	7.78 ± 0.29	41

for example the method *select_offsprings()* affects the global variables *parents* or *aux*. Using this method as an external service would require a whole change in many parts of the code. Thanks the loose-coupling of Perl, many lines of code are saved using Algorithm::Evolutionary, mainly because many parameters and operators are defined by default.

ECJ and OSGiLiath do not require code to combine different operators, just only to modify some configuration files without re-compilation. The main difference is that in ECJ the available operators must be known prior to execution (the interfaces are linked in the source code), while in OSGiLiath, all interfaces are bound to their implementations in configuration files, or even without them (for example, the implementations appear in the same network/machine). But there also exist limitations, because ECJ only provides fixed mechanisms of distribution, and only certain parts of the framework can be accessed remotely, while in OSGiLiath all operators have the chance to be distributed if desired, modifying the configuration files.

It must be remarked that OSGiLiath does not try to compete with the other frameworks (they are widely accepted, completed and tested), it is only an example of how to develop EAs under the SOA paradigm.

6.4 CONCLUSIONS

In this chapter, SOA-EA methodology has been used to create a framework, called OSGiLiath, that accomplish with the restrictions and benefits of using SOA for EAs (explained in Chapter 5) to fulfil the **Objective 3** of this thesis: Validate the methodology using a SOA technology. A number of services have been found in the identification step and the inputs and outputs have been described in the specification phase.

Then, in the implementation step, a comparison of two SOA technologies based in standards have been performed: OSGi has been selected because a number of advantages over Web Services, such as transmission speed and ease of development. The

abstract services of previous steps have been implemented to form a basic GA as an example. These services can be combined in several ways to obtain different algorithms (from a canonical GA, a NSGA-II has been created just adding new services).

This chapter also has presented experiments to show how services can be dynamically bound to change the needed EA aspects. The source code of the basic EA services have not been re-written or re-compiled to achieve this task. Also, no specific source code for a basic distribution have been added, neither the existing source code has been modified, allowing saving time in integration and development, with respect to other available EA frameworks. Two different transmission mechanisms have been tested (ECF Generic and SOAP) and two different standards for exposing services (WSDL and OSGi Remote Specification) have been used. Exposing services with public standards helps to Open Science, as explained in Section 4.4. Therefore, all services developed have accomplished the requirements of Section 5.5.

The source code of OSGiLiath is available in the web page `http://www.osgiliath.org` under a GNU/LGPL license. Appendix B describes the current components available in this framework.

Next chapters SOA-EA and OSGiLiath will be used to solve several problems: parameter adaptation in heterogeneous clusters and genetic programming to generate agents in video-games.

7

PARAMETER ADAPTATION IN HETEROGENEOUS MACHINES

It's a wonderful thing, as a writer, to be given parameters and walls and barriers.

— Neil Gaiman

INDEX

Adapting Evolutionary Algorithms to dynamic and heterogeneous architectures using SOA require some kind of mechanism to take advantage of the different capacities of the resources that are going to be used. As a part of the methodology to adapt EAs to the SOA paradigm, in this chapter several adaptation strategies are compared to evaluate which one is the most adequate.

In this chapter, the capabilities of OSGiLiath will be used to investigate if adapting the parameters of a distributed SOEA taking into account the computational capabilities of the different nodes of execution leads to an increase of performance. This question is interesting due to new trends in distributed computing presented in Chapter 4, such as Cloud Computing, GRID or

Service Oriented Science are leading to heterogeneous computational devices, including for instance, laptops, tablets or desktop PCs, working in the same environment. Thus, many laboratories, which do not count with classic clusters but the usual workstations used by scientists, can leverage this motley set as a heterogeneous cluster. As explained in Chapter 3, Distributed Evolutionary Algorithms have been tested successfully in this type of systems and they have become very popular because their implementation is not complex [71]. Also, as presented in Chapter 4, a possible way to increase the interoperability within these systems is SOA, and specifically, the use of OSGiLiath framework as an example.

In this chapter, OSGiliath will be used to create a heterogeneous distributed system to be used to develop a scientific research related with parameter tuning and control (explained in Section 3.3.1). Several services to deal with automatic binding and parameter control will be developed following SOA-EA, and deployed in different cluster configurations. The conclusions of this study could help to validate if the SOA paradigm, where different resources can collaborate, can take advantage of the heterogeneity. Also, to investigate if a change in a parameter affects all the services that are going to be executed in the whole system.

7.1 BACKGROUND AND PROBLEM DEFINITION

In Section 3.3.2 several works about adaptation in heterogeneous environment were presented. For example, the work of Alba *et al.*, where dEAs with the same parameter configuration could be more efficient in time and evaluations on heterogeneous hardware configurations than on clusters with homogeneous devices, or the work of Gong and Fukunaga, where different parameters in each island increased performance. The first aim of this chapter is to use OSGiLiath to demonstrate if adapting the subpopulation size to the computational power of an heterogeneous cluster nodes presents an improvement in execution time. New services will be created to deal with different distributed nodes and setting the population sizes to give an insight to the following research questions:

- Can a distributed SOEA be adapted to leverage the capability of a heterogeneous cluster?
- How the adaptation of the sub-population size to the computational power affects the execution time and number of evaluations?

```
population ← initializePopulation()
while stopcriterionnotmet do
  parents ← selection(population)
  offspring ← recombination(parents)
  offspring ← mutation(offspring)
  population ← population + offspring
  if time to migrate then
    migrants ← selectMigrants(population)
    remoteBuffer.send(migrants)
  end if
  if localBuffer.size ≠ zero then
    population ← population + localBuffer.read()
  end if
  population ← removeWorst(population)
end while
```

Figure 7.1 Pseudo-code of the used dEA: a distributed Genetic Algorithm (dGA).

- Is there any difference between using the same sub-population sizes in a homogeneous and a heterogeneous cluster?
- How is each service of the algorithm (selection, recombination, mutation, replacement and migration) affected by the different configurations?

These previous questions should help to give some information about the research of SOEAs, as the have to deal with dynamism and parameter adaptation to heterogeneous resources. For example, changing a parameter in one of the nodes that execute the SOEA may have an enormous impact in their performance.

7.1.1 *Algorithm to develop*

The experimentation is centred in a distributed GA. Figure 7.1 shows the pseudo-code of the used algorithm. The algorithm is steady-state, i.e. every generation the offspring is mixed with the parents and the worst individuals are removed. This algorithm is general enough and not designed specifically for this study.

The used neighbourhood topology for migration between islands (nodes) is a ring (see Figure 3.3 in Chapter 3). The best individual is sent to the neighbour in the ring, after a fixed number of generations in each island. The algorithm stops when the optimum (the solution to the problem) is found. Therefore, a mechanism to stop all the executing nodes must be implemented.

7.1.2 *Problems to solve*

The results should be independent of the problem used, but the next ones have been selected because they cover different characteristics and computational demands. The problems to evaluate are the Massively Multimodal Deceptive Problem (MMDP) and the OneMax problem. Both problems have been described previously in Section 6.3. Each one requires different actions/abilities by the GA at the level of population sizing, individual selection and building-blocks mixing [71].

7.1.3 *Hardware and parameter configurations*

As we are going to test parameter adaptation to hardware, different configurations should be used to compare and validate if the change in the parameters depends only of the parameters, the hardware heterogeneity, or the combination of both.

- HoSi/HeHa: Homogeneous Size/Heterogeneous Hardware. The same sub-population size in each island on a heterogeneous cluster.
- HeSi/HeHa: Heterogeneous Size/Heterogeneous Hardware. Different sub-population sizes in each island on a heterogeneous cluster.
- HoSi/HoHa: Homogeneous Size/Homogeneous Hardware. The same sub-population size in each island on a homogeneous cluster.
- HeSi/HoHa: Heterogeneous Size/Homogeneous Hardware. Different sub-population sizes (the obtained for HeSi/HeHa) in each island on a homogeneous cluster.
- AdSi/HeHa: Adaptive Size/Heterogeneous Hardware. Online adaptation of sub-population sizes in each island on a heterogeneous cluster.

7.1.4 *Homogeneous Size configuration*

In this configuration, each node has 256 individuals (so, the total amount is 1024). This value has been chosen empirically, as it is big enough to test different sub-population sizes.

The results of executing the algorithm in the will be used to set the sizes of the next configuration.

7.1.5 *Heterogeneous Size configuration*

In this chapter, for a possible offline way to calculate the computational performance of each node, the average number of generations obtained in the HoSi/HeHa configuration for both problems will be used to determine the computational power of the heterogeneous machines. This comparison takes into account all the evolutionary process in a fair manner (proportional to the memory, processor and network usage), instead of a traditional benchmark that usually relies only on the CPU speed. The proposed technique is a possible way to establish the computational power for the experiments of this chapter and to determine if changing the sub-population size according the computational power reduces the computing time of the whole approach.

Thus, we have used the obtained average number of generations in the previous sub-section (Table 7.2) to set proportionally the sizes in the HeSi/HeHa and HeSi/HoHa configurations, by dividing the total number of individuals (1024). Note that, even having two nodes with the same processors and memory (HeN1 and HeN2), they could have different computational power: this may be produced by different operating systems, virtual machine versions, or number of processes being executed (inside a node).

7.1.6 *Adaptive Size configuration*

A third experiment is proposed to validate the hypothesis of sub-population size adaptation to computational resources. In this case, the adaptation of the sub-population size to the computational power of the islands (nodes) is performed during runtime (online). Each time a node (N) receives an individual, it compares its current number of generations (Gen_N) with the ones of the node who sent the individual (node $N-1$ in the ring). Then, the sub-population size is adapted proportionally to the difference in the number of generations, following the next equation:

$$size'_N = \frac{Gen_N}{Gen_{N-1}} size_N \tag{7.1}$$

If the new size is larger than the actual size, new individuals are added to the sub-population cloning random existent ones. Otherwise, the sub-population must be reduced and thus, the worst are removed. Therefore, the service *Population* is used to manage the population, as explained in Chapter 6.

With this possible online adaptation scheme, each node only requires to receive information from one of the neighbours and not from the whole system. Thus, each node tends to have a number of individuals proportional to their computational power with respect to the other nodes. Experiments on homogeneous cluster do not alter the sub-population sizes, since the number of current generations are equal in all nodes during runtime.

7.1.7 *Restrictions*

Once the problem to solve, the algorithm to implement and the different configuration have been described, the restriction to the services to develop are the summarized:

- There is not a central control node.
- The number of nodes participating in the experiment should not be fixed.
- All nodes automatically bind the available distribution (migration) services.
- The nodes must stop when the optimum is found.
- Services must be executed in heterogeneous machines with different operating systems and architectures.
- It is necessary a log service to show the current state of the algorithm and service timings.

7.2 DESIGNING THE SERVICES WITH SOA-EA

Once the description of the system to develop has been presented, the SOA-EA methodology (explained in Chapter 5) is used to create a SOEA that fulfils the previous requirements.

7.2.1 *Identification*

In addition of the services for calculating the fitness of the problems (*MMDPFitnessCalculator* and *OneMaxFitnessCalculator*), or the *OptimumStopCriterion* and crossovers and mutators created in previous chapter, new services should be added. The first one deals with the migration between islands, so, a service *Migrator* (to receive and send individuals from/to other nodes) needs to be created. Also, it is necessary a service to start or stop the remote loops in all islands at the same time (service *Launcher*).

Finally, it is necessary a to manage the received individuals from the migrators and control the population size, this can be performed in a new *Replacer* interface: *AdaptiveReplacer*.

7.2.2 Specification

The service *Launcher* and its specification *ExperimentLauncher* automatically binds all available *Algorithm* services in the network. This service can start all the distributed EAs at the same time (for example, from command line when all nodes are online and they services bound). When one of the EAs has finished, it has to notify the others to stop.

To perform the migration taking into account the previous requirements, each node offers a migration buffer to accept foreign individuals. Also, in order to reduce bottlenecks in distributed executions, asynchronous communication needs to be provided to avoid idle time using reception buffers (that is, the algorithm does not wait until new individuals arrive, but the buffers cannot be used again until the reception is done). This kind of communication offers an excellent performance when working with different nodes and operating systems, as demonstrated in [105].

The *Migrator* has two operations: *send* and *read*. The first one is used to send the individuals to the migrator, and the other is used to read the individuals of that migrator. Usually, each node (island) has one migrator to receive individuals, and references to the other nodes' migrators. In our case, the implementation of *Replacer* binds the local *Migrator* to write in it the individual(s) to sent. In this chapter, the *Migrator* implementation is the *MigratorRingBuffer*: this class implements that interface and automatically binds all the Migrators available in the environment (in a vector of references). So, the migrators can be added during runtime, and no stop the algorithm if one node fails. The *MigratorRingBuffer* sends the individuals to the remote Migrator whose id is inmediatelly higher than the local id (or the smaller, if it not exist) following a ring topology. Figure 7.2 shows this configuration. The *Replacer* implementation, a reference to the local *Migrator* interface just send and read the individuals. The *MigratorRingBuffer* implementation binds an unbinds other migrators in other nodes, keeping a reference to these remote service interfaces. The *AdaptiveReplacer* implementation binds the local *Migrator* service and it manages the population sizes.

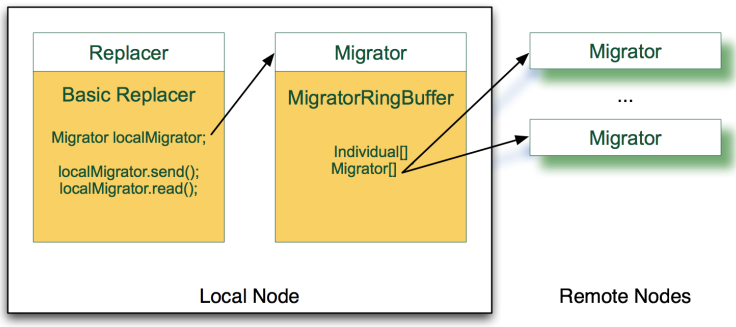


Figure 7.2
Using the Migrator service to create a distributed island EA with a ring topology (white boxes are service interfaces and orange boxes are implementations).

Table 7.1
Details of the clusters used: a homogeneous cluster (Ho), and a heterogeneous cluster (He)

Name	Processor	Memory	Operating System	Network
Homogeneous cluster				
HoN[1-4]	Intel(R) Xeon(R) CPU E5320 @ 1.86GHz	4GB	CentOS 6.7	Gigabit Ethernet
Heterogeneous cluster				
HeN1	Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz	4GB	Ubuntu 11.10 (64 bits)	Gigabit Ethernet
HeN2	Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz	4GB	Ubuntu 11.04 (64 bits)	Gigabit Ethernet
HeN3	AMD Phenom(tm) 9950 Quad-Core Processor @ 1.30Ghz	3GB	Ubuntu 10.10 (32 bits)	100MB Ethernet
HeN4	Intel (R) Pentium 3 @ 800MHz	768 MB	Ubuntu 10.10 (32 bits)	10MB Ethernet

7.2.3 *Implementation and Deployment*

All services have been implemented in OSGiLiath. The *Migrator* and *Algorithm* services are exposed using ECF Generic Server, as explained in Section 6.2.1). This services are, therefore, automatically bound to each node in the clusters, without notify their IP address. Extra code to manage the communication has not been added, as all services are undistinguishable of being remote or local. Remote *Migrators* and *Algorithms* are bound thanks to the bind/unbind methods of declarative services and ECF (explained in Section 6.2.1). Several properties can added to the service allows to ECF automatically announce the implementation to all nodes in the network and no specific code is required to change from one distribution mechanism to another.

The services have been deployed in two different computational systems: a *heterogeneous cluster* and a *homogeneous cluster*. The first one is formed by four different computers of our lab with different processors, operating systems and memory size. The latter is a dedicated scientific cluster formed by homogeneous nodes. Table 7.1 shows the features of each system and the name of the nodes.

7.3 EXPERIMENTAL RESULTS

Once the services have been created, the different combinations of systems and parameter are evaluated. Table 7.3 summarizes all the parameters used in the experiments.

Node	HeN1	HeN2	HeN3	HeN4
MMDP problem				
Generations	10990.25	10732.075	7721.15	717.95
Proportion	36.43	35.58	25.59	2.38
OneMax problem				
Generations	2430.27	2353.77	1423.77	91.5
Proportion	38.58	37.36	22.6	1.45

Table 7.2 Average number of generations in each node needed to find the optimum on the heterogeneous cluster with heterogeneous size.

Name	Value
Crossover type	Two-point crossover
Crossover rate	0.5
Mutation probability of each gene	1/individual size
Selection	2-tournament
Replacement	Steady-state
Generations to migrate	64
Number of individuals to migrate	1
Stop criterion	Optimum found
Individual size for MMDP	150
Individual size for OneMax	5000
Runs per configuration	40
Total individuals in HoSi and HeSi	1024
Sub-population size in each node in HoSi	256
Sub-population sizes in HeSi for MMDP	374, 364, 262 and 24 (from N1 to N4) (see Section 7.3.1)
Sub-population sizes in HeSi for OneMax	396, 382, 232 and 14 (from N1 to N4) (see Section 7.3.1)
Maximum island size in AdSi	1024
Minimum island size in AdSi	16
Initial island size in AdSi	256

Table 7.3 Parameters used in all configurations.

The three main objectives of parallel programming are to tackle large computational problems, increase the performance of algorithms in a finite time, or reduce computational time to solve the problem (reaching the optimum). In this chapter, we focus in the last objective. As claimed by Alba and Luque in [3], assessing the performance of a parallel EA by the number of fitness function evaluations required to attain a solution may be misleading. In our case, for example, the evaluation time is different in each node of the heterogeneous cluster, so the real algorithm speed (in time) could not be reflected correctly. However, the number of evaluations has been included in this chapter to better understanding the results. The total number of generations carried out by all nodes, and the maximum number of generations required by the faster node in each configuration are also shown. It is difficult to compare the performance of HoHa and HeHa for the same reason: the evaluation time is different in each system (and even in each node). Thus, one of the objectives in this chapter is not making the heterogeneous cluster comparable or better

Table 7.4
Results for the MMDP problem.

Configuration	Max. generations	Total generations	Total evaluations	Time (ms)
HoSi/HeHa	11194.8 ± 18810.08	30161.42 ± 50722.03	7723372.8 ± 12984841.71	27871.075 ± 44583.14
HeSi/HeHa	2506.1 ±5308.872	8683.9 ± 18459.58	2453677 ±5217896.18	8110.9 ± 17162.86
AdSi/HeHa	2407.10 ±3938.43	8376.35 ± 14140.55	2948946.15 ± 5165324.99	10235.89 ± 17193.98
HoSi/HoHa	2614 ±5889.93	10259.22 ± 23153.23	2628409.6 ± 5927278.22	11560.8 ± 26072.14
HeSi/HoHa	5411.92 ±15608.81	10689.15 ± 30790.7	1844908.1 ± 5314771.88	9520.325 ± 27237.35

in time than the homogeneous one (because they are, obviously, different), but showing that the same parameter configuration can improve performance in time on heterogeneous clusters and could not have an effect on homogeneous ones.

7.3.1 *Obtaining the HeSi sizes*

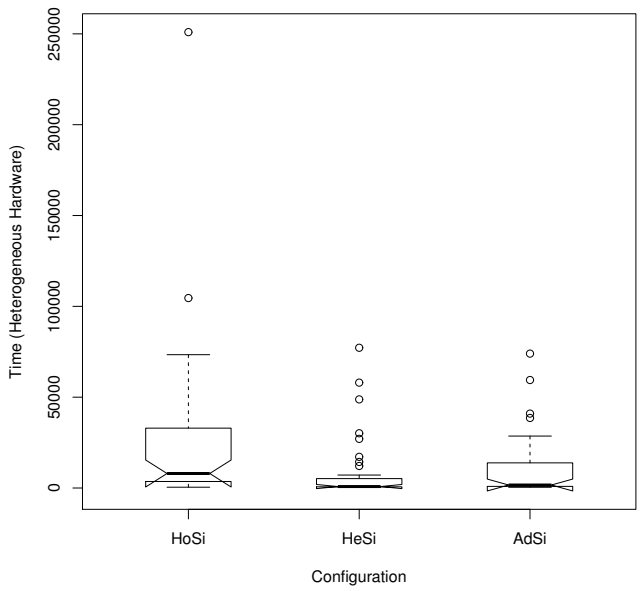
After executing the algorithm 40 times per problem on the heterogeneous cluster, we have obtained the average number of generations in each node, as it can be seen in Table 7.2. Note how the generations attained (and their proportion in every node) to reach the optimum depends on the problem considered (besides the hardware).

7.3.2 *MMDP results*

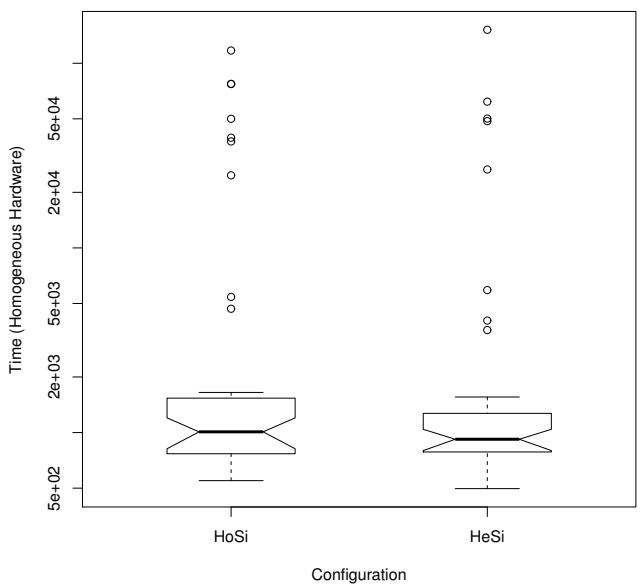
Table 7.4 shows the results for the MMDP problem. These results are also shown in the boxplots of Figure 7.3 (time) and Figure 7.4 (evaluations). Table 7.8 shows the statistical significance of the results. First, a Kolmogorov-Smirnov test is performed to assess the normality of the distributions. As all distributions are not normal, we use non-parametric tests. To compare between two methods (HoSi and HeSi in the homogeneous cluster) a Wilcoxon test has been applied. For a three methods comparison (HoSi, HeSi and AdSi on heterogeneous cluster) a Kruskal-Wallis test has been used.

In the HeHa system, offline adaptation of the sub-population to the computational power of each node makes the algorithm finish significantly earlier, and also, needing a lower number of evaluations to reach the solution. On the other hand, in the HoHa system, setting the same sub-population sizes makes no difference in time and evaluations, that is, changing this parameter has no influence in the algorithm's performance (p-value=0.52 for time and 0.08 for evaluations).

To see the differences on how the evolution is being performed, the average fitness in each node of HeHa is shown in Figures 7.5 and 7.6. As it can be seen, with the HeSi (Figure 7.6), the local optima are overtaken in less time than HoSi (Figure 7.5).



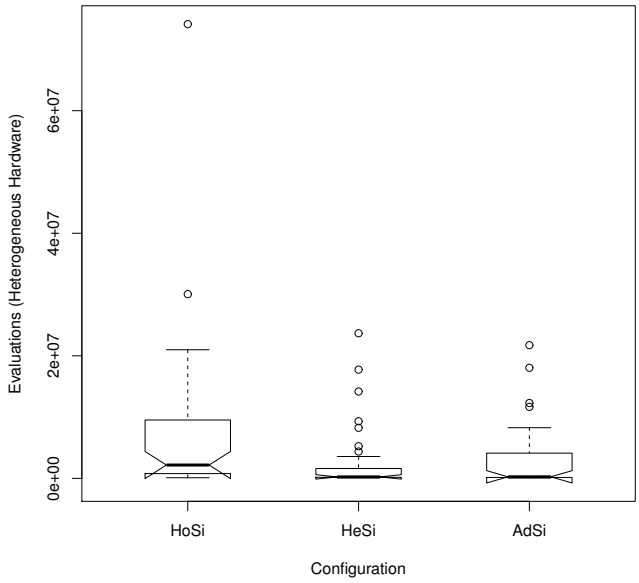
(a) Heterogeneous cluster



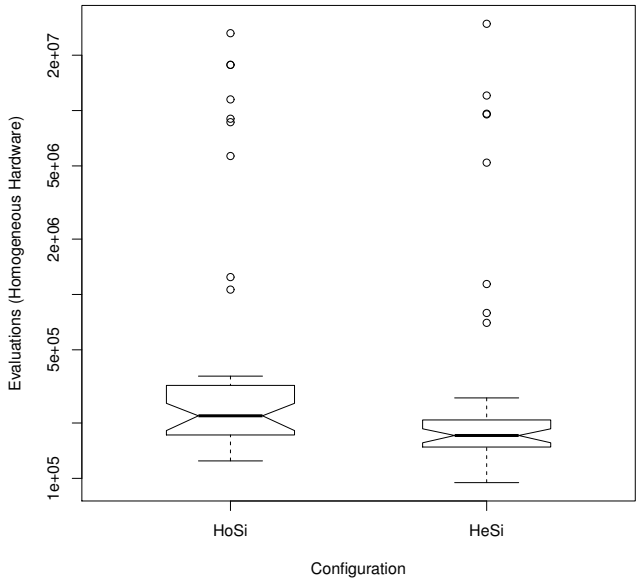
(b) Homogeneous cluster

Figure 7.3
Time to obtain the optimum in the MMDP problem (milliseconds).

Figure 7.4
Number of evaluations for MMDP problem.



(a) Heterogeneous cluster



(b) Homogeneous cluster

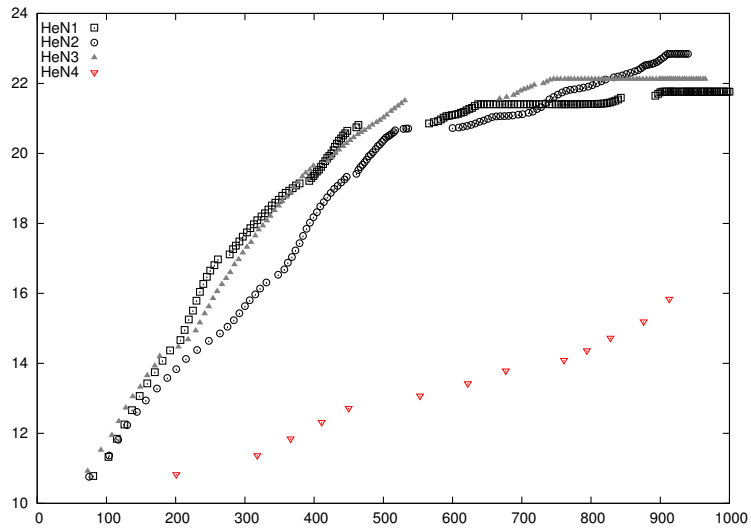


Figure 7.5 Average fitness in the first 1000 milliseconds of execution of the four nodes of the heterogeneous cluster with the same sub-population sizes (HoSi/HeHa) for the MMDP problem.

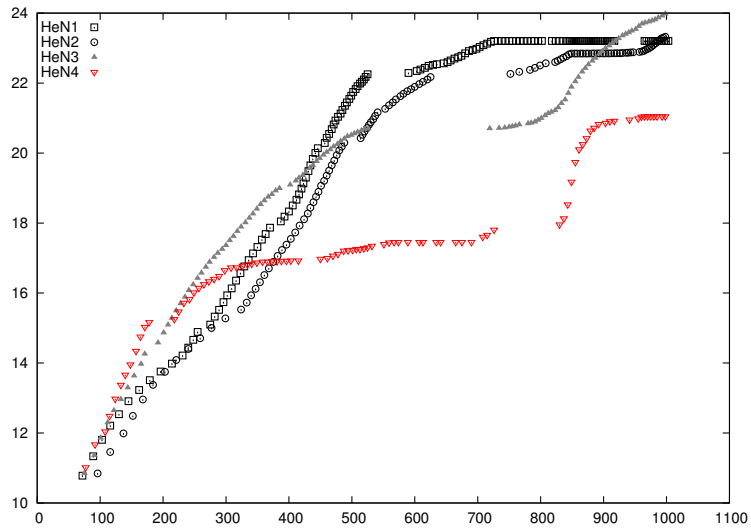


Figure 7.6 Average fitness in the first 1000 milliseconds of execution of the four nodes of the heterogeneous cluster with different sub-population sizes (HeSi/HeHa) for the MMDP problem.

This can be explained because in HeSi, the migration from HeN4 to HeN1 is performed faster, adding more heterogeneity to the whole system. Gaps in the figures correspond to the time spent in the nodes for sending the migrant individual to other nodes (not while they are receiving them). In the HoHa configurations, the evolution of sub-population is performed at the same time, being the average fitness similar for all nodes during all runs.

With respect to AdSi/HeHa, results are significantly equal (p-value 0.139) to HeSi/HeHa (and, therefore, better than HoSi/HeHa), but this time no previous tuning has been required. Average sub-population sizes in each node are shown in Table 7.5. The proportions of size are similar to the proportions in Table 7.2. Figure 7.7 plots all the possible sizes in each node during all the runs.

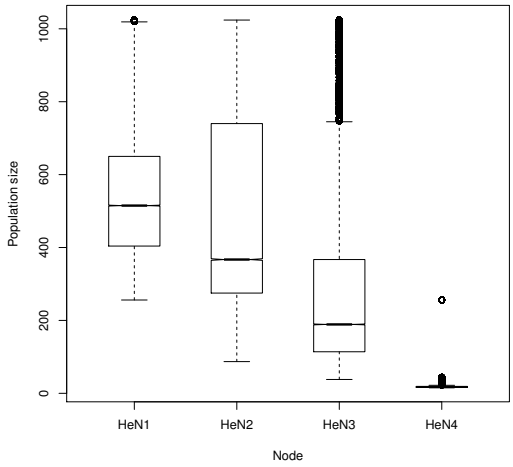


Figure 7.7 Boxplots of the sub-population sizes in each node of the AdSi/HeHa configuration during all the runs for the MMDP problem.

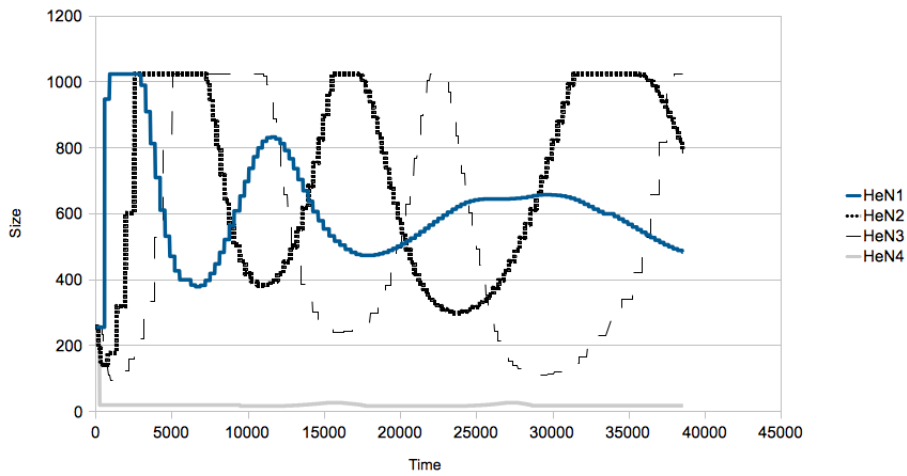


Figure 7.8 Population size in all nodes of the AdSi/HeHa configuration during one execution to solve the MMDP problem.

This figure shows that the variation of the sub-population sizes lies proportionally to the computational power of each node. The outliers in boxplots are produced during the size changing, as it can be seen in Figure 7.8. As N4 is the slower node with difference it keeps its size always close to the minimum (16 individuals).

Summarizing, adapting the sub-population sizes to the computational power of each machine (offline and online) has reduced the time to obtain the optimum. The same heterogeneous fixed sizes in the homogeneous cluster does not produce a significant decrease of running time, so the improvement is produced by the heterogeneity and not due to the different island sizes. More-

Node	HeN1	HeN2	HeN3	HeN4
Size	556.31	504.30	321.15	19.81
Proportion	39.69	35.98	22.91	1.41

Table 7.5 Average sub-population size in each node on the heterogeneous cluster with adaptive size (MMDP) after all runs.

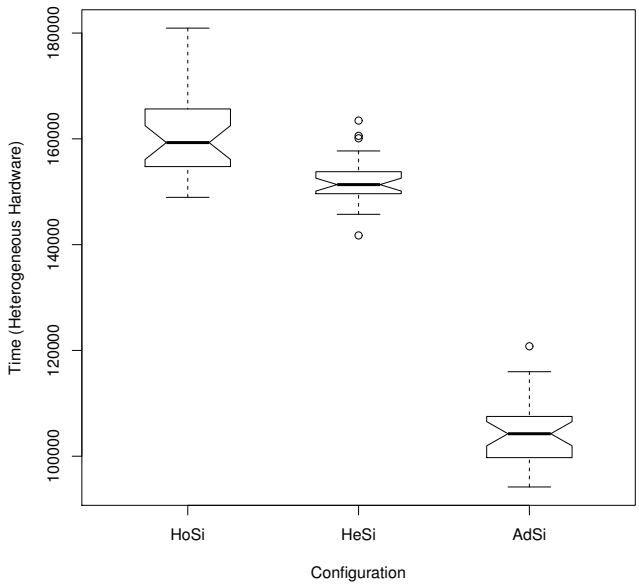
over, the AdSi proposal is not applicable in HoHa because there is not differences of generations during runtime.

7.3.3 *OneMax results*

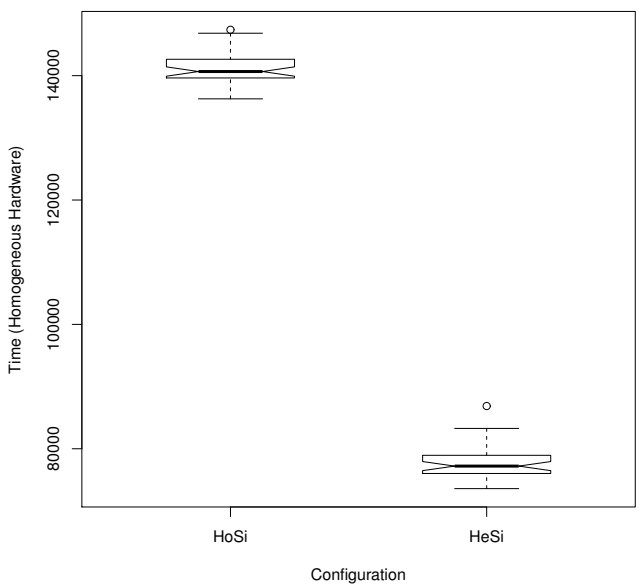
Results for this problem are shown in Table 7.6 and Figures 7.9 and 7.10. In this case, adapting offline the sub-population sizes significantly decreases the running time for solving it in the heterogeneous cluster, but this time, the number of evaluations is increased (see statistical significance in Table 7.9). In the homogeneous system, the effect of changing the sub-population sizes is clearer, and this time the number of evaluations (and therefore, the time) are reduced (both significantly).

The efficiency to resolve OneMax problem depends mainly on the ability to mix the building-blocks, and less on the genetic diversity and size of the population (as with MMDP). No genetic diversity is particularly required. When properly tuned, a simple Genetic Algorithm is able to solve OneMax in linear time. Sometimes, problems like OneMax are used as control functions, in order to check if very efficient algorithms on hard functions fail on easier ones. As it can be seen in Figure 7.11, the average fitness of all sub-populations are increasing in linear way in the HoSi/HeHa configuration. However, the slower node evaluates extremely fewer times. On the other side, in Figure 7.12, smaller sub-population sizes make that slower nodes increase the number of evaluations, but the average fitness is also maintained in linear way (and in smaller increase rate) between migrations. Nevertheless, the other nodes still perform a higher number of evaluations. That is the reason why the number of evaluations is higher in HeHa, and lower in HoHa. Computational time is more efficiently spent in faster nodes, having a higher chance to cross the individuals. In addition, due to the larger size of individuals in the OneMax problem (5000 bits vs. 150 of the MMDP), the transmission time is larger, (white gaps in the figures). It also implies that HeN4 sends its best individual to HeN1 in an extremely large amount of time when using HoSi (every 64 generations).

Figure 7.9
Time to obtain the optimum in the OneMax problem (milliseconds).

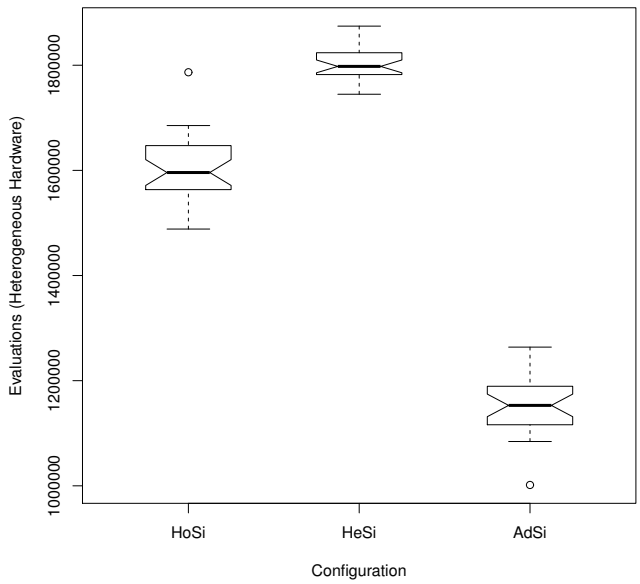


(a) Heterogeneous cluster

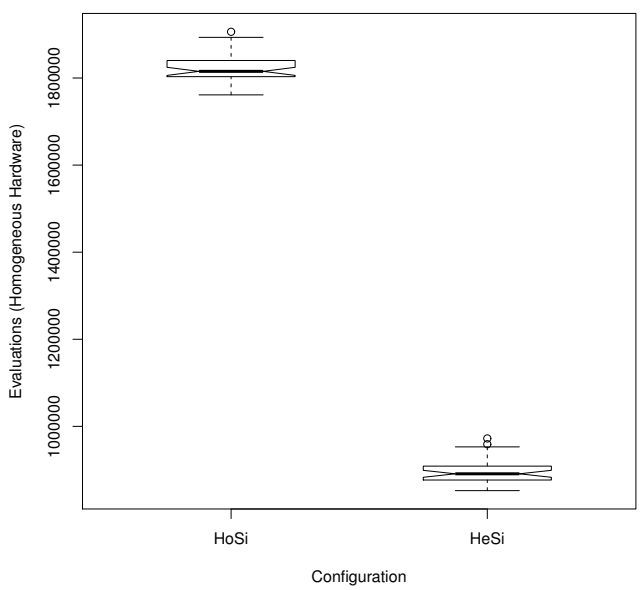


(b) Homogeneous cluster

Figure 7.10 Number of evaluations for OneMax problem.



(a) Heterogeneous cluster



(b) Homogeneous cluster

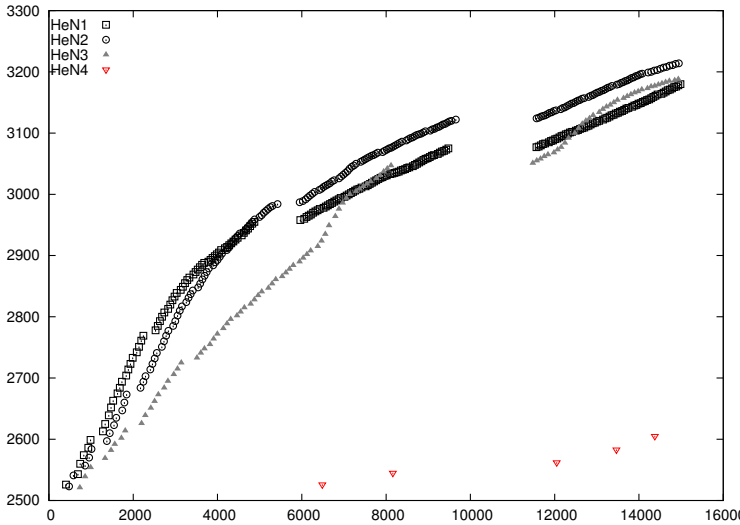


Figure 7.11 Average fitness in the first 15000 milliseconds of execution of the four nodes of the heterogeneous cluster with the same sub-population sizes (HoSi/HeHa) for the OneMax problem.

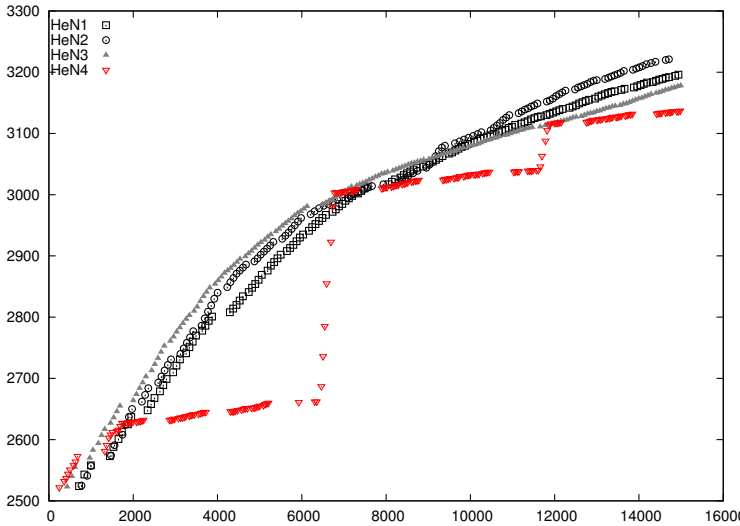


Figure 7.12 Average fitness in the first 15000 milliseconds of execution of the four nodes of the heterogeneous cluster with different sub-population sizes (HeSi/HeHa) for the OneMax problem.

Configuration	Max. generations	Total generations	Total evaluations	Time (ms)
HoSi/HeHa	2430.34 ± 70.16	6299.31 ± 250.87	1614673.45 ± 64223.09	160713.65 ± 8873.46
HeSi/HeHa	2643.34 ±150.82	7969.58 ±214.92	1802321.65 ± 30511.96	151822.75 ±4764.95
AdSi/HeHa	3698.30 ± 494.56	9465.25 ± 635.07	1149277.43 ± 58887.13	103919.33 ± 6296.39
HoSi/HoHa	1791.32 ± 31.64	7111.05 ±125.11	1822476.8 ±32029.78	141176.1 ±2493.72
HeSi/HoHa	13698.12 ± 406.85	16012.625 ± 482.61	895698.2 ± 29520.99	77898.85 ± 2935.57

Table 7.6 Results for the OneMax problem.

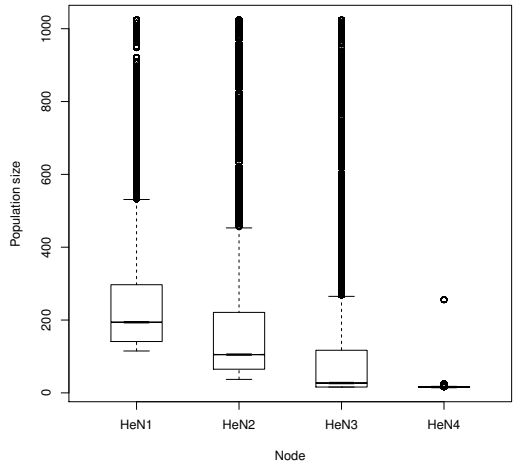


Figure 7.13 Boxplots of the sub-population sizes in each node of the AdSi/HeHa configuration during all the runs for the OneMax problem.

In the AdSi/HeHa configuration significantly better results in terms of execution time (and number of evaluations) are also attained, and even better than those obtained with HeSi. Average sizes (Table 7.7) and boxplots (in Figure 7.13) during all the runs also show proportionality to the computational power of each machine. As in MMDP case, some oscillations (outliers in boxplots) may appear during the execution (as it can be seen in Figure 7.14).

7.3.4 *Running time analysis*

This sub-section shows the analysis the time spent by each node of the clusters in every service of the EA for each configuration with fixed sizes (HoSi and HeSi). Tables 7.10 and 7.11 show the average and standard deviation of the time spent in each stage of the algorithm (He=Heterogeneous cluster, Ho=Homogeneous

Node	HeN1	HeN2	HeN3	HeN4
Size	267.09	158.63	74.20	16.29
Proportion	51.73	30.72	14.37	3.15

Table 7.7 Average sub-population size in each node on the heterogeneous cluster with adaptive size (OneMax).

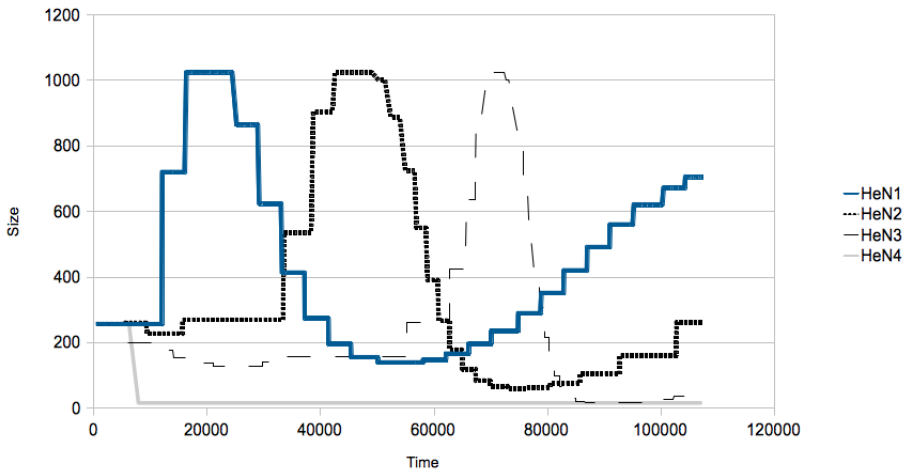


Figure 7.14 Sub-population size in each node during one execution of the AdSi/HeHa configuration to solve the OneMax problem.

Table 7.8 Statistical significance of the results for MMDP.

Time					
Kruskal-Wallis chi-squared = 20.3042, df = 2, p-value = 3.899e-05					
Configuration	Test	obs.dif	critical.dif	p-value	difference
AdSi/HeHa-HeSi/HeHa	K-W	13.19231	18.38851	0.1390	FALSE
AdSi/HeHa-HoSi/HeHa	K-W	21.11538	18.38851	0.0067	TRUE
HeSi/HeHa-HoSi/HeHa	K-W	34.30769	18.38851	9×10^{-5}	TRUE
HoSi/HoHa-HeSi/HoHa	Wilcoxon	-	-	0.52	FALSE
Evaluations					
Kruskal-Wallis chi-squared = 11.9676, df = 2, p-value = 0.002519					
AdSi/HeHa-HeSi/HeHa	K-W	2.794872	18.38851	1.0	FALSE
AdSi/HeHa-HoSi/HeHa	K-W	21.487179	18.38851	0.0207	TRUE
HeSi/HeHa-HoSi/HeHa	K-W	24.282051	18.38851	0.0028	TRUE
HoSi/HoHa-HeSi/HoHa	Wilcoxon	-	-	0.08	FALSE

cluster). Figures 7.15 and 7.16 graphically compare these results. As it can be seen, the migration is the most time consuming operation in all configurations, being the migration in HeHa more expensive than in HoHa. This happens because we are using the multi-purpose laboratory network to communicate the nodes, instead of the specific one used in the HoHa system. Note that the standard deviation of the migration is larger in the HeHa cluster because the network is having real conditions of traffic during the experiment. In the MMDP problem (Table 7.10) changing the sub-population size does not affect the migration time, but it affects the rest of the algorithm's stages. However, with larger data communications (individuals of 5000 elements of the OneMax problem), the sub-population size affects the migration time of all nodes. This might be due to the synchronization of migration

Time					
Kruskal-Wallis chi-squared = 66.4965, df = 2, p-value = 3.635e-15					
Configuration	Test	obs.dif	critical.dif	p-value	difference
AdSi/HeHa-HeSi/HeHa	K-W	33.27586	15.87987	2.3×10^{-10}	TRUE
AdSi/HeHa-HoSi/HeHa	K-W	53.56897	15.87987	$<2\times10^{-16}$	TRUE
HeSi/HeHa-HoSi/HeHa	K-W	20.29310	15.87987	4.2×10^{-6}	TRUE
HoSi/HoHa-HeSi/HoHa	Wilcoxon	-	-	3×10^{-8}	TRUE
Evaluations					
Kruskal-Wallis chi-squared = 75.7342, df = 2, p-value < 2.2e-16					
AdSi/HeHa-HeSi/HeHa	K-W	57.72414	15.87987	$<2\times10^{-16}$	TRUE
AdSi/HeHa-HoSi/HeHa	K-W	29.27586	15.87987	$<2\times10^{-16}$	TRUE
HeSi/HeHa-HoSi/HeHa	K-W	28.44828	15.87987	$<1.3\times10^{-14}$	TRUE
HoSi/HoHa-HeSi/HoHa	Wilcoxon	-	-	3×10^{-8}	TRUE

Table 7.9 Statistical significance of the results for OneMax.

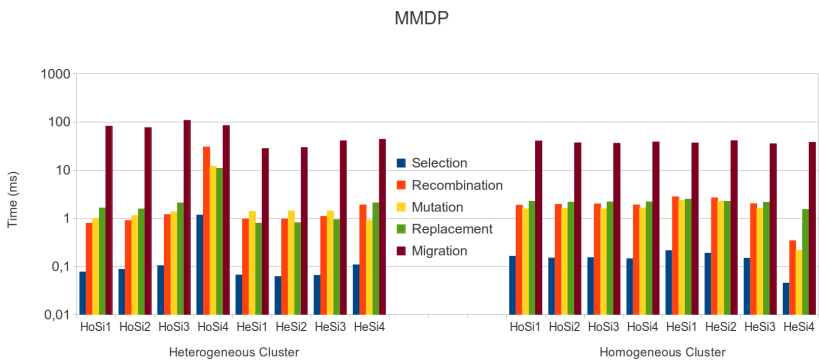


Figure 7.15 Average running time in each stage of the algorithm for the MMDP problem.

buffers: if the slowest machine is sending/receiving, bottlenecks can be propagated (as it can be seen in Figure 7.11).

Results also show how the stages of the algorithms depends on the node of execution. For example, recombination needs more time than mutation in both problems only in the node HeN4. The reason might be the creation of new objects (memory allocation), which in Java and in limited memory (and swapping) requires more time than the iteration of elements previously created (for example, in the mutation). Adapting the sub-population size makes the slower node of HeHa behave in similar way than the other nodes (same time in each stage). Moreover, the size of the individuals affects to some parts of the EA; for example, in OneMax the mutation requires more time than the replacement. However, it must be taken into account that the duration of each part of the algorithm is not related to the time to attain the optimum, but rather to how the diversity and search guidance is maintained in the whole system.

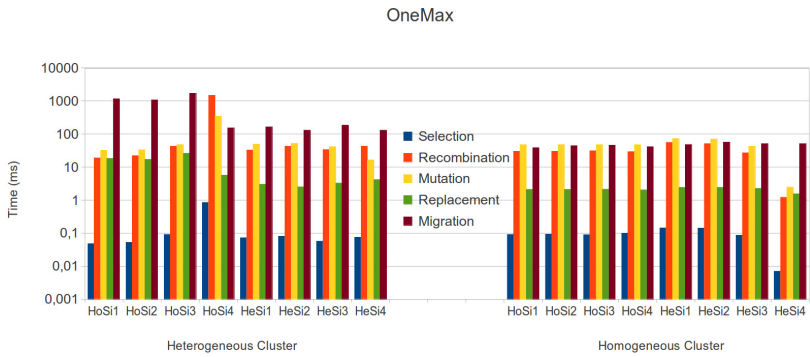


Figure 7.16 Average running time in each stage of the algorithm for the ONEMAX problem.

Table 7.10 Times of the stages of the algorithm for the MMDP problem (in ms).

Heterogeneous Cluster					
Node	Selection	Recombination	Mutation	Replacement	Migration
HoSi HeN1	0.077 ± 0.170	0.788 ± 0.779	1.004 ± 0.187	1.648 ± 20.185	82.458 ± 143.266
HoSi HeN2	0.088 ± 0.190	0.907 ± 0.932	1.145 ± 0.425	1.579 ± 17.907	76.725 ± 126.360
HoSi HeN3	0.105 ± 0.163	1.207 ± 0.927	1.374 ± 0.301	2.108 ± 21.848	108.605 ± 142.633
HoSi HeN4	1.165 ± 1.526	30.445± 59.553	12.221 ± 7.412	10.978 ± 57.135	84.936 ± 0.000
HeSi HeN1	0.067 ± 0.065	0.973 ± 0.403	1.411 ± 0.166	0.790 ± 6.266	28.081 ± 42.169
HeSi HeN2	0.062 ± 0.075	0.973 ± 0.470	1.433 ± 0.265	0.811 ± 7.056	29.667 ± 48.702
HeSi HeN3	0.066 ± 0.108	1.104 ± 0.346	1.435 ± 0.296	0.937 ± 7.072	40.964 ± 40.027
HeSi HeN4	0.109 ± 0.257	1.895 ± 5.611	0.913 ± 0.834	2.085 ± 5.626	43.880 ± 7.535
Homogeneous Cluster					
Node	Selection	Recombination	Mutation	Replacement	Migration
HoSi HoN1	0.163 ± 0.223	1.884 ± 2.386	1.591 ± 0.479	2.254 ± 5.513	40.256 ± 8.726
HoSi HoN2	0.151 ± 0.212	1.952 ± 2.876	1.597 ± 0.574	2.178 ± 4.922	37.110 ± 6.999
HoSi HoN3	0.154 ± 0.206	1.990 ± 3.010	1.591 ± 0.577	2.215 ± 4.743	36.413 ± 5.266
HoSi HoN4	0.146 ± 0.196	1.913 ± 2.697	1.651 ± 1.167	2.194 ± 5.124	38.429 ± 6.192
HeSi HoN1	0.214 ± 0.288	2.800 ± 3.793	2.359 ± 0.691	2.516 ± 4.706	36.972 ± 4.214
HeSi HoN2	0.190 ± 0.252	2.672 ± 3.902	2.277 ± 0.649	2.261 ± 4.546	41.171 ± 9.672
HeSi HoN3	0.148 ± 0.208	2.030 ± 3.161	1.623 ± 0.500	2.164 ± 4.512	35.551 ± 6.132
HeSi HoN4	0.045 ± 0.052	0.345 ± 1.121	0.217 ± 0.142	1.531 ± 4.856	38.106 ± 9.251

Table 7.11 Times of the stages of the algorithm for the OneMax problem (in ms).

Heterogeneous Cluster					
Node	Selection	Recombination	Mutation	Replacement	Migration
HoSi HeN1	0.048 ± 0.043	18.713 ± 13.454	31.984 ± 2.104	18.375 ± 197.676	1172.986 ± 1108.388
HoSi HeN2	0.052 ± 0.051	22.266 ± 22.716	33.553 ± 4.931	17.176 ± 180.580	1085.508 ± 995.382
HoSi HeN3	0.091 ± 1.005	42.634 ± 21.621	47.674 ± 0.546	26.094 ± 252.667	1708.402 ± 1207.925
HoSi HeN4	0.851 ± 0.435	1491.568 ± 1185.723	344.872± 6.634	5.655 ± 16.175	154.019 ±0.000
HeSi HeN1	0.072 ± 0.063	32.917 ± 26.792	49.103 ± 2.655	3.023 ± 27.647	163.479 ±157.172
HeSi HeN2	0.080 ± 0.092	43.001 ± 51.680	52.288 ± 13.210	2.527 ± 21.861	131.063 ±124.404
HeSi HeN3	0.057 ± 0.052	33.951 ± 15.063	41.375 ± 1.707	3.284 ± 30.170	186.467 ±163.906
HeSi HeN4	0.075 ± 0.107	42.443 ± 88.536	16.236 ± 12.028	4.194 ± 33.119	131.135 ±144.359
Homogeneous Cluster					
Node	Selection	Recombination	Mutation	Replacement	Migration
HoSi HoN1	0.091 ± 0.078	29.969 ± 21.459	47.445 ± 2.194	2.073 ± 6.970	38.782 ± 40.369
HoSi HoN2	0.093 ± 0.082	30.119 ± 22.029	47.247 ± 2.146	2.108 ± 7.440	44.303 ± 42.759
HoSi HoN3	0.089 ± 0.080	30.951 ± 21.904	47.103 ± 2.031	2.138 ± 8.006	46.107 ± 47.351
HoSi HoN4	0.098 ± 0.075	29.468 ± 20.876	47.086 ± 1.856	2.043 ± 7.491	41.458 ± 44.970
HeSi HoN1	0.144 ± 0.151	56.124 ± 48.229	72.811 ± 5.177	2.424 ± 9.056	48.165 ±57.798
HeSi HoN2	0.141 ± 0.152	51.226 ± 41.016	70.047 ± 4.152	2.427 ± 10.890	57.152 ±74.177
HeSi HoN3	0.086 ± 0.088	26.932 ± 20.460	42.963 ± 3.935	2.239 ± 8.658	51.014 ±49.648
HeSi HoN4	0.007 ± 0.008	1.215 ± 1.133	2.470 ± 0.098	1.553 ± 10.078	50.498 ± 63.983

7.4 CONCLUSIONS

The SOA paradigm imply to deal with heterogeneous and dynamic environments. The different components of these environment can be adapted to take advantage of their computational resources, so the services that conform a SOEA can also be adapted in consideration. In this chapter, OSGiLiath has been used to present a study on the adaptation of the sub-population sizes of a distributed EA to the computational power of the different nodes of an heterogeneous environment. Different services for migration and algorithm control have been created to be automatically bound and two adaptation schemes (offline and online) that use information of the computational load of the algorithm have been tested. The results of this study reveal some interesting information about the use of heterogeneous systems to develop EAs with the paradigm proposed in this thesis.

Results show that adapting (online or offline) the sub-population size to the computational power of each node in the heterogeneous cluster yields significantly better results in time than keeping the same parameter in all nodes. This advantage is due to the combination of the heterogeneous parameters with the heterogeneity of the machines. On the contrary, the same (heterogeneous) parameter setting in all islands of the homogeneous cluster could not improve the results than considering the same parameter value in all nodes.

Furthermore, changing the sub-population size affects to the services (stages) of the SOEA that, in principle, are independent of this parameter, such as the migration. The sub-population size adaptation is also affected by the problem to solve. Therefore, this should be taken into consideration when dealing with this kind of environments and creating services for them.

In this chapter, as a possible offline parameter setting, we have calculated the computational power of each node proportionally to the average number of generations of the homogeneous parameter set. Moreover, a possible way to adapt online the sub-population sizes has been performed comparing the current generation with the neighbour generation. These results are a promising starting for adapting SOEAs to the performance of each execution node, using more adequate benchmarks or in a dynamic way.

8

GENERATION OF BOTS FOR RTS GAMES USING GENETIC PROGRAMMING

This last week with Fry has been great. Beneath his warm, soft exterior beats the cold, mechanical heart of a robot.

— Bender. I, Roomate. Futurama.

INDEX

The last application to fulfill the OBJECTIVE 4 is to use OSGiLiath to obtain competitive bots for RTS games. With this application, Genetic Programming (explained in Section 3.1.1) will be used to validate if the genericity in evolutionary model explained in Section 3.4.1, can also be adapted to SOA. SOA-EA and OSGiLiath will be used to perform an study on tree depth influence of GP to create competitive bots for RTS games. New individuals will be used and new implementations for mutation and crossover, and services to execute remote environments outside OSGi will be developed with SOA-EA.

8.1 BACKGROUND

RTS Time Strategy (RTS) games are a type of videogame where the play takes action in real time (that is, there are not turns, as in chess). Well-known games of this genre are Age of Empires™ or Warcraft™. In this kind of game the players have units, structures and resources and they have to confront with other players to win battles. Artificial Intelligence (AI) in these games is usually very complex, because they are dealing with a lot of actions and strategies at the same time.

The *Planet Wars* game, presented under the Google AI Challenge 2010[1] has been used by several authors for the study of computational intelligence in RTS games [47, 89, 107]. This is because it is a simplification of the elements that are present in the complex games previously mentioned (only one type of resource and one type of unit).

Although this game has been described in previous works [47, 89, 107], we summarize saying that the objective of the player is to conquer enemy and neutral planets in a space-like simulator. Each player has planets (resources) that produce ships (units) depending of a growth-rate. The player must send these ships to other planets (literally, crashing towards the planet) to conquer them. A player win if he is the owner of all the planets. As requirements, only one second is the limit to calculate next actions (this time windows is called *turn*[2]), and no memory about the previous turns must be used. Figure 8.1 shows a screen capture of the game.

In this chapter we use Genetic Programming (GP) to obtain agents that play Planet Wars game. The reason is to obtain agents without any human knowledge, obtaining the rules to play automatically.The objective of GP is to create functions or programs to solve determined problems. Individual representation is usually in form of a tree, formed by operators (or *primitives*) and variables (*terminals*). These sets are usually fixed and known. The genome size is, therefore, variable, but the maximum size (depth) of the individuals is usually fixed, to avoid high evaluation costs.

We try to solve the next questions:

- Can a tree-generated behaviour of an agent defeat an agent hand-coded by a player with experience and whose parameters have been also optimized?

1 http://planetwars.aichallenge.org/

2 Although in this work we are using this term, note that the game is always performed in real time.

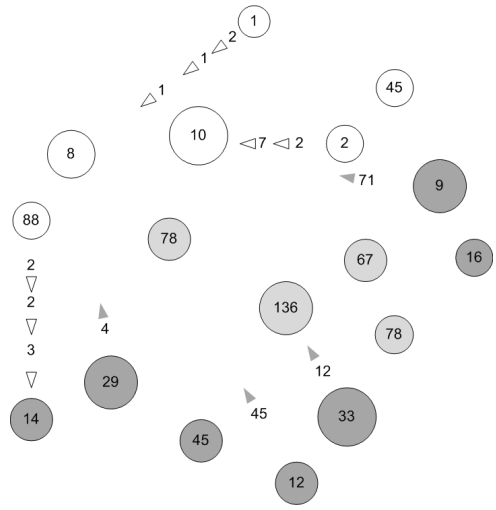


Figure 8.1 Example of execution of the Player Wars game. White planets and ships are owned by the player and dark gray ones are controlled by the enemy. Clear gray are neutral planets (not invaded).

- Can this agent beats a more complicated opponent that is adapted to the environment?
- How does the maximum depth affects the results?

RTS games have been used extensively in the computational intelligence area (see [90] for a review). Among other techniques, Evolutionary Algorithms (EAs) have been widely used in computational intelligence in RTS games [90]. For example, for parameter optimization [44], learning [135] or content generation [101].

One of these types, genetic programming, has been proved as a good tool for developing strategies in games, achieving results comparable to human, or human-based competitors [131]. They also have obtained higher ranking than solvers produced by other techniques or even beating high-ranking humans [40]. GP has also been used in different kind of games, such as boardgames [13], or (in principle) simpler games such as Ms. Pac-Man [16] and Spoof [144] and even in modern video-games such as First Person Shothers (FPS) (for example, Unreal™ [43]).

Planet Wars, the game we are going to use in this chapter, has been used as experimental environment for testing agents in other works. For example, in [107] the authors programmed the behaviour of a *bot* (a computer-controlled player) with a decision tree of 3 levels. Then, the values of these rules were optimized using a genetic algorithm to tune the strategy rates and percentages. Results showed a good performance confronting with other bots provided by the Google AI Challenge. In [47] the authors improved this agent optimizing in different types of maps and selecting the set of optimized parameters depending of the map where the game was taking place, using a tree of 5 levels. These

results outperformed the previous version of the bot with 87% of victories.

In this paper we use GP to create the decision tree, instead of using our own gaming experience to model it, and compare this agent with the two presented before.

8.2 APPLICATION OF SOA-EA

8.2.1 *Identification*

As in previous examples, in the *Problem domain* an *Initializer* of individual and a *FitnessCalculator* service need to be used. The first one generates needs to generate individuals codified as a tree of decisions and operations (*TreeGenome*), and the latter one will integrate the Planet Wars environment to OSGiLiath.

The operators of the *Algorithm Domain* to deal with this new codification of indivudals needs to be created: a Crossover and the Mutation. However, some of the operators previously defined in previous chapter (such as *Parent Selector*) does not need modification. Finally, inside the *Infrastructure domain* services to test each individual (*IndidivualTester*) and convert the codification of the individual to the appropriate codification to different playing environments (*Conversor*) needs to be created.

8.2.2 *Specification*

The proposed agent receives a tree to be executed. The generated tree is a binary tree of expressions formed by two different types of nodes:

- *Decision*: a logical expression formed by a variable, a less than operator ($<$), and a number between 0 and 1. They are the equivalent to the "primitives" in the field of GP.
- *Action*: the leaves of the the tree (therefore, the "terminals"). Each decision is the name of the method to call that indicates to which planet send a percentage of available ships (from 0 to 1) from the planet that executes the tree.

The different variables for the decisions are:

- *myShipsEnemyRatio*: Ratio between the player's ships and enemy's ships.
- *myShipsLandedFlyingRatio*: Ratio between the player's landed and flying ships.

- *myPlanetsEnemyRatio*: Ratio between the number of player's planets and the enemy's ones.
- *myPlanetsTotalRatio*: Ration between the number of player's planet and total planets (neutrals and enemy included)-
- *actualMyShipsRatio*: Ratio between the number of ships in the specific planet that evaluates the tree and player's total ships.
- *actualLandedFlyingRatio*: Ratio between the number of ships landed and flying from the specific planet that evaluates the tree and player's total ships.

The decision list is:

- *Attack Nearest (Neutral | Enemy | NotMy) Planet*: The objective is the nearest planet.
- *Attack Weakest (Neutral | Enemy | NotMy) Planet*: The objective is the planet with less ships.
- *Attack Wealthiest (Neutral | Enemy | NotMy) Planet*: The objective is the planet with higher lower rate.
- *Attack Beneficial (Neutral | Enemy | NotMy) Planet*: The objective is the planet more beneficial, that is the one with growth rate divided by the number of ships.
- *Attack Quickest (Neutral | Enemy | NotMy) Planet*: The objective is the planet with higher facility to conquest: the lowest product between the distance from the planet that executes the tree and the number of the ships in the objective planet.
- *Attack (Neutral | Enemy | NotMy) Base*: The objective is the planet with more ships (that is, the base).
- *Attack Random Planet*.
- *Reinforce Nearest Planet*: Reinforce the nearest player's planet to the planet that executes the tree.
- *Reinforce Base*: Reinforce the player's planet with higher number of ships.
- *Reinforce Wealthiest Planet*: Reinforce the player's planet with higher grown rate.
- *Do nothing*.

Figure 8.2 Example of a generated Java tree.

```
if(myShipsLandedFlyingRatio<0.796)
  if(actualMyShipsRatio<0.201)
    attackWeakestNeutralPlanet(0.481);
  else
    attackNearestEnemyPlanet(0.913);
else
  attackNearestEnemyPlanet(0.819);
```

An example of a possible tree is shown in Figure 8.2. This example tree has a total of 5 nodes, with 2 decisions and 3 actions, and a depth of 3 levels.

The bot behaviour is explained in Figure 8.3.

Figure 8.3 Pseudocode of the proposed agent. The tree is fixed during all the agent's execution

```
tree ← readTree()
while gamenotfinished do
  starts the turn
  calculateGlobalPlanets()
  calculateGlobalRatios()
  for p in PlayerPlanets do
    calculateLocalPlanets(p)
    calculateLocalRatios(p)
    executeTree(p,tree)
  end for
end while
```

A hierarchical fitness (*HierarchichalFitness* implementation) will be used, as proposed in [107]. An individual is better than another if it wins in a higher number of maps. In case of equality of victories, then the individual with more turns to be defeated (i.e. it is stronger) is considered better. The *PlanetWarsFitnessCalculator* will confront each individual to other agents a number of times.

8.2.3 *Implementation and Deployment*

The *TreeGenome* individual is composed of *Decisions* and *Actions* codified as strings coincident with the actions the agent can execute, according the description in previous section. The *Conversor* implementation translates the tree to a string of Java code to be executed in the Planet Wars environment by the *IndividualTester*, using the *Javassist*[3] library to compile the string into executable Java bytecode.

3 http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/

The rest of implementations used are the ones available in OSGiLiath (and previously explained in chapter 6): *ListPopulation*, *DeterministicTournamentSelector*, *NGenerationsStopCriterion* and *DistributedFitness* to execute several simulations at the same time. Besides using Genetic Programming, the flow to use the previous services is the *EvolutionaryAlgorithm* implementation used in previous chapters.

8.3 EXPERIMENTAL SETUP

Sub-tree crossover and 1-node mutation evolutionary operators have been used, following other researchers' proposals that have used these operators obtaining good results [43]. In our case, the mutation randomly changes the decision of a node or mutate the value with a step-size of 0.25 (an adequate value empirically tested). Each configuration is executed 30 times, with a population of 32 individuals and a 2-tournament selector for a pool of 16 parents.

To test each individual during the evolution a battle with a previously created bot is performed in 5 different (but representative) maps provided by Google. The maximum fitness is, therefore 5 victories and 0 turns. Also, as proposed by [107], and due to the noisy fitness effect, in every generation all individuals are re-evaluated.

Two publicly available bots have been chosen for our experiments[4]. The first bot to confront is *GeneBot*, proposed in [107]. This bot was trained using a GA to optimize the 8 parameters that conforms a set of hand-made rules, obtained from an expert human player experience. The second one is an advanced version of the previous, called *Exp-Genebot* (Expert Genebot) [47]. This bot outperformed Genebot widely. Exp-Genebot bot analyses the distribution of the planets of the map to chose a previously optimized set of parameters by a GA. Both bots are the best individual obtained of all runs of their algorithm (not an average one).

After running our algorithm without tree limitation in depth, it has also been executed with the lower and average levels obtained for the best individuals: 3 and 7, respectively, to study if this number has any effect on the results. Table 8.1 summarizes all the parameters used.

After all the executions we have evaluated the obtained best individuals in all runs confronting to the bots in a larger set of maps (the 100 maps provided by Google) to study the behaviour

4 Both can be downloaded from https://github.com/deantares/genebot

Table 8.1 Parameters used in the experiments.

Parameter Name	*Value*
Population size	32
Crossover type	Sub-tree crossover
Crossover rate	0.5
Mutation	1-node mutation
Mutation step-size	0.25
Selection	2-tournament
Replacement	Steady-state
Stop criterion	50 generations
Maximum Tree Depth	3, 7 and unlimited
Runs per configuration	30
Evaluation	Playing versus Genebot [107] and Exp-Genebot [47]
Maps used in each evaluation	map76.txt map69.txt map7.txt map11.txt map26.txt

of the algorithm and how good are the obtained bots in maps that have not been used for training.

8.4 RESULTS

Tables 8.2 and 8.3 summarize all the obtained results of the execution of our EA. These tables also show the average age, depth and number of nodes of the best individuals obtained and also the average population at the end of the run. The average turns rows are calculated only taking into account the individuals with lower victories than 5, because this number is 0 if they have win the five battles.

As can be seen, versus Genebot, the average population fitness is nearest to the optimum than versus Exp-Genebot, even with the lowest depth. Highest permanence in the population is also with the depth of 3 levels. On the contrary, confronting with Exp-Genebot the configuration with unlimited depth achieves better results. This make sense because more decisions should be taken because the enemy can be different in each map.

In the second experiment, we have confronted the 30 bots obtained in each configuration again with Genebot and Exp-Genebot, but in the 100 maps provided by Google. This has been used to validate if the obtained individuals of our method can be competitive in terms of quality in maps not used for evaluation. Results are shown in Table 8.4 and boxplots in Figure 8.4. It can be seen that in average, the bots produced by our algorithm perform equal or better than the best obtained by the previous authors. Note that, even obtaining individuals with maximum fitness (5 victories) that have been kept in the population several generations (as presented before in Tables 8.2 and 8.3) cannot be representative of a extremely good bot in a wider set of maps that have not been used for training. As the distributions are not normal, a Kruskal-Wallis test has been used, obtaining significant differences in turns for the experiment versus Genebot (p-value = 0.0028) and victories in Exp-genebot (p-value = 0.02681). Therefore, there are differences using a maximum depth in the

		Depth 3	*Depth 7*	*Unlimited Depth*
Best Fitness	Victories	**4.933** ± 0.25	4.83 ± 0.53	4.9 ± 0.30
	Turns	244.5 ± 54.44	466 ± 205.44	266.667 ± 40.42
Population Ave. Fitness	Victories	**4.486**± 0.52	4.43 ± 0.07	4.711 ± 0.45
	Turns	130.77± 95.81	139.43 ± 196.60	190.346 ± 102.92
Depth	Best	3 ± 0	5.2 ± 1.78	6.933 ± 4.05
	Population	3 ± 0	5.267 ± 1.8	7.353 ± 3.11
Nodes	Best	7 ± 0	13.667 ± 7.68	22.133 ± 22.21
	Population	7 ± 0	13.818 ± 5.86	21.418 ± 13.81
Age	Best	**8.133** ± 3.95	5.467 ± 2.95	5.066 ± 2.11
	Population	**4.297** ± 3.027	3.247 ± 0.25	3.092 ± 1.27

Table 8.2 Average results obtained from each configuration versus Genebot. Each one has been tested 30 times.

Table 8.3
Average results obtained from each configuration versus Exp-Genebot. Each one has been tested 30 times.

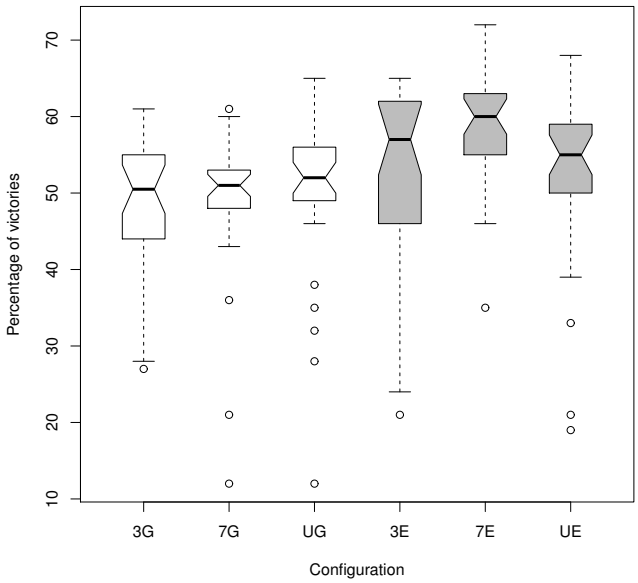
		Depth 3	*Depth 7*	*Unlimited Depth*
Best Fitness	Victories	4.133 ± 0.50	4.2 ± 0.48	**4.4** ± 0.56
	Turns	221.625 ± 54.43	163.667 ± 106.38	123.533 ± 112.79
Population Ave. Fitness	Victories	3.541 ± 0.34	3.689 ± 0.37	**4.043** ± 0.38
	Turns	200.086 ± 50.79	184.076 ± 57.02	159.094 ± 61.84
Depth	Best	3 ± 0	5.2 ± 1.84	6.966 ± 4.44
	Population	3 ± 0	5.216 ± 0.92	6.522 ± 1.91
Nodes	Best	7 ± 0	12.6 ± 6.44	18.466 ± 15.46
	Population	7 ± 0	13.05 ± 3.92	16.337 ± 7.67
Age	Best	4.266 ± 5.01	4.133 ± 4.26	**4.7** ± 4.72
	Population	3.706 ± 0.58	3.727 ± 0.62	**3.889** ± 0.71

Table 8.4
Results confronting the 30 best bots attained from each configuration in the 100 maps each.

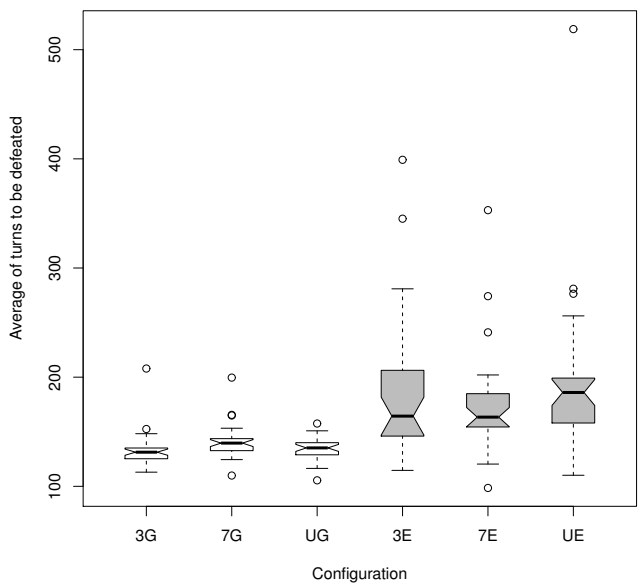
Configuration	*Average maps won*	*Average turns*
Versus Genebot		
Depth 3	47.033 ± 10.001	133.371 ± 16.34
Depth 7	48.9 ± 10.21	**141.386** ± 15.54
Unlimited Depth	50.23 ± 11.40	133.916 ± 10.55
Versus Exp-Genebot		
Depth 3	52.367 ± 13.39	191.051 ± 67.79
Depth 7	**58.867** ± 7.35	174.694± 47.50
Unlimited Depth	52.3 ± 11.57	197.492 ± 72.30

generation of bots. In both configurations, the trees created with 7 levels of depth as maximum have obtained the better results.

To explain why results versus Genebot (a weaker bot than Exp-Genebot) are slightly worse than versus Exp-Genebot, even when the best individuals produced by the GP have higher fitness, we have to analyse how the best individual and the population are being evolved. Figure 8.5 shows that best individual using Genebot reaches the optimal before Exp-Genebot, and also the average population converges quicker. This could lead to over-specialization: that is, the generated bots are over-trained to win in the five maps, and because re-evaluation these individuals are still changing after they have reached the optimal.



(a) Victories



(b) Turns

Figure 8.4 Average of executing the 30 best bots in each configuration (3, 7 and U) versus Genebot (G) and Exp-Genebot (E).

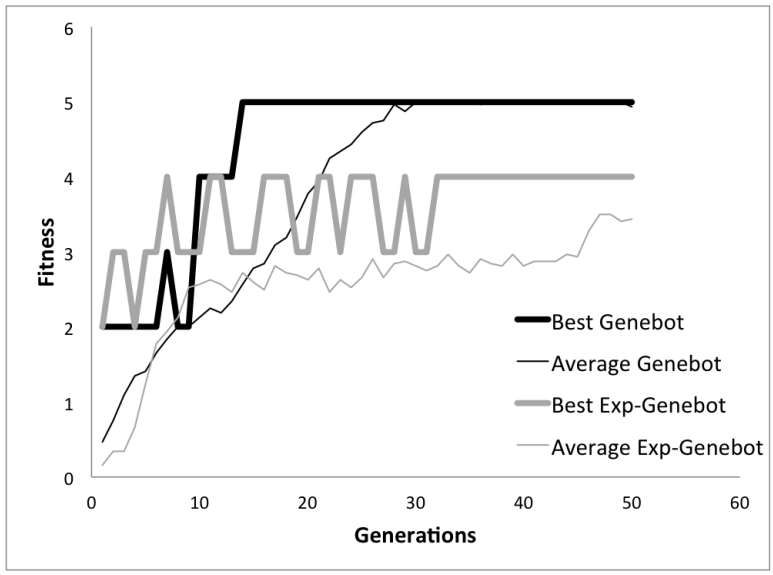


Figure 8.5 Evolution of the best individual and the average population during one run for depth 7 versus Genebot and Exp-Genebot.

8.5 CONCLUSIONS

This chapter presents a Service Oriented Genetic Programming algorithm that generates agents for playing Planet Wars game without using human knowledge. OSGiLiath has been used to obtain relevant results in this field, adding services to manipulate individuals codified as a tree. All services developed follow the genericity in EA development and the SOA requirements. Independence of the individual representation with respect to the existent services to facilitate reuse of existent services have been shown.

Part IV

CONCLUSIONS

9 CONCLUSIONS AND FUTURE WORK

I didn't jump. I took a tiny step,
and there conclusions were.

— Buffy Summers. Phases. Buffy: the Vampire Slayer

INDEX

This thesis studies the viability of the Service Oriented Architecture paradigm to create distributed, dynamic and standards-based environments for EAs. To that end, the concept Service Oriented Evolutionary Algorithm (SOEA) has been presented, and a methodology to develop this kind of algorithms have been proposed. This methodology has been used to validate this paradigm under several scenarios, using specific technologies.

Some conclusions have been obtained while trying to achieve this objective. The first one is that the Evolutionary Algorithms can be successfully migrated to SOA, and therefore, they can take advantage of this paradigm in scenarios of heterogeneity and dynamism. The used SOA technology have a huge impact in several issues (such as the publication mechanisms or transmission time), so the technology should be chosen depending on the necessities to address. In our case, OSGi has helped to save development time, as no specific code has been added to announce the distributed interfaces of the developed services or mechanisms to find and bind these interfaces. Also, it must be remarked that SOA does not force to use only distributed services, thus it can help in development in EAs that can be executed in one machine (locally).

We consider that the SOA paradigm can be applied successfully to EAs to facilitate the integration, distribution, dynamism and development in some scenarios. In particular, the following contributions have been provided with this thesis:

- The Service Oriented Architecture paradigm has been proposed to create distributed, heterogeneous, dynamic and standards-based environments for Evolutionary Algorithms, as it provides mechanisms for interoperability, integration and dynamic control.
- The requirements to develop EAs in the SOA paradigm have been identified.
- These requirements have been taken into account to propose SOA-EA, a methodology that is able to successfully adapt evolutionary algorithms to distributed, heterogeneous, dynamic, standards-based environments.
- Several steps to design all the elements in an EA has been proposed inside this methodology.
- The methodology has been validated using a specific SOA technology: OSGi.
- A SOA-based implementation (OSGiLiath) of distributed, dynamic, standards-based evolutionary algorithms has been able to solve efficiently different problems.
- As an application of this methodology, two different parameter adaptation schemes of island-based EAs to heterogeneous hardware have been proposed, and an algorithm to obtain competent bots for RTS games has been obtained.

9.1 OUTLOOK

The results presented in this thesis can be considered as the starting point to a new research line in automatic adaptation of parameters and operators in dynamic and heterogeneous environments under the SOA paradigm.

Taking into account the dynamic nature of SOA, other adaptive mechanisms to enable or disable services (remotely or locally) depending on some metrics could be created. For example, more experiments to enable or disable different implementations of services depending on the current state of the EA or the execution node, as proposed in this thesis. Different benchmark services to analyse the algorithm can also be used to enable automatic parameter adaptation at runtime. Also, adapting parameters or operators of different nodes entering or exiting the topology, or adapting the parameters to the current load of the system. For example, as the adaptation of the sub-population size to heterogeneous hardware has been proved in this thesis,

other parameters such as migration rate or crossover probability could be adapted to the execution nodes.

Different transmission mechanisms (R-OSGi, JMS, REST, XMMP, among others) can be compared easily, as no modification in the source code of the services is necessary. More experiments about service binding and automatic service composition can be carried out, using different distribution technologies apart from OSGi and Web Services (such as SLP or Zookeeper).

Although this thesis is focused on EAs, the concept of service oriented algorithms can be extended to other meta-heuristic of the field of EC, such as Ant Colony Optimization [108] or Particle Swarm Optimization [86]. Their specific development restrictions can be taken into account to modify SOA-EA to create services for these kind of algorithms in the environments presented in this work.

OSGiLiath will be updated with new modules and services to address new problems. Also, a web portal to centralize services and implementations being offered to the community will be created as a future task. Finally, questionnaires will be proposed to EA practitioners with different skills in programming and different research areas, to validate if this change of paradigm is contributing to enhance their work.

9.2 PUBLICATIONS RELATED WITH THIS THESIS

Some of the results presented in this thesis have been published in several peer reviewed journal and conference proceedings:

- Pablo García-Sánchez, J. González, Pedro A. Castillo, Maribel García Arenas, Juan Julián Merelo Guervós *Service oriented evolutionary algorithms*. Soft Comput. 17(6): 1059–1075 (2013).

- Pablo García-Sánchez, Maria I. García Arenas, Antonio Miguel Mora, Pedro A. Castillo, Carlos Fernandes, Paloma de las Cuevas, Gustavo Romero, Jesús González, Juan Julián Merelo Guervós *Developing services in a service oriented architecture for evolutionary algorithms*. In Proceesings of Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013, Companion Material. p: 1341–1348. ACM, 2013.

- Pablo García-Sánchez, J. González, Pedro A. Castillo, Juan Julián Merelo Guervós, Antonio Miguel Mora, Juan Luís Jiménez Laredo, Maribel García Arenas *A Distributed Service Oriented Framework for Metaheuristics Using a Public Stan-*

dard. In Proceeding of Nature Inspired Cooperative Strategies for Optimization. Studies in Computational Intelligence. p: 211–222. Springer, 2010.

- Pablo García-Sánchez, Antonio Fernández-Ares, Antonio Miguel Mora, Pedro Ángel Castillo, Jesús González and Juan Julián Merelo *Tree depth influence in Genetic Programming for generation of competitive agents for RTS games*. Applications of Evolutionary Computation, EvoApplicatons 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Proceedings. Lecture Notes in Computer Science Springer, 2014. (to appear).

In addition to previous publications, other works related with SOA and their methodologies have been also published during the development of this thesis. The next works have helped to identify some of the requirement in SOA design and implementation (with different technologies used in this thesis):

- Pablo García-Sánchez, Jesús González, Antonio Miguel Mora, Alberto Prieto *Deploying intelligent e-health services in a mobile gateway*. Expert Syst. Appl. 40(4): 1231-1239 (2013).
- Pablo García-Sánchez, Jesús González, Pedro A. Castillo, and Alberto Prieto *Using UN/CEFACT's modelling methodology (UMM) in e-health projects*. In Bio-Inspired Systems: Computational and Ambient Intelligence, 10th International Work Conference on Artificial Neural Networks, IWANN 2009, Salamanca, Spain, June 10-12, 2009. Proceedings, Part I, volume 5517 of Lecture Notes in Computer Science. p: 925–932. Springer, 2009.
- Pablo García-Sánchez, S. González, A. Rivadeneyra, M. P. Palomares, and J. González. *Context-awareness in a service oriented e-health platform*. In José Bravo, Ramón Hervás, and Vladimir Villarreal (editors): Ambient Assisted Living - Third International Workshop, IWAAL 2011, Held at IWANN 2011, Torremolinos-Málaga, Spain, June 8-10, 2011. Proceedings, volume 6693 of Lecture Notes in Computer Science, pages 172-179. Springer, 2011.

With respect to the application of EAs in heterogeneous environments, some lessons learned have been also published in next works:

- Khaled Meri, Maribel García Arenas, Antonio Miguel Mora, Juan Julián Merelo, Pedro Ángel Castillo, Pablo García-Sán-

chez, and Juan Luís Jiménez Laredo. *Cloud-based evolutionary algorithms: An algorithmic study*. Natural Computing, p: 1–13, 2013.

- Pablo García-Sánchez, Juan P. Sevilla, Juan Julián Merelo Guervós, Antonio Miguel Mora, Pedro A. Castillo, Juan Luís Jiménez Laredo, and Francisco Casado. *Pervasive evolutionary algorithms on mobile devices*. In Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living, 10th International Work-Conference on Artificial Neural Networks, IWANN 2009 Workshops, Salamanca, Spain, June 10-12, 2009. Proceedings, Part II, volume 5518 of Lecture Notes in Computer Science, p: 163–170. Springer, 2009.

Finally, the OSGiLiath framework has also been used to compare different fitness functions in Evolutionary Art [62]. Also, during the development of this thesis, other works related with EA applications have been published: such as Evolutionary Robotics [56], video-games bot optimization [44, 47, 48, 107], inventory and route management [42, 45] or document transformation [60, 61].

Following the principles of Open Science, all the work of this thesis has been released using open licenses. OSGiLiath and all experiments presented in this thesis are available under GNU/LGPL V3 License in our GitHub repository `https://github.com/fergunet/osgiliath`.

The LaTeX files that generate this thesis have also been released in `https://github.com/fergunet/osgiliath` under a Creative Commons License. Finally, the web page `http://www.osgiliath.org` describes the steps of development of this thesis: news, publications, awards and documentation.

10 CONCLUSIONES Y TRABAJO FUTURO

INDEX

Esta tesis estudia la viabilidad del paradigma de la Arquitectura Orientada a Servicios (AOS) para crear entornos distribuidos, dinámicos y basados en estándares para Algoritmos Evolutivos. Para ello se ha presentado el concepto Algoritmo Evolutivo Orientado a Servicios (AEOS), junto con una metodología para desarrollar este tipo de algoritmos. Esta metodología ha sido usada para validar este paradigma en distintos escenarios, utilizando tecnologías específicas.

Se han obtenido algunas conclusiones mientras se desarrollaba esta tesis. La primera es que los AEs pueden migrarse con éxito a AOS, y por lo tanto, pueden aprovecharse de este paradigma en escenarios de heterogeneidad y dinamismo. La tecnología SOA usada tiene un gran impacto en diferentes ámbitos (como los mecanismos de publicación y el tiempo de transmisión), por lo tanto, la tecnología a usar debe escogerse dependiendo de las necesidades a abordar. En nuestro caso, OSGi ha servido para ahorrar tiempo de desarrollo, ya que no ha hecho falta código específico para publicar las interfaces de los servicios desarrollados en esta tesis, o los mecanismos para encontrar y enlazar esas interfaces. También es importante remarcar que AOS no fuerza a utilizar servicios distribuidos, por lo que puede ayudar en el desarrollo de AEs que se ejecutan en local.

Consideramos que el paradigma SOA puede aplicarse con éxito a los AEs para facilitar la integración, distribución, dinamismo y desarrollo en escenarios particulares. En concreto, las siguientes contribuciones han sido propuestas en esta tesis:

- Se ha propuesto el paradigma de Arquitectura Orientada a Servicios para crear entornos distribuidos, heterogéneos dinámicos y basados en estándares para Algoritmos Evolutivos, ya que proporciona mecanismos para interoperabilidad, integración y control del dinamismo.

- Se han identificado requisitos para desarrollar AEs en el paradigma AOS.
- Estos requisitos se han tenido en cuenta para proponer SOA-EA, una metodología que es capaz de adaptar con éxito algoritmos evolutivos a entornos distribuidos, heterogéneos, dinámicos y basados en estándares.
- En esta metodología se han propuesto varios pasos para diseñar todos los elementos de un AE.
- Esta metodología se ha validado utilizando una tecnología SOA específica: OSGi.
- Una implementación basada en SOA (OSGiLiath) de algoritmos evolutivos dinámicos y basados en estándares, ha sido capaz de resolver eficientemente diferentes problemas.
- Como aplicación de esta metodología, se han propuesto dos diferentes esquemas de adaptación de parámetros a hardware heterogéneo en AEs basados en islas, junto con un algoritmo para obtener bots competitivos para juegos en tiempo real.

10.1 TRABAJO FUTURO

Los resultados presentados en esta tesis pueden considerarse como el punto de inicio de una nueva linea de investigación en adaptación automática de parámetros y operadores bajo el paradigma AOS.

Teniendo en cuenta la naturaleza dinámica de AOS, se pueden crear otros mecanismos adaptativos para activar o desactivar servicios (remota o localmente) dependiendo de algunas métricas. Por ejemplo, más experimentos para activar o desactivar diferentes implementaciones de servicios dependiendo del estado actual del AE o del nodo de ejecución, como se propone en esta tesis. Diferentes servicios de *benchmarking* para analizar el algoritmo pueden utilizarse para activar adaptación de parámetros en tiempo real. También, adaptando los parámetros u operadores de diferentes nodos que entran o salen de la topología, o adaptando parámetros a la carga actual del sistema. Teniendo en cuenta que en esta tesis se ha propuesto la adaptación del tamaño de las sub-poblaciones a hardware heterogéneo, se pueden adaptar otros parámetros del AE, como la tasa de cruce.

También se pueden comparar fácilmente distintos mecanismos de transmisión (R-OSGi, JMS, REST, XMMP, entre otros), ya que

no hace falta modificación del código fuente de los servicios. Así mismo, se pueden llevar a cabo más experimentos sobre el enlace de servicios y composición automática de servicios, utilizando distintas tecnologías de distribución además de OSGi y Servicios Web (como SLP o Zookeeper).

Aunque esta tesis está enfocada en EAs, el concepto de algoritmos orientados a servicios puede extenderse a otras metaheurísticas del campo de la computación evolutiva, como Ant Colony Optimization [108] o Particle Swarm Optimization [86]. Sus restricciones de desarrollo específicas deben tenerse en cuenta para modificar SOA-EA para crear servicios para este tipo de algoritmos en los entornos presentados en este trabajo.

OSGiLiath se actualizará con nuevos módulos y servicios para tratar nuevos problemas. También, se pretende crear un portal web para centralizar servicios e implementaciones ofrecidas a la comunidad. Finalmente, se propondrán cuestionarios a los desarrolladores de AEs, con distintas destrezas en programación y de distintas áreas de investigación, para validar si este cambio de paradigma está contribuyendo a facilitar su trabajo.

10.2 PUBLICACIONES RELACIONADAS CON ESTA TESIS

Algunos de los resultados presentados en esta tesis han sido publicados anteriormente en la siguientes revistas y actas de congresos revisadas por pares:

- Pablo García-Sánchez, J. González, Pedro A. Castillo, Maribel García Arenas, Juan Julián Merelo Guervós *Service oriented evolutionary algorithms*. Soft Comput. 17(6): 1059–1075 (2013).

- Pablo García-Sánchez, Maria I. García Arenas, Antonio Miguel Mora, Pedro A. Castillo, Carlos Fernandes, Paloma de las Cuevas, Gustavo Romero, Jesús González, Juan Julián Merelo Guervós *Developing services in a service oriented architecture for evolutionary algorithms*. In Proceesings of Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013, Companion Material. p: 1341–1348. ACM, 2013.

- Pablo García-Sánchez, J. González, Pedro A. Castillo, Juan Julián Merelo Guervós, Antonio Miguel Mora, Juan Luís Jiménez Laredo, Maribel García Arenas *A Distributed Service Oriented Framework for Metaheuristics Using a Public Standard*. In Proceeding of Nature Inspired Cooperative Strate-

gies for Optimization. Studies in Computational Intelligence. p: 211–222. Springer, 2010.

- Pablo García-Sánchez, Antonio Fernández-Ares, Antonio Miguel Mora, Pedro Ángel Castillo, Jesús González and Juan Julián Merelo *Tree depth influence in Genetic Programming for generation of competitive agents for RTS games*. Applications of Evolutionary Computation, EvoApplicatons 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Proceedings. Lecture Notes in Computer Science Springer, 2014. (to appear).

Además de las publicaciones anteriores, durante el desarrollo de esta tesis se han publicado otros trabajos relacionados con AOS y sus metodologías. Los siguientes trabajos han ayudado a identificar algunos de los requisitos en el diseño e implementación en AOS (y las diferentes tecnologías utilizadas en esta tesis):

- Pablo García-Sánchez, Jesús González, Antonio Miguel Mora, Alberto Prieto *Deploying intelligent e-health services in a mobile gateway*. Expert Syst. Appl. 40(4): 1231-1239 (2013).

- Pablo García-Sánchez, Jesús González, Pedro A. Castillo, and Alberto Prieto *Using UN/CEFACT's modelling methodology (UMM) in e-health projects*. In Bio-Inspired Systems: Computational and Ambient Intelligence, 10th International Work Conference on Artificial Neural Networks, IWANN 2009, Salamanca, Spain, June 10-12, 2009. Proceedings, Part I, volume 5517 of Lecture Notes in Computer Science. p: 925–932. Springer, 2009.

- Pablo García-Sánchez, S. González, A. Rivadeneyra, M. P. Palomares, and J. González. *Context-awareness in a service oriented e-health platform*. In José Bravo, Ramón Hervás, and Vladimir Villarreal (editors): Ambient Assisted Living - Third International Workshop, IWAAL 2011, Held at IWANN 2011, Torremolinos-Málaga, Spain, June 8-10, 2011. Proceedings, volume 6693 of Lecture Notes in Computer Science, pages 172-179. Springer, 2011.

Con respecto a la aplicación de AEs en entornos heterogéneos, se han aplicado algunas de las lecciones aprendidas en los siguientes trabajos:

- Khaled Meri, Maribel García Arenas, Antonio Miguel Mora, Juan Julián Merelo, Pedro Ángel Castillo, Pablo García-Sánchez, and Juan Luís Jiménez Laredo. *Cloud-based evolution-*

ary algorithms: An algorithmic study. Natural Computing, p: 1–13, 2013.

- Pablo García-Sánchez, Juan P. Sevilla, Juan Julián Merelo Guervós, Antonio Miguel Mora, Pedro A. Castillo, Juan Luís Jiménez Laredo, and Francisco Casado. *Pervasive evolutionary algorithms on mobile devices*. In Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living, 10th International Work-Conference on Artificial Neural Networks, IWANN 2009 Workshops, Salamanca, Spain, June 10-12, 2009. Proceedings, Part II, volume 5518 of Lecture Notes in Computer Science, p: 163–170. Springer, 2009.

Finalmente, el framework OSGiLiath ha sido utilizado para comparar diferentes funciones fitness en Arte Evolutivo [62]. Durante el desarrollo de esta tesis, también se han publicado otros trabajos relacionados con distintas aplicaciones de AEs: como la Robótica Evolutiva [56], optimización de bots en video-juegos [44, 47, 48, 107], administración de inventario y transporte [42, 45] o transformación de documentos [60, 61].

Siguiendo los principios del movimiento de Ciencia Abierta, todo el trabajo de esta tesis se ha liberado utilizando licencias libres. OSGiLiath y todos los experimentos presentados en esta tesis están disponibles en el repositorio de GitHub `https://github.com/fergunet/osgiliath` bajo una licencia GNU/GPL V3.

Los ficheros LaTeX que generan esta tesis también han sido liberados en `https://github.com/fergunet/osgiliath` bajo una licencia Creative Commons. Finalmente, la página web `http://www.osgiliath.org` ha descrito los pasos de desarrollo de esta tesis: noticias, publicaciones, premios y documentación.

Part V

APPENDIX

APPENDIX: OSGI

A

INDEX

A.1 OSGI ARCHITECTURE

To understand how OSGi [109] works and which capabilities could offer to the EA practitioners it is necessary to understand how OSGi is built. OSGi has a layered model that is depicted in Figure A.1. The terms present in this figure are:

- Bundles: OSGi components made by developers. They are a normal *jar* file including Java classes, interfaces and extra files (such as the *Service Descriptions*). They also have additions to the MANIFEST.MF file (a metadata file with information about the *jars*).
- Services: This layer connects bundles in a dynamic way by offering a publish-find-bind model.
- Life-Cycle: The API to install, start, stop, update, and uninstall bundles.
- Modules: This layer defines how a bundle can import and export code (using the MANIFEST.MF file).
- Security: Security aspects are handled in this layer.
- Execution Environment: Defines what methods and classes are available in a specific platform. For example, mobile devices have less Java classes due to performance constraints.

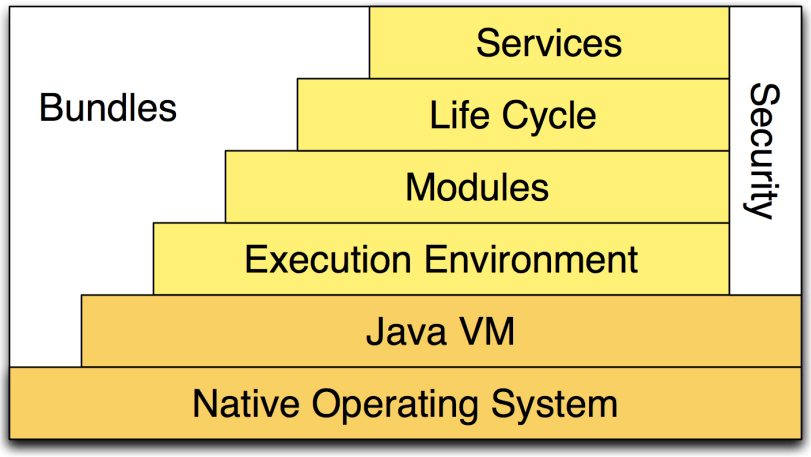


Figure A.1 OSGi layered architecture. Every layer is built from the one just below.

A.2 OSGI CONFIGURATION FILES

With respect to the OSGi layers introduced above, this section details how to use all OSGi capabilities.

OSGi implements a dynamic component model, unlike normal Java environments. Applications or components (also called *bundles*) can be remotely installed, started, stopped, updated or uninstalled on the fly; moreover, the classes and packaging management is specified in detail. The OSGi framework provides APIs for the management of services that are exposed or used by the bundles.

A *bundle* is a file containing compiled and packaged classes and a configuration file. This file indicates which classes are importe or exported by the bundle. Being SOA, the most important concept in OSGi is the *service*. Services allow *bundles* to be dinamically connected, offering a publication-search-connection model. That is, a *bundle* exposes a service by a Java interface (service interface in Figure 4.2), and another bundle (or itself) implements that interface. A third *bundle* can access this service using the exposed interface without having any knowledge of how it is implemented, using the *service registry* (equivalent to service broker of Figure 4.2). Figure A.2 shows an example of the OSGi architecture.

Java programmers are familiar with the *jar* concept. The first difference among a *bundle* and a *jar* is that the first one has a MANIFEST.MF file adapted to be used in OSGi. This file indicates which classes imports or exports the *bundle*. An example can be seen in Figure A.3. This file shows the name of the bundle and its version (this is useful to select specific services), and

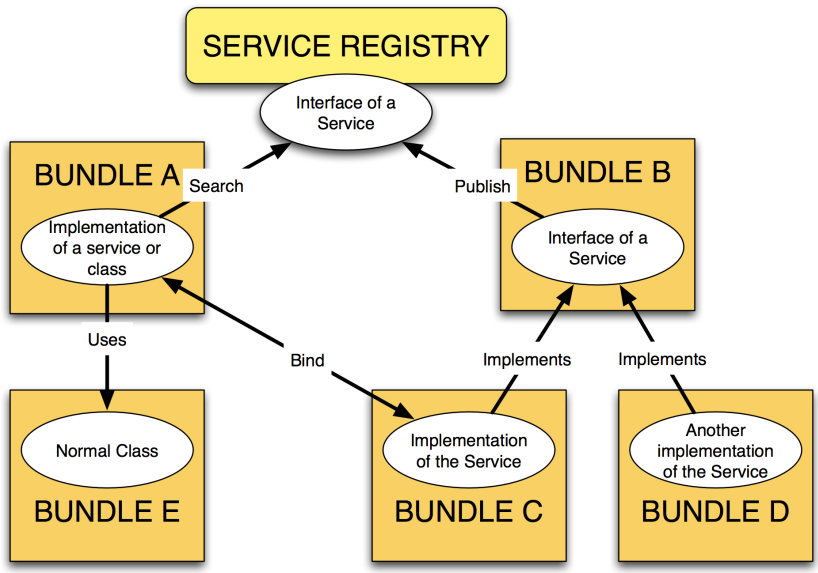


Figure A.2
In OSGi, a service can be implemented by several bundles. Other bundles may chose among this implementations using the service registry. In this figure, Bundles C and D implement a service, and A uses the service registry to use one of them.

the execution environment (that is, the Java Virtual Machine required). Also, this file specifies the XML files of the declarative services (in section *Service-Component*). However, this *bundle* can be used as a normal *jar* outside OSGi.

In normal environments, the creation of a specific implementation of an interface (i.e. *FitnessCalculator*) is done as shown in Figure A.4.

With Declarative Services, the `new ExampleFunction()` part is not used, so if a new implementation is desired no code recompilation is necessary. Figure A.5 shows a declarative service description file, which establishes in execution time which implementation is bound to the interfaces. This example indicates that the implementation of service `FitnessCalculator` is `VRPFitnessCalculator`, but this service is not activated until all their references (other services, like `TransportData`) are also activated. The tag `cardinality` means that at least one service of that kind must exist (the first `1` represents optionality) and the second part (the other `1` indicates the number of different implementations that can be managed: one (`1`) or many ($*$). It is also necessary to create XML files for the rest of services to be exposed (i.e. `TransportData`) . The file where these capabilities are defined is declared in the section `Service-Component` of the MANIFEST.MF file, as can be seen in Figure A.3.

Code in Figure A.6 shows the code for this implementation.

Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: VRP
Bundle-SymbolicName: VRP
Bundle-Version: 1.0.0
Bundle-RequiredExecutionEnvironment: JAVA-1.6
Import-Package: es.ugr.osgiliath,
es.ugr.osgiliath.algorithms,
es.ugr.osgiliath.events,
es.ugr.osgiliath.evolutionary,
es.ugr.osgiliath.evolutionary.basiccomponents.genomes,
es.ugr.osgiliath.evolutionary.basiccomponents.individuals,
es.ugr.osgiliath.evolutionary.elements,
es.ugr.osgiliath.evolutionary.individual,
es.ugr.osgiliath.evolutionary.migrator,
es.ugr.osgiliath.geneticalgorithm.distributed,
es.ugr.osgiliath.problem
Export-Package: es.ugr.osgiliath.vrp,
es.ugr.osgiliath.vrp.individual
Service-Component: OSGI-INF/vrpinitializer.xml,
OSGI-INF/vrpfitnesscalculator.xml,
OSGI-INF/vrpcrossover.xml,
OSGI-INF/vrpmutation.xml

Figure A.3 Example of MANIFEST.MF. This example defines which packages are necessary to activate the bundle and which packages are exported.

```
class EvolutionaryAlgorithm implements Algorithm{
 FitnessCalculator fc;
 //A new instance is bound to a reference
 fc = new ExampleFunction();
}
```

Figure A.4 Normal way to implement an interface in Java.

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
name="VRPFitnessCalculator" >
<implementation class="es.ugr.osgiliath.vrp.VRPFitnessCalculator"/>
<service>
<provide
interface="es.ugr.osgiliath.evolutionary.elements.FitnessCalculator"/>
</service>
<reference bind="setTransportData"
unbind="unsetTransportData"
cardinality="1..1"
interface="es.ugr.osgiliath.vrp.TransportData"
name="TransportData"
policy="static"
/>
<property name="name" type="String"
value="vrpfitnesscalculator"/>
</scr:component>
```

Figure A.5 Service Description. This documents indicates that the implementation of the service `FitnessCalculator` is `VRPFitnessCalculator`, but it can not activate until their references (other services) are activated.

```
class VRPFitnessCalculator implements FitnessCalculator{
 //Other service references,
 TransportData tdata;

 //Methods to bind/unbind each reference
 public TransportData
    setTransportData(TransportData tdata){
  this.tdata = tdata;
 }

 public void
    unsetTransportData(TransportData tdata){
  this.tdata = null;
 }

 //Implementation of the interface method
 List<Fitness> calculateFitness(List<Individual> inds){
         ...
 }
}
```

Figure A.6
Code of the implementation.

A.3 EVENT ADMINISTRATION

The Event Administration in OSGi lets the usage of a blackboard communication architecture where bundles can broadcast or receive events without advertising which bundles are sending or receiving these events.

The steps needed to send events to other bundles are:

- Acquire a reference to the EventAdmin OSGi service (via Declarative Services, for example).
- Pick a topic name for the event (for example "`es/ugr/osgiliath/algorithms/endgeneration`")
- Send the event using the `postEvent` method of EventAdmin, with the topic plus other desired properties

Code to send an event to other bundles is shown in Figure A.7. The programmer specifies the topic String and optional properties to send to other bundles that are listening. The `eventAdmin` variable is a reference to "`org.osgi.service.event. EventAdmin`" service, obtained via Declarative Services or by hand (not showed).

On the other hand, the steps to handle events are:

- Register a service that implements the OSGi EventHandler interface (via Declarative Services or manually).
- Specify in this service the topics to subscribe to. For example, the String "`es/ugr/osgiliath/algorithms/*`" (the * is a wildcard) inside the <`property`> tag in the Service Description.
- Overwrite the handleEvent method of this interface with the desired code.

The code in Figure A.8 shows how to handle events. In this case published the `ExampleService` have been published with the implementation `ExampleImpl`, that is listening under the topic "`es/ugr/osgiliath/algorithms/*`".

Figure A.7 Code to send an event.

```
1 Properties props = new Properties(); //Optional
2 String topic =
3    "es/ugr/osgiliath/algorithms/endgeneration";
4 Event evt = new Event(topic,props);
5 eventAdmin.postEvent(evt);
```

```
class ExampleImpl implements ExampleService,EventHandler{

 public void handleEvent(Event ev){
  if(evt.getTopic().endsWith("endgeneration")){
   // An event with topic
   // "es/ugr/osgiliath/algorithms
   // /endgeneration"
   System.out.println("Generation over");
  else{
   // Other event with topic starts with
   // "es/ugr/osgiliath/algorithms/"
   System.out.println("Other event received");
  }
 }
}
```

Figure A.8 Code to read an event.

B

APPENDIX: OSGILIATH COMPONENTS

This appendix describes the organization of the bundles of OSGiLiath (available at `https://github.com/fergunet/osgiliath`):

- osgiliath: This is the core bundle. It includes all the interfaces common to the algorithms such as *Algorithm*, *AlgorithmParameters* or *Problem*.
- Evolutionary Algorithm: Includes the *EvolutionaryAlgorithm* implementation and interfaces to create the rest of the services that form an EA: *Recombinator* and *Crossover*, *Mutator* and *Mutation*, *StopCriterion* or *FitnessCalculator*. It also provides interfaces for the creation of individuals: *Individual*, *Fitness*, *Gene*, and *Genome*.
- Basic Evolutionary Components: Includes several implementations (the most common ones) of the previous interfaces: *ListPopulation*, *ListIndividual*, *DoubleFitness*, *NGenerationStopCriterion*, *BasicOrderRecombinator*, *UPXListCrossover* and others.
- Binary Problems: Includes implementation of well-known problems, such as OneMax and MMDP: *OneMaxFitnessCalculator*, *MMDPFitnessCalculator* or *BinaryProblemRandomInitializer*.
- Function Problems: Multi-dimensional optimization functions, such as Griegwank or Rastrigin are implemented in this bundle, with their associate Initializers or Fitness Calculators.
- NSGA2: Interfaces and implementations of services for the NSGA2 algorithm.
- OSGiLiART: Service implementation for the creation of Evolutionary Art: *ArtisticIndividual* or *HistogramFitnessCalculator* are examples.
- NoOSGi: Because OSGi allows the separation of source code with the OSGi framework capabilities, this bundle includes Java code to integrate the services without any specific technology (just using basic Object Oriented programming).

- IntelligentManager: An example of how the services can be bound/unbound in real-time. Includes, implementation *IntelligentRandomManager* selects randomly from the available Crossovers, Mutators and Replacers implementations. Also, gatherers for other operations (such as *SelectorGatherer*) and automatic enablers of services (*AsynchronousEnabler*).

BIBLIOGRAPHY

[1] Emile H.L. Aarts and Jan K. Lenstra: *Local search in combinatorial optimization*. Princeton University Press, 1997. (Cited on page 21.)

[2] Enrique Alba, Francisco Almeida, María Blesa, Carlos Cotta, Manuel Díaz, Isabel Dorta, Joaquim Gabarró, Coromoto León, Gabriel Luque, Jordi Petit, Casiano Rodríguez, A. Rojas, and Fatos Xhafa: *Efficient parallel LAN/WAN algorithms for optimization. the MALLBA project*. Parallel Computing, 32(5-6):415–440, 2006. (Cited on page 32.)

[3] Enrique Alba and Gabriel Luque: *Evaluation of parallel metaheuristics*. In Springer (editor): *Parallel Problem Solving from Nature (PPSN)*, volume 4193 of *LNCS*, pages 9–14, 2006. (Cited on page 95.)

[4] Enrique Alba, Gabriel Luque, and Sergio Nesmachnow: *Parallel metaheuristics: recent advances and new trends*. International Transactions in Operational Research, 20(1):1–48, 2013, ISSN 1475-3995. http://dx.doi.org/10.1111/j.1475-3995.2012.00862.x. (Cited on pages 5, 13, and 23.)

[5] Enrique Alba, Antonio J. Nebro, and José M. Troya: *Heterogeneous computing and parallel genetic algorithms*. Journal of Parallel and Distributed Computing, 62(9):1362 – 1385, 2002. (Cited on pages 29 and 68.)

[6] Enrique Alba and Marco Tomassini: *Parallelism and evolutionary algorithms*. Evolutionary Computation, IEEE Transactions on, 6(5):443–462, 2002. (Cited on pages 5, 13, and 23.)

[7] Mine Altunay, Paul Avery, Kent Blackburn, Brian Bockelman, Michael Ernst, Dan Fraser, Robert Quick, Robert Gardner, Sebastien Goasguen, Tanya Levshina, Miron Livny, John McGee, Doug Olson, Ruth Pordes, Maxim Potekhin, Abhishek Rana, Alain Roy, Chander Sehgal, Igor Sfiligoi, Frank WÃ¼rthwein, and Open Sci Grid Executive Board: *A Science Driven Production Cyberinfrastructure-the Open Science Grid*. Journal of GRID Computing, 9(2, Sp. Iss. SI):201–218, UNJUN 2011, ISSN 1570-7873. (Cited on pages 6, 14, 30, and 39.)

[8] Alessia Amelio and Clara Pizzuti: *A genetic algorithm for color image segmentation*. In *Applications of Evolutionary Computation - 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3-5, 2013. Proceedings*, volume 7835 of *Lecture Notes in Computer Science*, pages 314–323. Springer, 2013. (Cited on pages 7 and 15.)

[9] David P Anderson, Eric Korpela, and Rom Walton: *High-performance task distribution for volunteer computing*. In *e-Science and Grid Computing, 2005. First International Conference on*, pages 8–pp. IEEE, 2005. (Cited on page 27.)

[10] M.G. Arenas, Pierre Collet, A.E. Eiben, Márk Jelasity, J. J. Merelo, Ben Paechter, Mike Preuß, and Marc Schoenauer: *A framework for distributed evolutionary algorithms*. In *Parallel Problem Solving from Nature, PPSN VII*, pages 665–675, 2002. (Cited on page 32.)

[11] Ali Arsanjani, Shuvanker Ghosh, Abdul Allam, Tina Abdollah, Sella Ganapathy, and Kerrie Holley: *SOMA: A method for developing service-oriented solutions*. IBM Systems Journal, 47(3):377–396, UL-SEPJUL-SEP 2008, ISSN 0018-8670. (Cited on pages 41 and 46.)

[12] Sean Bechhofer, Iain Buchan, David De Roure, Paolo Missier, John Ainsworth, Jiten Bhagat, Philip Couch, Don Cruickshank, Mark Delderfield, Ian Dunlop, Matthew Gamble, Danius Michaelides, Stuart Owen, David Newman, Shoaib Sufi, and Carole Goble: *Why linked data is not enough for scientists*. Future Generation Computer Systems, 29(2):599 – 611, 2013, ISSN 0167-739X. http://www.sciencedirect.com/science/article/pii/S0167739X11001439, Special section: Recent advances in e-Science. (Cited on page 38.)

[13] Amit Benbassat and Moshe Sipper: *Evolving both search and strategy for reversi players using genetic programming*. pages 47–54, 2012. (Cited on page 113.)

[14] Christopher A. Bohn and Gary B. Lamont: *Load balancing for heterogeneous clusters of PCs*. Future Generation Computer Systems, 18(3):389 – 400, 2002, ISSN 0167-739X. http://www.sciencedirect.com/science/article/pii/S0167739X01000589. (Cited on page 29.)

[15] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Nielsen, Satish Thatte,

and Dave Winer: *Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000*, 2011. http://www.w3.org/TR/SOAP. (Cited on page 42.)

[16] Matthias F. Brandstetter and Samad Ahmadi: *Reactive control of ms. pac man using information retrieval based on genetic programming*. pages 250–256, 2012. (Cited on page 113.)

[17] Pedro A. Castillo, Pablo García-Sánchez, Maribel G. Arenas, Antonio M. Mora, Gustavo Romero, and Juan Julián Merelo: *Using SOAP and REST web services as communication protocol for distributed evolutionary computation*. International Journal of Computers and Technology, 10:1659–1677, 2013, ISSN 2277-3061. (Cited on page 57.)

[18] Pedro A. Castillo, Víctor Rivas, Juan Julián Merelo, Jesús Gonzalez, Alberto Prieto, and Gustavo Romero: *G-Prop-II: global optimization of multilayer perceptrons using GAs*. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages –2027 Vol. 3, 1999. (Cited on pages 6, 15, and 21.)

[19] Ethan Cerami: *Web Services Essentials*. O'Reilly, 2002. (Cited on pages 42 and 43.)

[20] Peter Cowling, Graham Kendall, and Eric Soubeiga: *A hyperheuristic approach to scheduling a sales summit*. In *Practice and Theory of Automated Timetabling III*, pages 176–190. Springer, 2001. (Cited on page 22.)

[21] Simon J. Cox, Matt J. Fairman, Gang Xue, Jasmin L. Wason, and Andy J. Keane: *The grid: Computational and data resource sharing in engineering optimisation and design search*. In *30th International Workshops on Parallel Processing (ICPP 2001 Workshops), 3-7 September 2001, Valencia, Spain*, pages 207–212. IEEE Computer Society, 2001. (Cited on pages 26 and 47.)

[22] J. Albert Cruz, Juan Julián Merelo Guervós, Antonio M. Mora García, and Paloma de las Cuevas: *Adapting evolutionary algorithms to the concurrent functional language erlang*. In *Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion*, GECCO '13 Companion, pages 1723–1724, New York, NY, USA, 2013. ACM, ISBN 978-1-4503-1964-5. http://doi.acm.org/10.1145/2464576.2480782. (Cited on page 30.)

[23] Robert Daigneau: *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services* . Addison-Wesley Professional. 1 Ed. ISBN 978-0321544209, 2011. (Cited on page 44.)

[24] Charles Darwin: *On the Origin of Species by Means of Natural Selection*. Murray, London, 1859. (Cited on page 19.)

[25] Susan B. Davidson and Juliana Freire: *Provenance and scientific workflows: challenges and opportunities*. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1345–1350. ACM, 2008. (Cited on page 43.)

[26] Richard Dawkins: *The selfish gene*. Oxford university press, 2006. (Cited on page 22.)

[27] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan: *A fast and elitist multiobjective genetic algorithm: NSGA-II*. IEEE Transactions on Evolutionary Computation, 6(2):182–197, PRAPR 2002, ISSN 1089-778X. (Cited on page 64.)

[28] Julián Domínguez and Enrique Alba: *HydroCM: A hybrid parallel search model for heterogeneous platforms*. In El Ghazali Talbi (editor): *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*, pages 219–235. Springer Berlin Heidelberg, 2013. (Cited on page 29.)

[29] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet: *The LINPACK benchmark: Past, present, and future.* Concurrency and Computation: Practice and Experience, 15:2003, 2003. (Cited on page 29.)

[30] Alan Dorin: *Aesthetic selection and the stochastic basis of art, design and interactive evolutionary computation*. In *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013*, pages 311–318. ACM, 2013. (Cited on pages 7 and 15.)

[31] Bernabé Dorronsoro and Pascal Bouvry: *Cellular genetic algorithms without additional parameters*. The Journal of Supercomputing, 63:816–835, 2013, ISSN 0920-8542. (Cited on page 24.)

[32] Juan José Durillo, Antonio J. Nebro, and Enrique Alba: *The jmetal framework for multi-objective optimization: Design and architecture*. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010. (Cited on page 32.)

[33] Agoston E. Eiben: *Evolutionary computing and autonomic computing: Shared problems, shared solutions?* In *Self-star Properties in Complex Information Systems*, pages 36–48. Springer, 2005. (Cited on page 21.)

[34] Agoston E. Eiben and Thomas Bäck: *Empirical investigation of multiparent recombination operators in evolution strategies*. Evolutionary Computation, 5(3):347–365, 1997. (Cited on page 31.)

[35] Agoston E. Eiben, Zbigniew Michalewicz, Marc Schoenauer, and James E. Smith: *Parameter control in evolutionary algorithms*. In Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz (editors): *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 19–46. Springer, 2007, ISBN 978-3-540-69431-1. (Cited on pages 27 and 55.)

[36] Agoston E. Eiben and Selmar K. Smit: *Evolutionary algorithm parameters and methods to tune them*. In Youssef Hamadi, Eric Monfroy, and Frédéric Saubion (editors): *Autonomous Search*, pages 15–36. Springer Berlin Heidelberg, 2012, ISBN 978-3-642-21433-2. http://dx.doi.org/10.1007/978-3-642-21434-9_2. (Cited on pages 5, 13, and 27.)

[37] Agoston E. Eiben and James E. Smith: *Introduction*. Springer, 2003. (Cited on page 21.)

[38] Agoston E. Eiben and James E. Smith: *Introduction to Evolutionary Computing*, chapter What is an Evolutionary Algorithm? Springer, 2003. (Cited on pages 4, 12, 19, and 20.)

[39] Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz: *Parameter control in evolutionary algorithms*. Evolutionary Computation, IEEE Transactions on, 3(2):124–141, 1999. (Cited on page 27.)

[40] Achiya Elyasaf, Ami Hauptman, and Moshe Sipper: *Evolutionary design of freecell solvers*. IEEE Transactions on Computational Intelligence and AI in Games, 4(4):270–281, 2012. (Cited on page 113.)

[41] Toshio Endo, Akira Nukada, Satoshi Matsuoka, and Naoya Maruyama: *LINPACK evaluation on a supercomputer with heterogeneous accelerators*. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–8, 2010. (Cited on page 29.)

[42] Anna Esparcia-Alcázar, Manuel Cardós, Juan Julián Merelo Guervós, Anais Martínez-García, Pablo García-Sánchez, Eva Alfaro-Cid, and Ken Sharman: *Evita: An integral evolutionary methodology for the inventory and transportation problem*. In *Bio-inspired Algorithms for the Vehicle Routing Problem*, volume 161 of *Studies in Computational Intelligence*, pages 151–172. Springer, 2009. (Cited on pages 6, 15, 21, 129, and 135.)

[43] Anna I. Esparcia-Alcázar and Jaroslav Moravec: *Fitness approximation for bot evolution in genetic programming*. Soft Computing, 17(8):1479–1487, 2013, ISSN 1432-7643. http://dx.doi.org/10.1007/s00500-012-0965-7. (Cited on pages 6, 15, 113, and 117.)

[44] Anna Isabel Esparcia-Alcázar, Ana Isabel Martínez García, Antonio Mora García, Juan Julián Merelo Guervós, and Pablo García-Sánchez: *Controlling bots in a first person shooter game using genetic algorithms*. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010. (Cited on pages 113, 129, and 135.)

[45] Anna Isabel Esparcia-Alcázar, Anaís Martínez-García, Pablo García-Sánchez, Juan Julián Merelo Guervós, and Antonio Miguel Mora: *Towards a multiobjective evolutionary approach to inventory and routing management in a retail chain*. In *IEEE Congress on Evolutionary Computation*, pages 3166–3173. IEEE, 2013, ISBN 978-1-4799-0452-5, 978-1-4799-0453-2. (Cited on pages 129 and 135.)

[46] Carlos Fernandes and Agostinho Rosa: *Self-regulated population size in evolutionary algorithms*. In *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *Lecture Notes in Computer Science*, pages 920–929. Springer Berlin Heidelberg, 2006, ISBN 978-3-540-38990-3. http://dx.doi.org/10.1007/11844297_93. (Cited on pages 6, 14, and 29.)

[47] Antonio Fernández-Ares, Pablo García-Sánchez, Antonio Miguel Mora, and Juan J. Merelo Guervós: *Adaptive bots for real-time strategy games via map characterization*. In *2012 IEEE Conference on Computational Intelligence and Games, CIG 2012, Granada, Spain, September 11-14, 2012*, pages 417–721. IEEE, 2012, ISBN 978-1-4673-1193-9. (Cited on pages 112, 113, 117, 118, 129, and 135.)

[48] Antonio Fernández-Ares, Antonio Miguel Mora, Juan Julián Merelo Guervós, Pablo García-Sánchez, and Carlos

Fernandes: *Optimizing player behavior in a real-time strategy game using evolutionary algorithms*. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2011, New Orleans, LA, USA, 5-8 June, 2011*, pages 2017–2024. IEEE, 2011. (Cited on pages 21, 129, and 135.)

[49] Roy T. Fielding and Richard N. Taylor: *Principled Design of the Modern Web Architecture*. ACM Transactions on Internet Technology (TOIT) (New York: Association for Computing Machinery) 2 (2): 115-150, 2002. (Cited on page 43.)

[50] Erwin Folmer and Joris Bastiaans: *Methods for Design of Semantic Message-Based B2B Interaction Standards*, volume Enterprise Interoperability III. 2008. (Cited on page 46.)

[51] Ian Foster: *Globus Toolkit version 4: Software for service-oriented systems*. In Jin, H and Reed, D and Jiang, W (editor): *Network and Parallel Computing Proceedings*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13, 2005. (Cited on page 39.)

[52] Ian Foster: *Service-oriented science*. Science, 308(5723):814, 2005. (Cited on pages 4, 6, 12, 15, 38, and 47.)

[53] Christian Gagné and Marc Parizeau: *Genericity in evolutionary computation software tools: Principles and case-study*. International Journal on Artificial Intelligence Tools, 15(2):173, 2006. (Cited on page 31.)

[54] Juan Francisco Garamendi and José Luis Bosque: *Parallel implementation of evolutionary strategies on heterogeneous clusters with load balancing*. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 8 pp., april 2006. (Cited on pages 28 and 29.)

[55] José García-Nieto, Francisco Chicano, and Enrique Alba: *Remote Optimization Service*, pages 443–456. John Wiley and Sons, Inc., 2008. (Cited on page 46.)

[56] Pablo García-Sánchez, A. E. Eiben, Evert Haasdijk, Berend Weel, and Juan Julián Merelo Guervós: *Testing diversity-enhancing migration policies for hybrid on-line evolution of robot controllers*. In *Applications of Evolutionary Computation - EvoApplications 2012: EvoCOMNET, EvoCOMPLEX, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoNUM, EvoPAR, EvoRISK, EvoSTIM, and EvoSTOC, Málaga, Spain, April 11-13, 2012, Proceedings*, volume 7248 of *Lecture Notes in Computer Science*, pages 52–62. Springer, 2012. (Cited on pages 6, 14, 15, 21, 25, 70, 129, and 135.)

[57] Pablo García-Sánchez, Jesú González, Antonio Miguel Mora, and Alberto Prieto: *Deploying intelligent e-health services in a mobile gateway*. Expert Syst. Appl., 40(4):1231–1239, 2013. (Cited on pages 45 and 71.)

[58] Pablo García-Sánchez, Jesús González, Pedro A. Castillo, and Alberto Prieto: *Using un/cefact's modelling methodology (umm) in e-health projects*. In *Bio-Inspired Systems: Computational and Ambient Intelligence, 10th International Work-Conference on Artificial Neural Networks, IWANN 2009, Salamanca, Spain, June 10-12, 2009. Proceedings, Part I*, volume 5517 of *Lecture Notes in Computer Science*, pages 925–932. Springer, 2009. (Cited on pages 45 and 46.)

[59] Pablo García-Sánchez, S. González, A. Rivadeneyra, M. P. Palomares, and J. González: *Context-awareness in a service oriented e-health platform*. In José Bravo, Ramón Hervás, and Vladimir Villarreal (editors): *Ambient Assisted Living - Third International Workshop, IWAAL 2011, Held at IWANN 2011, Torremolinos-Málaga, Spain, June 8-10, 2011. Proceedings*, volume 6693 of *Lecture Notes in Computer Science*, pages 172–179. Springer, 2011, ISBN 978-3-642-21302-1. (Cited on page 45.)

[60] Pablo García-Sánchez, Juan J. Merelo Guervós, Pedro A. Castillo, Jesús González, Juan Luís Jiménez Laredo, Antonio Mora García, and Maria I. García Arenas: *Evolution of xpath lists for document data selection*. In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Günter Rudolph (editors): *Parallel Problem Solving from Nature - PPSN XI, 11th International Conference, Kraków, Poland, September 11-15, 2010. Proceedings, Part II*, volume 6239 of *Lecture Notes in Computer Science*, pages 341–350. Springer, 2010. (Cited on pages 129 and 135.)

[61] Pablo García-Sánchez, Juan Julián Merelo Guervós, Juan Luís Jiménez Laredo, Antonio Mora García, and Pedro A. Castillo: *Evolving xslt stylesheets for document transformation*. In *Parallel Problem Solving from Nature - PPSN X, 10th International Conference Dortmund, Germany, September 13-17, 2008, Proceedings*, volume 5199 of *Lecture Notes in Computer Science*, pages 1021–1030. Springer, 2008. (Cited on pages 6, 15, 22, 129, and 135.)

[62] Pablo García-Sánchez, Juan Julián Merelo, Daniel Calandria, Ana Belén Pelegrina, Roberto Morcillo, Fruela Palacio, and Rubén H. García-Ortega: *Testing the differences of*

using rgb and hsv histograms during evolution in evolutionary art. In *Proceedings of the International Conference in Evolutionary Computation Theory and Applications 2013*, 2013. (Cited on pages 6, 15, 21, 129, and 135.)

[63] Pablo García-Sánchez, Juan P. Sevilla, Juan Julián Merelo Guervós, Antonio Miguel Mora, Pedro A. Castillo, Juan Luís Jiménez Laredo, and Francisco Casado: *Pervasive evolutionary algorithms on mobile devices*. In *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living, 10th International Work-Conference on Artificial Neural Networks, IWANN 2009 Workshops, Salamanca, Spain, June 10-12, 2009. Proceedings, Part II*, volume 5518 of *Lecture Notes in Computer Science*, pages 163–170. Springer, 2009. (Cited on pages 6, 14, and 25.)

[64] David E. Goldberg, Kalyanmoy Deb, and Jeffrey Horn: *Massive multimodality, deception, and genetic algorithms*. In R. Männer and B. Manderick (editors): *Parallel Problem Solving from Nature, 2*, pages 37–48, Amsterdam, 1992. Elsevier Science Publishers, B. V. (Cited on page 80.)

[65] David E. Goldberg and John H. Holland: *Genetic algorithms and machine learning*. Machine learning, 3(2):95–99, 1988. (Cited on page 21.)

[66] Yiyuan Gong and Alex Fukunaga: *Distributed island-model genetic algorithms using heterogeneous parameter settings*. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2011, New Orleans, LA, USA, 5-8 June, 2011*, pages 820–827. IEEE, 2011. (Cited on page 30.)

[67] Yiyuan Gong, Morikazu Nakamura, and Shiro Tamaki: *Parallel genetic algorithms on line topology of heterogeneous computing resources*. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1447–1454, New York, NY, USA, 2005. ACM. (Cited on page 29.)

[68] Jesús González, Ignacio Rojas, Julio Ortega, Héctor Pomares, Francisco Javier Fernandez, and Antonio F Díaz: *Multiobjective evolutionary optimization of the size, shape, and position parameters of radial basis function networks for function approximation*. Neural Networks, IEEE Transactions on, 14(6):1478–1495, 2003. (Cited on page 21.)

[69] Juan Julián Merelo Guervós: *Fluid evolutionary algorithms*. In *Proceedings of the IEEE Congress on Evolutionary Computa-*

tion, CEC 2010, Barcelona, Spain, 18-23 July 2010, pages 1–8. IEEE, 2010. (Cited on page 26.)

[70] Juan Julián Merelo Guervós, Antonio Mora García, Juan Luís Jiménez Laredo, Juan Lupión, and Fernando Tricas: *Browser-based distributed evolutionary computation: performance and scaling behavior*. In *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007, Companion Material*, pages 2851–2858. ACM, 2007. (Cited on pages 27 and 33.)

[71] Juan Julián Merelo Guervós, Antonio Miguel Mora, Pedro A. Castillo, Juan Luís Jiménez Laredo, Lourdes Araujo, Ken Sharman, Anna Esparcia-Alcázar, Eva Alfaro-Cid, and Carlos Cotta: *Testing the intermediate disturbance hypothesis: Effect of asynchronous population incorporation on multi-deme evolutionary algorithms*. In *Parallel Problem Solving from Nature - PPSN X, 10th International Conference Dortmund, Germany, September 13-17, 2008, Proceedings*, volume 5199 of *Lecture Notes in Computer Science*, pages 266–275. Springer, 2008, ISBN 978-3-540-87699-1. (Cited on pages 88 and 90.)

[72] George Harik, Erick Cantú-Paz, David E. Goldberg, and Brad L. Miller: *The gambler's ruin problem, genetic algorithms, and the sizing of populations*. Evolutionary Computation, 7(3):231–253, September 1999, ISSN 1063-6560. http://dx.doi.org/10.1162/evco.1999.7.3.231. (Cited on pages 5, 14, and 29.)

[73] Paul K. Harmer, Michael A. Temple, Mark A. Buckner, and Ethan Farquhar: *Using differential evolution to optimize 'learning from signals' and enhance network security*. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011*, pages 1811–1818. ACM, 2011. (Cited on pages 7 and 15.)

[74] Randy Heffner, Mike Gilpin, and Steven Kesler: *SOA Plays An Important Role In A Telco Merger*, 2013. Available at: http://www.forrester.com/SOA+Plays+An+Important+Role+In+A+Telco+Merger/fulltext/-/E-RES103261. (Cited on page 37.)

[75] Randy Heffner, Gene Leganza, and Lauren Blackburn: *SOA Adoption 2010: Still Important, Still Strong*, 2010. Available at: http://www.forrester.com/SOA+Adoption+2010+Still+Important+Still+Strong/fulltext/-/E-RES59058. (Cited on page 37.)

[76] Quoc Thuan Ho, Yew Soon Ong, and Wentong Cai: *"gridifying" aerodynamic design problem using GridRPC*. In *Grid and Cooperative Computing*, volume 3032 of *Lecture Notes in Computer Science*, pages 83–90. Springer Berlin Heidelberg, 2004. (Cited on page 47.)

[77] Birgit Hofreiter, Christian Huemer, Philipp Liegl, Rainer Schuster, and Marco Zapletal: *UN/CEFACT'S modeling methodology (UMM): A UML profile for B2B e-commerce*, volume 4231 LNCS. 2006. (Cited on page 46.)

[78] John H. Holland: *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975. (Cited on page 21.)

[79] Stephan Hutterer, Michael Affenzeller, and Franz Auinger: *Evolutionary algorithm based control policies for flexible optimal power flow over time*. In *Applications of Evolutionary Computation - 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3-5, 2013. Proceedings*, volume 7835 of *Lecture Notes in Computer Science*, pages 152–161. Springer, 2013. (Cited on pages 7 and 15.)

[80] Hiroaki Imade, Ryohei Morishita, Isao Ono, Norihiko Ono, and Masahiro Okamoto: *A grid-oriented genetic algorithm framework for bioinformatics*. New Gen. Comput., 22(2):177–186, January 2004. (Cited on pages 27 and 47.)

[81] Ejaz Jamil: *White Paper: What really is SOA. A comparison with Cloud Computing, Web 2.0, SaaS, WOA, Web Services, PaaS and others.* , 2009. Available at http://soalib.com/doc/whitepaper/SoalibWhitePaper_SOAJargon.pdf. (Cited on pages xxi and 50.)

[82] Márk Jelasity, Wojtek Kowalczyk, and Maarten Van Steen: *Newscast computing*. Technical report, Technical Report IR-CS-006, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, 2003. (Cited on page 26.)

[83] Zhuoan Jiao, Jasmin L. Wason, Wenbin Song, Fenglian Xu, M. Hakki Eres, Andy J. Keane, and Simon J. Cox: *Databases, workflows and the grid in a service oriented environment*. In *Euro-Par 2004 Parallel Processing, 10th International Euro-Par Conference, Pisa, Italy, August 31-September 3, 2004, Proceedings*, volume 3149 of *Lecture Notes in Computer Science*, pages 972–979. Springer, 2004. (Cited on page 47.)

[84] Matjaz B. Juric: *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition*. Packt Publishing, 2006, ISBN 1904811817. (Cited on page 43.)

[85] Michael Kampouridis, Abdullah Alsheddy, and Edward P. K. Tsang: *On the investigation of hyper-heuristics on a financial forecasting problem*. Ann. Math. Artif. Intell., 68(4):225–246, 2013. (Cited on pages 7 and 15.)

[86] James Kennedy and Russell Eberhart: *Particle swarm optimization*. Volume 4, pages 1942–1948, 1995. (Cited on pages 127 and 133.)

[87] John R. Koza: *Genetically breeding populations of computer programs to solve problems in artificial intelligence*. In *Tools for Artificial Intelligence, 1990.,Proceedings of the 2nd International IEEE Conference on*, pages 819–827, 1990. (Cited on page 22.)

[88] Natalio Krasnogor, BP Blackburne, Edmund K Burke, and Jonathan D Hirst: *Multimeme algorithms for protein structure prediction*. In *Parallel Problem Solving from Nature-PPSN VII*, pages 769–778. Springer, 2002. (Cited on page 22.)

[89] Raúl Lara-Cabrera, Carlos Cotta, and Antonio J. Fernández-Leiva: *A procedural balanced map generator with self-adaptive complexity for the real-time strategy game planet wars*. In *EvoApplications*, volume 7835 of *Lecture Notes in Computer Science*, pages 274–283. Springer, 2013, ISBN 978-3-642-37191-2. (Cited on page 112.)

[90] Raúl Lara-Cabrera, Carlos Cotta, and Antonio J. Fernández-Leiva: *A review of computational intelligence in rts games*. In *FOCI*, pages 114–121. IEEE, 2013. (Cited on page 113.)

[91] Juan Luís J. Laredo, Carlos Fernandes, Juan Julián Merelo, and Christian Gagné: *Improving genetic algorithms performance via deterministic population shrinkage*. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 819–826, New York, NY, USA, 2009. ACM, ISBN 978-1-60558-325-9. http://doi.acm.org/10.1145/1569901.1570014. (Cited on pages 5, 14, and 29.)

[92] Juan Luís Jiménez Laredo, Agoston E. Eiben, Maarten van Steen, and Juan Julián Merelo Guervós: *Evag: a scalable peer-to-peer evolutionary algorithm*. Genetic Programming and

Evolvable Machines, 11(2):227–246, 2010. (Cited on pages 5, 13, and 25.)

[93] Pedro Larrañaga and Jose A. Lozano: *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer, 2002. (Cited on page 22.)

[94] Coromoto León, Gara Miranda, and Carlos Segura: *Metco: A parallel plugin-based framework for multi-objective optimization*. International Journal on Artificial Intelligence Tools, 18(4):569–588, 2009. (Cited on page 32.)

[95] Xuekang Li, Xiaohong Hao, Yi Chen, Muhao Zhang, and Bei Peng: *Multi-objective optimizations of structural parameter determination for serpentine channel heat sink*. In *Applications of Evolutionary Computation - 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3-5, 2013. Proceedings*, volume 7835 of *Lecture Notes in Computer Science*, pages 449–458. Springer, 2013. (Cited on pages 7 and 15.)

[96] Dudy Lim, Yew Soon Ong, Yaochu Jin, Bernhard Sendhoff, and Bu Sung Lee: *Efficient hierarchical parallel genetic algorithms using grid computing*. Future Generation Computer Systems, 23(4):658 – 670, 2007. (Cited on pages 27 and 47.)

[97] Jung Eun Lim, O Hoon Choi, and Doo Kwon Baik: *An evaluation method for dynamic combination among OSGi bundles based on service gateway capability*. IEEE Transactions on Consumer Electronics, 54(4):1698 –1704, november 2008, ISSN 0098-3063. (Cited on page 70.)

[98] Fernando G. Lobo and Cláudio F. Lima: *Adaptive population sizing schemes in genetic algorithms*. In Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz (editors): *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 185–204. Springer Berlin Heidelberg, 2007, ISBN 978-3-540-69431-1. http://dx.doi.org/10.1007/978-3-540-69432-8_9. (Cited on pages 6, 14, and 29.)

[99] Bertram Ludascher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao: *Scientific workflow management and the kepler system*. Concurrency and Computation: Practice and Experience, 18(10):1039–1065, 2006, ISSN 1532-0634. http://dx.doi.org/10.1002/cpe.994. (Cited on page 43.)

[100] Sean Luke *et al.*: *ECJ: A Java-based Evolutionary Computation and Genetic Programming Research System*, 2009. Available at http://www.cs.umd.edu/projects/plus/ec/ecj. (Cited on page 32.)

[101] Tobias Mahlmann, Julian Togelius, and Georgios N. Yannakakis: *Spicing up map generation*. In *Proceedings of the 2012t European conference on Applications of Evolutionary Computation*, EvoApplications'12, pages 224–233, Berlin, Heidelberg, 2012. Springer-Verlag, ISBN 978-3-642-29177-7. http://dx.doi.org/10.1007/978-3-642-29178-4_23. (Cited on page 113.)

[102] Ross Mason: *How rest replaced soap on the web: What it means to you*, 2011. Available at: http://www.infoq.com/articles/rest-soap/. (Cited on page 44.)

[103] Juan Juliá Merelo, Antonio M. Mora, Carlos M. Fernandes, and Anna I. Esparcia-Alcázar: *Designing and testing a pool-based evolutionary algorithm*. Natural Computing, 12(2):149–162, 2013, ISSN 1567-7818. http://dx.doi.org/10.1007/s11047-012-9338-5. (Cited on pages 5, 13, and 26.)

[104] Juan Merelo Guervós, Pedro Castillo, and Enrique Alba: *Algorithm::evolutionary, a flexible Perl module for evolutionary computation*. Soft Computing - A Fusion of Foundations, Methodologies and Applications, 14:1091–1109, 2010, ISSN 1432-7643. (Cited on pages 32 and 83.)

[105] Juan Juliá Merelo-Guervos, Pedro A. Castillo, Juan Luis J. Laredo, Antonio Mora Garcia, and Alberto Prieto: *Asynchronous distributed genetic algorithms with javascript and JSON*. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1372–1379, 2008. (Cited on page 93.)

[106] Zbigniew Michalewicz: *Genetic algorithms+ data structures= evolution programs*. Springer, 1996. (Cited on page 21.)

[107] Antonio Miguel Mora, Antonio Fernández-Ares, Juan J. Merelo Guervós, Pablo García-Sánchez, and Carlos M. Fernandes: *Effect of noisy fitness in real-time strategy games player behaviour optimisation using evolutionary algorithms*. J. Comput. Sci. Technol., 27(5):1007–1023, 2012. (Cited on pages 7, 15, 112, 113, 116, 117, 118, 129, and 135.)

[108] Antonio Miguel Mora, Pablo García-Sánchez, Juan J. Merelo Guervós, and Pedro A. Castillo: *Pareto-based multi-colony multi-objective ant colony optimization algorithms: an*

island model proposal. Soft Comput., 17(7):1175–1207, 2013. (Cited on pages 127 and 133.)

[109] Hachem Moussa, Tong Gao, I Ling Yen, Farokh B. Bastani, and Jun Jang Jeng: *Toward effective service composition for real-time SOA-based systems*. Service Oriented Computing and Applications, 4(1):17–31, 2010. (Cited on pages 37 and 139.)

[110] Asim Munawar, Mohamed Wahib, Masaharu Munetomo, and Kiyoshi Akama: *The design, usage, and performance of gridufo: A grid based unified framework for optimization*. Future Generation Computer Systems, 26(4):633 – 644, 2010. (Cited on page 32.)

[111] Hee Khiang Ng, Yew Soon Ong, Terence Hung, and Bu Sung Lee: *Grid enabled optimization*. In *Advances in Grid Computing - EGC 2005*, volume 3470 of *Lecture Notes in Computer Science*, pages 296–304. Springer Berlin Heidelberg, 2005. (Cited on pages 27 and 47.)

[112] OASIS: *Web Services Reliable Messaging (WS-ReliableMessaging)*, 2009. http://docs.oasis-open.org/ws-rx/wsrm/200702. (Cited on page 43.)

[113] OASIS BCM TC: *Business-Centric Methodology for Enterprise Agility and Interoperability. Executive White Paper*, 2003. http://businesscentricmethodology.com/. (Cited on page 46.)

[114] Oasis Open: *UDDI Specification*, 2006. http://www.uddi.org/specification.html. (Cited on page 43.)

[115] Thomas M. Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, R. Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, and Peter Li: *Taverna: a tool for the composition and enactment of bioinformatics workflows*. Bioinformatics, 20(17):3045–3054, OV 22NOV 22 2004. (Cited on page 43.)

[116] Yew Soon Ong, Meng Hiot Lim, Ning Zhu, and Kok Wai Wong: *Classification of adaptive memetic algorithms: a comparative study*. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 36(1):141–152, 2006. (Cited on page 22.)

[117] OSGi Alliance: *OSGi service platform release 4.2*, 2010. Available at: http://www.osgi.org/Release4/Download. (Cited on page 47.)

[118] Mike P. Papazoglou and Willem Jan van den Heuvel: *Service oriented architectures: approaches, technologies and research issues*. The VLDB Journal, 16:389–415, 2007, ISSN 1066-8888. 10.1007/s00778-007-0044-3. (Cited on pages 4, 6, 12, 14, 37, 42, and 43.)

[119] José Parejo, Antonio Ruiz-Cortés, Sebastián Lozano, and Pablo Fernandez: *Metaheuristic optimization frameworks: a survey and benchmarking*. Soft Computing - A Fusion of Foundations, Methodologies and Applications, 16:527–561, 2012, ISSN 1432-7643. http://dx.doi.org/10.1007/s00500-011-0754-8, 10.1007/s00500-011-0754-8. (Cited on pages 4, 5, 6, 12, 13, 14, 32, and 34.)

[120] Sanjay Patil and Eric Newcomer: *ebXML and Web Services*. IEEE Internet Computing, 7(3):74–82, 2003. (Cited on page 44.)

[121] Martin Pelikan: *Bayesian optimization algorithm*. In *Hierarchical Bayesian Optimization Algorithm*, pages 31–48. Springer, 2005. (Cited on page 22.)

[122] Srinath Perera and Dennis Gannon: *Enabling web service extensions for scientific workflows*. In *Workflows in Support of Large-Scale Science, 2006. WORKS'06. Workshop on*, pages 1–10. IEEE, 2006. (Cited on page 43.)

[123] Martin Petzold, Oliver Ullrich, and Ewald Speckenmeyer: *Dynamic distributed simulation of DEVS models on the OSGi service platform*. Proceedings of ASIM 2011, 2011. (Cited on page 71.)

[124] Francisco J. Rodríguez, Carlos García-Martínez, and Manuel Lozano: *Hybrid metaheuristics based on evolutionary algorithms and simulated annealing: Taxonomy, comparison, and synergy test*. IEEE Trans. Evolutionary Computation, 16(6):787–800, 2012. (Cited on pages 22 and 69.)

[125] G.A. Rollings and D.W. Corne: *Intelligent operators for localisation of dynamic smart dust networks*. In *Hybrid Intelligent Systems, 2008. HIS '08. Eighth International Conference on*, pages 477–482, 2008. (Cited on pages 6, 14, and 25.)

[126] Arnon Rotem-Gal-Oz: *Fallacies of distributed computing explained*. URL http://www.rgoarchitects.com/Files/fallacies.pdf, pages=20, year=2006,. (Cited on page 24.)

[127] Arnon Rotem-Gal-Oz, E Bruno, and U Dahan: *SOA patterns*. Manning, 2012. (Cited on page 38.)

[128] Carolina Salto and Enrique Alba: *Designing heterogeneous distributed gas by efficiently self-adapting the migration period*. Applied Intelligence, 36:800–808, 2012. (Cited on page 30.)

[129] J. David Schaffer and Larry J. Eshelman: *On Crossover as an Evolutionary Viable Strategy*. In R.K. Belew and L.B. Booker (editors): *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 61–68. Morgan Kaufmann, 1991. (Cited on page 78.)

[130] Martin Serpell and James E. Smith: *Self-Adaptation of Mutation Operator and Probability for Permutation Representations in Genetic Algorithms*. Evolutionary Computation, 18(3, Sp. Iss. SI):491–514, ALFAL 2010, ISSN 1063-6560. (Cited on page 69.)

[131] Moshe Sipper, Yaniv Azaria, Ami Hauptman, and Yehonatan Shichel: *Designing an evolutionary strategizing machine for game playing and beyond*. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 37(4):583–593, 2007. (Cited on page 113.)

[132] Jim Smith: *Co-evolving memetic algorithms: Initial investigations*. In *Parallel Problem Solving From Nature-PPSN VII*, pages 537–546. Springer, 2002. (Cited on page 22.)

[133] Wenbin Song, Andy Keane, and Simon Cox: *Cfd-based shape optimisation with grid-enabled design search toolkits*. In *UK e-Science All Hands Meeting 2003*, pages 619–627. EPSRC, 2003. (Cited on page 47.)

[134] Wenbin Song, Yew Soon Ong, Hee Khiang Ng, A. Keane, S. Cox, and Bu Sung Lee: *A service-oriented approach for aerodynamic shape optimisation across institutional boundaries*. In *Control, Automation, Robotics and Vision Conference, 2004. ICARCV 2004 8th*, volume 3, pages 2274 – 2279, dec. 2004. (Cited on page 47.)

[135] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen: *Real-time neuroevolution in the nero video game*. IEEE Transactions on Evolutionary Computation, pages 653–668, 2005. http://nn.cs.utexas.edu/?stanley:ieeetec05. (Cited on page 113.)

[136] Ralf Steinmetz and Klaus Wehrle: 2. *what is this peer-to-peer about?* In Ralf Steinmetz and Klaus Wehrle (editors): *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*, pages 9–16. Springer Berlin

Heidelberg, 2005, ISBN 978-3-540-29192-3. http://dx.doi.org/10.1007/11530657_2. (Cited on page 25.)

[137] Rainer Storn and Kenneth Price: *Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces*. Journal of global optimization, 11(4):341–359, 1997. (Cited on page 22.)

[138] James Styles, HolgerH. Hoos, and Martin Muler: *Automatically configuring algorithms for scaling performance*. In Youssef Hamadi and Marc Schoenauer (editors): *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 205–219. Springer Berlin Heidelberg, 2012, ISBN 978-3-642-34412-1. http://dx.doi.org/10.1007/978-3-642-34413-8_15. (Cited on pages 5, 14, and 28.)

[139] Mohammad Hadi Valipour, Bavar Amirzafari, Khashayar Niki Maleki, and Negin Daneshpour: *A brief survey of software architecture concepts and service oriented architecture*. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, pages 34–38, 2009. (Cited on page 39.)

[140] Francisco Fernández de Vega, Gustavo Olague, Leonardo Trujillo, and Daniel Lombraña Gonzalez: *Customizable execution environments for evolutionary computation using boinc + virtualization*. Natural Computing, 12(2):163–177, 2013. (Cited on pages 27 and 33.)

[141] Sebastian Ventura, Cristobal Romero, Amelia Zafra, Jose A. Delgado, and Cesar Hervas: *JCLEC: a Java framework for evolutionary computation*. Soft Computing, 12(4):381–392, EBFEB 2008. (Cited on page 32.)

[142] Stefan Wagner and Michael Affenzeller: *HeuristicLab: A generic and extensible optimization environment*. In Ribeiro, B and Albrecht, RF and Dobnikar, A and Pearson, DW and Steele, NC (editor): *Adaptive and Natural Computing Algorithms*, Springer Computer Science, pages 538–541, 2005. 7th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA), Coimbra, Portugal, MAR 21-23, 2005. (Cited on page 32.)

[143] Stefan Wagner, Stephan Winkler, Erik Pitzer, Gabriel Kronberger, Andreas Beham, Roland Braune, and Michael Affenzeller: *Benefits of plugin-based heuristic optimization software systems*. In Roberto Moreno Díaz, Franz Pichler, and

Alexis Quesada Arencibia (editors): *Computer Aided Systems Theory - EUROCAST 2007*, volume 4739 of *Lecture Notes in Computer Science*, pages 747–754. Springer Berlin / Heidelberg, 2007. (Cited on pages 6, 14, 45, and 71.)

[144] Mark Wittkamp, Luigi Barone, and R. Lyndon While: *A comparison of genetic programming and look-up table learning for the game of spoof*. pages 63–71, 2007. (Cited on page 113.)

[145] World Wide Web Consortium: *Web Services Description Language (WSDL) 1.1*, 2001. http://www.w3.org/TR/wsdl. (Cited on pages 42 and 47.)

[146] World Wide Web Consortium: *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, 2006. http://www.w3.org/TR/REC-xml/. (Cited on page 42.)

[147] Gang Xue, Wenbin Song, Simon J. Cox, and Andy J. Keane: *Numerical optimisation as grid services for engineering design*. J. Grid Comput., 2(3):223–238, 2004. (Cited on page 47.)

[148] Eckart Zitzler and Lothar Thiele: *Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach*. Evolutionary Computation, IEEE Transactions on, 3(4):257–271, 1999. (Cited on page 66.)