Developing Services in a Service Oriented Architecture for Evolutionary Algorithms

Pablo García-Sánchez, María Isabel García Arenas, Antonio Miguel Mora, Pedro Ángel Castillo, Carlos Fernandes, Paloma de las Cuevas, Jesús González and Juan Julián Merelo
University of Granada
Department of Computer Architecture and Computer Technology, ETSIIT and CITIC-UGR
18071 - Granada, Spain
pgarcia@atc.ugr.es

ABSTRACT

This paper shows the design and implementation of services for Evolutionary Computation in the Service Oriented Architecture paradigm. This paradigm allows independence in language and distribution, but the development requires to manage some technological and design issues, such as abstract design or unordered execution. To solve them, OSGiLiath, an implementation of an abstract Service Oriented Architecture for Evolutionary Algorithms, is used to develop new interoperable services taking into account these restrictions.

Categories and Subject Descriptors

D.2.11 [**Software**]: Software Engineering—*Software Architectures*; D.2.12 [**Software**]: Software Engineering—*Interoperability*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms

Keywords

Service oriented architecture, evolutionary algorithms, genetic algorithms, distributed algorithms, OSGi

1. INTRODUCTION

Service Oriented Architecture (SOA) [19] is becoming an important trend in software development. This paradigm allows the organization and distribution using the *service* concept. A service is an interaction depicted in Figure 1. The service provider publishes *service descriptions* (or interfaces) in the *service registry*, so the *service requesters* can discover services and bind to the *service providers* to use it.

GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.

SOA allows independence in language and distribution mechanisms, aiming to easy extension and integration, but it has the following restrictions:

- Services must be input/output functions.
- The services must not have state (i.e. not global variables).
- The order of execution of the services is not defined.
- Services must be designed as abstract as possible.

Distributed computing offers the possibility of taking advantage of parallel processing, in order to obtain a higher computing power [3]. SOA is also applied in this area, using platforms based on Web Services [19], and new standards for this paradigm have emerged, like OSGi (Open Services Gateway Initiative) [18].

OSGi allows build quality software systems considering a high level of modularity. Besides the benefits that classic modularization paradigms can offer (like object-oriented modelling), and the improvements in test, reusability, availability and maintainability, it is necessary to explore other modelling techniques, such as the plug-in based development and the SOA design. This kind of development simplifies aspects such as the complexity, personalization, configuration, development and cost of the software development. In the optimization heuristics software area, the benefits that using this kind of development can offer are carried out in the development of algorithms, experimental evaluation, and combination of different optimization paradigms [22].

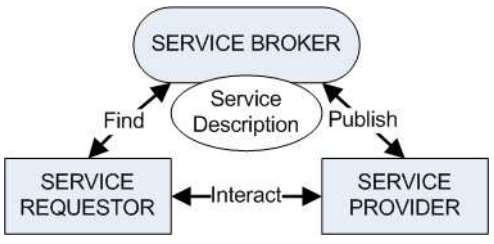

Figure 1: Service interaction schema. The service provider publish a service description that is used by the requestor to find and use services.

In our previous work [8] we presented an abstract Service Oriented Architecture for Evolutionary Algorithms (SOA-EA), with guidelines and steps to migrate from traditional development in Evolutionary Algorithms (EAs) to SOA. It also presented a specific implementation, called OSGiLiath (*OSGi Laboratory for Implementation and Testing of Heuristics*): an environment for the development of distributed algorithms extensible via plug-ins architecture and based in a wide-accepted software specification (OSGi). In this work, a full service development is presented, taking into account the specific technology used, instead an abstract design, as in our previous work.

The rest of the work is structured as follows: after the state of the art, we present design principles to create services for Evolutionary Computation (Section 3). Then, the OSGi implementation technology is explained in Section 4, used to build our framework (described in Section 5). Then, the steps to create services with this framework are explained (Section 6). Finally, conclusions and suggestions for future work are presented (Section 7).

2. STATE OF THE ART

Even as SOA is used extensively in software development, it is not widely accepted in the EA community. Most of the frameworks have the lack of low generality, because they are focused in a specific field, like EasyLocal++ [10] (focused in Local Search) or SIGMA [11] (in the field of optimization-based decision support systems). Another common issue is that they are just libraries or Perl modules [16], they have no GUIs, or they are complicated to install and require many programming skills. Another problem could be the lack of comfort, for example, C++ has a more complicate sintaxis than other languages. There also exist frameworks that use metaheuristics to apply in specific fields, like the KEEL framework [2], that let the creation of heuristics to apply in data-mining problems.

Among this great number of software tools we want to focus in the most widely accepted distributed algorithms frameworks. ECJ [15], Evolutionary Computation in Java, is a set of Java classes that can be extended and includes several communication modules. MALLBA [1] is based in software skelletons with a common and public interface. Every skeleton implements a resolution technique for optimization in the fields of exact, heuristic or hybrid optimization. It provides LAN and WAN capacity distribution with MPI . However, this both frameworks are not based in the plug-in development, so they can not take advantage of features like the life-cycle management, versioning, or dynamic service binding, as OSGi proposes.

Another important platform is DREAM [4], which is an open source framework for Evolutionary Algorithms based on Java that defines an island model and uses the Gossip protocol and TCP/IP sockets for communication. It can be deployed in P2P platforms and it is divided in five layers. Every layer provides an user interface and different interaction and abstraction level, but adding new functionalities is not so easy, due to the the fact that system must be stopped before adding new modules and the implementation of interfaces must be defined in the source code, so a new compilation is needed (as in ECJ). OSGi lets the addition of new functionalities only compiling the new features, not the existing ones.

The MALLBA authors are now working in the jMetal Framework [5], that is a newer Java-based framework, but it has not yet distributed capabilities and it is focused in multi-objective optimization.

ParadiseEO [14] allows the design of Evolutionary Algorithms and Local Search with hybridization, providing a variety of operators and evaluation functions. It also implement the most common parallel and distributed models, and it is based in standard libraries like MPI, PVM and Pthreads. However, it has the same problems that the previous frameworks, not lifecycle management or service oriented programming. GAlib [23] is very similar and share the same characteristics and problems.

In the field of the plug-in based frameworks, HeuristicLab [21] is the most outstanding example. It also allows the distributed programming using Web Services and a centralized database, instead using their own plug-in design for this distributed communication.

METCO framework [13] also have the same problems, it does not use a standard plug-in system or SOA, but let the implementation of existing interfaces, and lets the user configure its existing functionalities.

Finally, the only service oriented optimization framework is GridUFO [17], but it only allows the modification of the objective function and the addition of whole algorithms, without combining existing services.

Previous frameworks are designed to be extensible and reusable, but without taking into account the restrictions of SOA to achieve even more independence and development improvements.

3. DESIGNING SERVICES FOR A SERVICE ORIENTED ARCHITECTURE FOR EVOLUTIONARY ALGORITHMS

In [8] we demonstrated that it is possible to create a service oriented architecture for EAs using a specific SOA technology. This architecture used the features that SOA offers. To do this, loose coupling services for EAs were designed (SOA-EA), and they were implemented using a SOA technology and compared with other frameworks. These services can be combined in several ways to obtain different algorithms (for example, from a canonical GA, a NSGA-II can been created just adding new services). Also several techniques were presented to combine existing services in a flexible way.

3.1 Design principles

One of the main restrictions in SOA, appart from focusing in develop abstract services, is the stateless nature of services. Therefore, in SOA the services design must follow several guidelines.

First, as services are unaware of others, there must 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 are working in parallel, and calling this function at the same time the counter would not distinguish between algorithms, giving erroneous results. However, a service that maintains some kind of state is allowed, for example, a statistic service that read events from all the algorithms being executed at the same time, but this should be managed to avoid errors.

Also, a service must not be distinguishable from local or remote running in other node in the network. Every stage in the algorithm should be treated as 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 it should be correctly managed and used.

Moreover, a service is always a request-response function. For example, the fitness calculation must 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 things such as 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 separate concepts such 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 we should sepparate the concept *recombine* from *crossover*.

Finally, we must not make assumptions about services previously executed or being executed next. For example, services such *Recombinator* or *Mutator* should return the individuals with their fitness already calculated. 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.

3.2 Other technological restrictions

In [8] we also presented the advantages of using SOA in Evolutionary Algorithms area: firstly, SOA fits with the genericity advantages in the development of software for EAs [6] and adds new features, like language independence and distribution mechanisms. It also allows the addition and removal of services in execution time without altering the general execution of the algorithm (that is, it is not mandatory to stop it or to add extra code to support new operators). This issue increases the interoperability between different software elements. Moreover, this allows easy code distribution: SOA does not require the use of a concrete implementation or library.

In this work, a new process development, explaining the specific technology used is presented. The services developed must match with the next technological restrictions:

- These services can dynamically bound to change the needed EA aspects.
- The source code of the basic EA services must not been re-written or re-compiled to achieve this task.
- 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 exisitng services).

4. IMPLEMENTATION TECHNOLOGY

This section dives into some technical features of the OSGi platform, to guide the reader to understand the OSGiLiath framework in a deeper way, and to evaluate the advantages of using this features in the development of distributed algorithms to match with the previous restrictions.

The used technology, OSGi, was proposed by a consortium of more than eighty companies in order to develop an infrastructure for the deployment of service in heterogeneous network of devices, mainly oriented to domotics [9]. Nowadays it defines a specification for a Service Oriented Architecture for virtual machines (VMs). It provides very desiderable features, like packet abstraction, life-cycle management, packaging or versioning, allowing significant reduction of the building, support and deployment complexity of the applications.

OSGi technology allows dynamic discovery of new components, 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. Nowadays there are some challenges in the OSGi development [12], but they only affect the creation of very complex applications.

This advantages are not so costly, as can be thought: the OSGi framework can be implemented in a *jar* file[1] of 300KB. Also, and different of the normal usage of Java, each class pre-charges only the other classes to use, not all. Also 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 (Integrated Development Environment) is built over OSGi, and also big application servers (Glassfish or IBM Websphere) or residential gateways [9], among other examples.

4.1 OSGi Architecture

To understand how OSGi [18] works and which capabilities could offer to the OSGiLiath users it is necessary to understand how OSGi is built. OSGi has a layered model that is depicted in Figure 2. The terms present in this Figure are:

- Bundles: Bundles are the OSGi components made by developers. Is a normal *jar* file including Java classes and interfaces with different MANIFEST.MF and extra files (such as the Service Descriptions).
- 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.

[1]A jar file is a file that groups the compiled Java files.

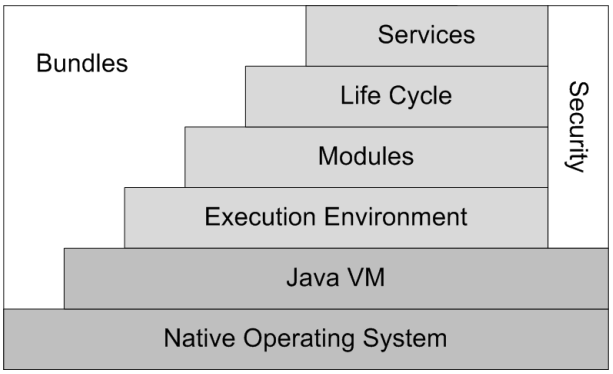

Figure 2: OSGi layered architecture. Every layer is built from the one just below.

4.2 OSGi configuration files

Regarding to explained OSGi layers how to use all OSGi capabilities is shown next.

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.

Java programmers are familiar with the *jar* concept. The first difference among a *bundle* and a *jar* is that the second has a MANIFEST.MF file adapted to be used in OSGi. This file indicates which clases imports or exports the *bundle*. An example can be seen in Figure 3. This file shows the name of the bundle and its version (this is useful to select specific services), and 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, to create a specific implementation of an interface (i.e. *FitnessCalculator*) is as follows:

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

With Declarative Services, the *new ExampleFunction()* part is not used, so if a new implementation is desired no code recompilation is necessary. Figure 4 shows a declarative service description file, which establish 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 (***). We need to create XML files for the rest of services to expose (i.e. *TransportData*). The file where these capabilities are defined is declared in section

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

Service-Component of MANIFEST.MF file, as can be seen in Figure 3.

Next code shows the code for this implementation:

```
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){
        ...
 }
}
```

4.3 Event Administration

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

To send events to other bundles:

- 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

```
<?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 4: 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.

Code to send an event to other bundles is shown below. 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).

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

For 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.

This code shows how to handle events. In this case we have published the *ExampleService* with the implementation *ExampleImpl*, that is listening under the topic *"es/ugr/osgiliath/algorithms/*"*.

```
class ExmplImpl implements ExmplService,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");
  }
 }
}
```

4.4 Distribution

In a good service-oriented framework for EAs all services must be capable to be indistinguishable of being a local or a remote service. 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.

OSGi allows several implementations for the service distribution. ECF (Eclipse Communication Framework)[2] has been chosen because it is the most mature and accepted implementation [20], and it also supports the largest number of transmission protocols, including both synchronous and asynchronous communication. It provides a modular implementation of the OSGi 4.2 Remote Services standard[3]. 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.

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 [4].
- Remote Service API: Includes protocols to establish the communication (data streams, formats and others): R-OSGi, ActiveMQ/JMS, REST, SOAP, XMPP, ECF Generic [5]. This allow to communicate to systems that do not use OSGi or Java.

5. OSGILIATH

All previous elements can be combined to create a service oriented environment. This section explains the function-

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

ality and design of the proposed environment, called OSGiLiath, presented in [7]. This environment is a framework for the development of heuristic optimization applications, not centered on a concrete paradigm, and whose main objective is to promote the OSGi and SOA usage and offer to programmers the next features:

- Easy interfaces. After a study of the previous frameworks a complete interface hierachy has been developed.
- Asynchronous data sending/receiving. Thanks to ECF distributed capabilities, the framework has easy distribution of services, without implementing specific source functions, like MPI or other distribution frameworks. Programmers do not need to write communication code.
- Component Oriented Programming. The framework is plug-in oriented, so new improvements can be added in easy way without modification of existent modules. Adding o modifying implementations of services can be performed without re-compilation of the source code.
- Client/Server or Distributed Model. All components of the framework can communicate in a bi-directional way, so a central broker is not necessary if it is not required.
- Paradigm independent. The framework is not focused in a type of metaheuristic.
- Declarative Services. Bind interfaces to specific implementations can be done without modifying existent source code. Programmers do not need to instantiate implementations of the services.
- Remote event handling: Using the OSGi advantages, users can use a powerful tool to synchronize or share data among services.

A comparison of this environment with other frameworks was also presented in [8]. OSGiLiath attained lower number of lines of code than other frameworks for combining existent services, attaining similar times than other frameworks in Java, such as ECJ. However, no required code for distribution was added.

The source code is available at `http://www.osgiliath.org`, under a LGPL license.

5.1 OSGiLiath organization

By now, OSGiLiath counts with the next bundles:

- 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. By now, in each step the *IntelligentRandomManager* selects randomly from the available Crossovers, Mutators and Replacers implementations.

6. DEVELOPMENT OF SERVICES IN OSGILIATH

This section presents the steps to add services to the existent OSGiLiath core. In this section the implementation to add the Vehicle Routing Problem (VRP) are explained.

6.1 Bundle creation

In OSGiLiath Services can be added to existent bundles or new bundles can be created. Each bundle includes a MANIFEST.MF file (as depicted in Figure 3). In this case, we have selected the packages to import (including interfaces and classed from the Osgiliath core) and to export. The section *Service-Description* shows the location of the component definitions that describe the services. In this case, two interfaces will be implemented: *TransportData* and *FitnessCalculator*. Other classes related with the VRP are added, such as *Route* or *Shop*.

6.2 Implementing services

To implement a service, a class must be created implementing an interface. For example, *VRPFitnessCalculator* implements the interface *FitnessCalculator*. The relation between these two elements is made in the Component Definition of the Figure 4. This way, the implementation is announced to the other services in the environment, that can bind or unbind. For example, the implementation *VRPInitializer* (implementing *Initializer*) requires this implementation to create the individuals. Services can automatically bind other services with the set/unset methods in the component definition. Also, other services appart from the EA can be added (for example, in this bundle the service

TransportData, who includes information about distances and time of the nodes has been included). Finally, *VRPMutation* and *VRPCrossover* are added, following the suggestions of Section 3.

6.3 Adding communication

Thanks to the OSGi 4.2 specification, services must be indistinguishable from the ones in the local OSGi environment or in other OSGi environment (in the same machine or even in the same network). To achieve this, the ECF is used to export services. In this case, the *Migration* service is used. This service 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. One example of 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) thanks to the bind/unbind methods of declarative services and ECF. 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 5 shows this configuration. The *Replacer* implementation, a reference to the local *Migrator* interface just send and read the individuasl. The *MigratorRingBuffer* implementation binds an unbinds other migrators in other nodes, keeping a reference to these remote service interfaces. 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.

7. CONCLUSIONS

This work shows the requirements to create a service oriented evolutionary framework and the technology used to accomplish these requirements. Service Oriented Architecture (SOA) offers independence of language, distribution or even operating systems, allowing the integration of different elements. However, some issues have to be considered in the development: the services are stateless input/output functions, services can appear or disappear in real time and the order of execution could not be fixed. In the Evolutionary Algorithms (EAs) area, services must be developed taking into account these issues, so the abstract design of elements for EAs has been explained. Technological requirements are also solved using an existent service oriented technology: OSGi. The elements to create a service oriented architecture for EAs using this technology have been described, and an example of development has been shown.

As future work a study about scalability using other algorithms (such as GRASP, Scatter Search, Ant Colony Optimization and others) will be performed. In addition, we are going to increase the usage of the OSGi capabilities, like the Event Administration or automatic service management in a deeper way. Additionally we intend to create a web portal or a Maven[6] repository to centralize all new implementations of problems and algorithms to let the distribution within the base platform. A study of porting existing software to our framework (especially those works that are written in Java, like DREAM or ECJ) will be performed. Moreover, due to the ease of implementations binding with their interfaces, it is planned to develop the functionality of choosing one implementation or another depending on several parameters or, for example, using Genetic Programming to evolve and hybridize algorithms.

[6] `http://maven.apache.org`

8. ACKNOWLEDGMENTS

This work has been supported in part by FPU research grant AP2009-2942 and projects EvOrq (P08-TIC-03903), Project 83 (CANUBE) awarded by the CEI-BioTIC UGR, and TIN2011-28627-C04-02 (ANYSELF).

9. REFERENCES

[1] E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, C. León, G. Luque, J. Petit, C. Rodríguez, A. Rojas, and F. Xhafa. Efficient parallel LAN/WAN algorithms for optimization. the MALLBA project. *Parallel Computing*, 32(5-6):415–440, 2006.

[2] J. Alcalá-Fdez, L. Sánchez, S. García, M. J. del Jesús, S. Ventura, J. M. Garrell i Guiu, J. Otero, C. Romero, J. Bacardit, V. M. Rivas, J. C. Fernández, and F. Herrera. KEEL: a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3):307–318, 2009.

[3] M. Altunay, P. Avery, K. Blackburn, B. Bockelman, M. Ernst, D. Fraser, R. Quick, R. Gardner, S. Goasguen, T. Levshina, M. Livny, J. McGee, D. Olson, R. Pordes, M. Potekhin, A. Rana, A. Roy, C. Sehgal, I. Sfiligoi, F. Wuerthwein, 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, JUN 2011.

[4] 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.

[5] J. J. Durillo, A. J. Nebro, and E. Alba. The jmetal framework for multi-objective optimization: Design and architecture. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[6] C. Gagné and M. Parizeau. Genericity in evolutionary computation software tools: Principles and case-study. *International Journal on Artificial Intelligence Tools*, 15(2):173, 2006.

[7] P. García-Sánchez, J. González, P. Castillo, J. Merelo, A. Mora, J. Laredo, and M. Arenas. A Distributed Service Oriented Framework for Metaheuristics Using a Public Standard. *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, pages 211–222, 2010.

[8] P. García-Sánchez, J. González, P.A. Castillo, M.G. Arenas, and J.J. Merelo-Guervós. Service oriented evolutionary algorithms. *Soft Computing*, pages 1–17, 2013. In press.

[9] P. García-Sánchez, J. González, A. Miguel Mora, and A. Prieto. Deploying intelligent e-health services in a

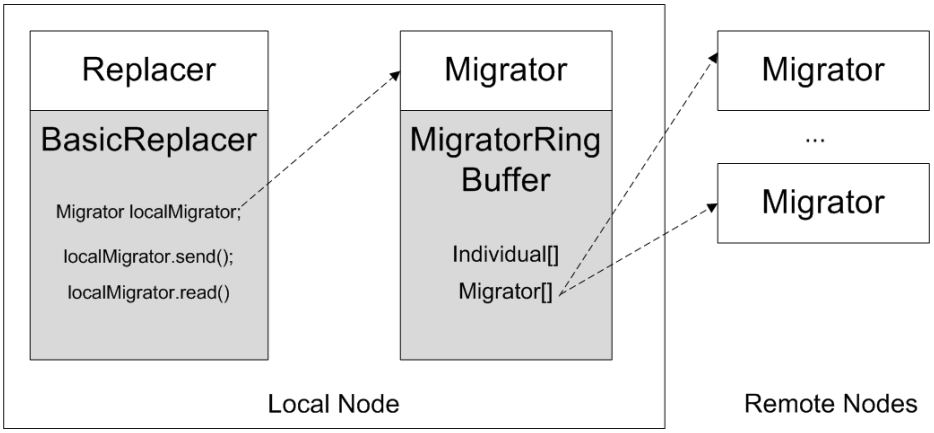

Figure 5: Using the Migrator service to create a distributed island EA with a ring topology (white boxes are service interfaces and grey boxes are implementations).

mobile gateway. *Expert Syst. Appl.*, 40(4):1231–1239, 2013.

[10] L.D. Gaspero and A. Schaerf. Easylocal++: an object-oriented framework for the flexible desgin of local search algorithms and metaheuristics. In *Proceedings of 4th Metaheuristics International Conference (MIC'2001)*, pages 287–292, 2001.

[11] J. R. González, D. A. Pelta, and A. D. Masegosa. A framework for developing optimization-based decision support systems. *Expert Systems with Applications*, 36(3, Part 1):4581 – 4588, 2009.

[12] P. Kriens. Research challenges for OSGi. 2008. Available at: `http://www.osgi.org/blog/2008/02/research-challenges-for-osgi.html`.

[13] C. León, G. Miranda, and C. Segura. Metco: A parallel plugin-based framework for multi-objective optimization. *International Journal on Artificial Intelligence Tools*, 18(4):569–588, 2009.

[14] A. Liefooghe, L. Jourdan, and E.G. Talbi. A software framework based on a conceptual unified model for evolutionary multiobjective optimization: ParadisEO-MOEO. *European Journal of Operational Research*, 2010.

[15] S. 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`.

[16] J.J. Merelo Guervós, P. Castillo, and E. Alba. Algorithm::evolutionary, a flexible Perl module for evolutionary computation. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 14:1091–1109, 2010.

[17] A. Munawar, M. Wahib, M. Munetomo, and K. Akama. The design, usage, and performance of gridufo: A grid based unified framework for optimization. *Future Generation Computer Systems*, 26(4):633 – 644, 2010.

[18] OSGi Alliance. OSGi service platform release 4.2, 2010. Available at: `http://www.osgi.org/Release4/Download`.

[19] M. Papazoglou and W.-J. van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16:389–415, 2007. 10.1007/s00778-007-0044-3.

[20] M. Petzold, O. Ullrich, and E. Speckenmeyer. Dynamic distributed simulation of DEVS models on the OSGi service platform. *Proceedings of ASIM 2011*, 2011.

[21] S. Wagner and M. Affenzeller. HeuristicLab: A generic and extensible optimization environment. In Ribeiro, B. and Albrecht, R.F. and Dobnikar, A. and Pearson, D.W. and Steele, N.C., 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.

[22] S. Wagner, S. Winkler, E. Pitzer, G. Kronberger, A. Beham, R. Braune, and M. 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.

[23] B. M. Wall. A genetic algorithm for resource-constrained scheduling, Ph.D. thesis, MIT. 1996. Available at: `http://lancet.mit.edu/ga`.