Developing services in a service oriented architecture for evolutionary algorithms

Anonymous
Lost island
Unknown
Pacific Ocean
jack,sawyer,hurley@lost.com

Anonymous
Lost island
Unknown
Pacific Ocean
lock@lost.com

ABSTRACT

This paper shows a

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; G.1.6 [**Mathematics of Computing**]: NUMERICAL ANALYSIS—*Optimization*

General Terms

Algorithms

Keywords

service oriented architecture, distributed algorithms, osgi

1. INTRODUCTION

When building quality software systems it is necessary to design them with 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 another modelling techniques, like the plug-in based development [**?**]. This kind of development simplifies aspects such as the complexity, personalization, configuration, development and cost of the software systems. In the optimization heuristics software area, the benefits the usage of this kind of development can offer are carried out in the development of algorithms, experimental evaluation, and combination of different optimization paradigms [**?**].

On the other hand, other patterns for integration, like SOA, have emerged. SOA (Service Oriented Architecture) [**?**] is a paradigm for organizing and utilizing distributed capabilities, called *services*. A service is an interaction depicted in Figure **??**.

In our previous work [1] we presente an abstract Service Oriented Architecture for Evolutionary Algorithms (SOA-EA), with guidelines and steps to migrate from traditional development to SOA. It also presents an specific implementation, called OSGiLiath: an environment for the development of distributed algorithms extensible via plug-ins architecture and based in a wide-accepted software specification (OSGi: Open Services Gateway Initiative [**?**]).

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.

Distributed computing offers the possibility of taking advantage of parallel processing in order to obtain a higher computing power than other multiprocessor architectures. Two clear examples are the research lines centred in clusters [**?**] and GRID [**?**] for parallel processing. SOA it is also used in this area, using platforms based on Web Services [**?**], and new standards for this paradigm have emerged, like OSGi.

Therefore, the objective of the proposed environment in this paper is to facilitate the development of distributed computing applications by using the OSGi standard, taking advantage of the plug-in software development and SOA that can compete with existing distributed applications in easy of use, compatibility and development. The rest of the work is structured as follows: after the state of the art, we present the developed algorithms and experimental setting. Then, the results of the experiments are shown (Section **??**), followed by conclusions and suggestions for future work.

2. STATE OF THE ART

Even as SOA is used extensively in software development, it is not widely accepted in the main EA software.

Nowadays there are many works about heuristic frameworks. Most of them have the lack of low generality, because they are focused in an specific field, like EasyLocal++ [**?**] (focused in Local Search) or SIGMA [**?**] (in the field of optimization-based decision support systems). Another common issue is that they are just libraries or Perl modules [**?**], 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 [**?**], that let the creation of heuristics to apply in data-mining problems.

Among this great number of frameworks we want to focus in the most widely accepted distributed algorithms frameworks. ECJ [**?**], Evolutionary Computation in Java, is a set of Java classes that can be extended and includes several communication modules. MALLBA [**?**] is based in soft-

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

ware 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 [**?**], 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 modules. Every module 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. OSGi lets the addition of new functionalities only compiling the new features, not the existing ones.

Moreover, MALLBA and DREAM could be considered abandoned frameworks, due to no new publications or versions have been released. The MALLBA authors are now working in the jMetal Framework [**?**], that is a newer Java-based framework, but it has not yet distributed capabilities and it is focused in multi-objective optimization.

ParadiseEO [**?**] 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 [**?**] is very similar and share the same characteristics and problems.

In the field of the plug-in based frameworks, HeuristicLab [**?**] 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.

HeuristicLab [**?**] is one of the few plug-in and service oriented frameworks. It uses web services for communication, but just to distribute the load, after consulting a central database of available jobs.

Finally, METCO framework [**?**] 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 [**?**], but it only allows the modification of the objective function and the addition of whole algorithms, without combining existing services.

3. SERVICE ORIENTED ARCHITECTURE IN EVOLUTIONARY ALGORITHMS

Previous sections have demonstrated that it is possible to create a service oriented architecture for EAs using a specific SOA technology. This architecture uses the features that SOA offers. To do this: In summary, the previous works present a number of shortcomings when designing and adding new features: they need to modify source code or be stopped in order to add new features and they are not based in a public plug-in specification. Also they have not an event administration mechanism and they are not service-oriented, so they not take advantage of this paradigm.

- In Sections **??** and **??** loose coupling services for EAs have been designed (SOA-EA), and they have been implemented in Section **??**.
- 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). These services are 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.
- New services can be added in execution time using our implementation.
- No specific source code for a basic distribution have been added, neither the existing source code has been modified.
- Several techniques have been presented to combine existing services in a flexible way.

However, after the explanation of the most important issues of EAs in SOA (Algorithm representation, dynamism, load distribution and self-adapting), we want to share the benefits of SOA with the rest of EA researchers:

- Firstly, SOA fits with the genericity advantages in the development of software for EAs [**?**] and adds new features, like language independence and distribution mechanisms.
- SOA 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).
- It also increases the interoperability between different software elements (for example, it is possible to add communication libraries without modifying existing code).
- Related to the previous point, the existing EA frameworks could be re-used thanks to SOA, because it provides language independence.
- Easiness for code distribution: SOA does not require the use of a concrete implementation or library.
- Access to already created and operative services.
- Collaboration among geografically distributed work teams.

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.

OSGi (Open Service Gateway Initiative) [**?**] 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 [**?**]. 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 [**?**], 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 [**?**], among other examples.

4.1 OSGi Architecture

To understand how OSGi [**?**] 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 1. The terms present in this Figure are:

- Bundles: Bundles are the OSGi components made by developers (explained in previous sections).
- 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.

4.2 OSGi configuration files

Regarding to explained OSGi layers how to use all OSGi capabilities is shown next. 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 clasess imports or exports the *bundle*. An example can be seen in Figure 4.2. This file shows the name of the bundle and its version (this is useful to select specific services), and the

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

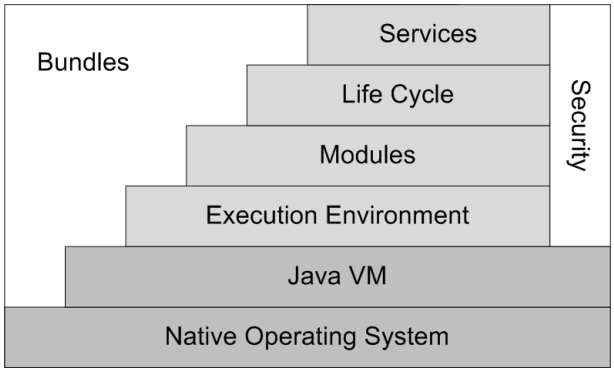

Figure 1: OSGi layered architecture. Every layer is built from the one just below.

Manifest-Version: `1.0`
Bundle-ManifestVersion: `2`
Bundle-Name: `GeneticAlgorithm Plug-in`
Bundle-SymbolicName: `geneticalgorithm`
Bundle-Version: `1.0.0`
Bundle-RequiredExecutionEnvironment: `J2SE-1.5`
Import-Package: `ch.ethz.iks.r-osgi,`
`es.ugr.osgiliath.algorithms,`
`es.ugr.osgiliath.algorithms.types,`
`es.ugr.osgiliath.network,`
`es.ugr.osgiliath.problem,`
`es.ugr.osgiliath.utils,`
`org.osgi.framework`
Export-Package: `es.ugr.osgiliath.geneticalgorithm,`
`es.ugr.osgiliath.geneticalgorithm.distributed`
`es.ugr.osgiliath.geneticalgorithm.individual`
`es.ugr.osgiliath.geneticalgorithm.operators`
Service-Component: `OSGI-INF/distributedGA.xml`

Figure 2: Example of MANIFEST.MF. This example defines which packages are necessary to activate the bundle and which packages are exported.

execution environment (that is, the Java Virtual Machine required). Also, this file specifies the XML files of the declarative services (in section (*Service-Component*).

As can be seen in previous sections, it is in the second and third level of the development in OSGiLiatH where the declarative services specification is used to automatically obtain the implementation of the interfaces. In normal environments, to create an specific implementation of an interface (i.e. *ObjectiveFunction*) is as follows:

```
class GeneticAlgorithm implements Algorithm{
 ObjectiveFunction obj;
 //A new instance is bind to a reference
 ObjectiveFunction f = new ExampleFunction();
}
```

With Declarative Services, the *new Implementation()* part is not necessary. For example, Figure 4.2 shows a service component file, which stablish in execution time which implementation is bound to the interfaces. This example indicates that the implementation of service *Algorithm* is *GeneticAlgorithm*, but this service is not activated until all their references (other services, like *ObjectiveFunction* or

```
<?xml version="1.0"?>
<component name="geneticalgorithm">
 <implementation
  class="es.ugr.osgiliath.geneticalgorithm.GeneticAlgorithm"/>
 <service>
  <provide
   interface="es.ugr.osgiliath.algorithms.Algorithm"/>
 </service>
 <reference name="OBJECTIVEFUNCTION"
  interface="es.ugr.osgiliath.geneticalgorithm.ObjectiveFunction"
  bind="setObjectiveFunction"
  unbind="unsetObjectiveFunction"
  cardinality="1..1"/>
 <reference name="PARAMETERS"
    bind="setParameters"
  unbind="unsetParameters"
  cardinality="1..1"/>
 <reference name="CROSSOVER"
  interface="es.ugr.osgiliath.geneticalgorithm.operators.Crossover"
  bind="setCrossover"
  unbind="unsetCrossover"
  cardinality="1..1"/>
 <!-- Rest of references ...-->
</component>
```

Figure 3: Service Description. This documents indicates that the implementation of the service *Algorithm* is *GeneticAlgorithm*, but it can not activate until their references (other services) are activated.

Crossover) 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 (***). The file where these capabilities are defined is declared in section *Service-Component* of MANIFEST.MF file, as can be seen in Figure 4.2.

This document indicates that the implementation of the service *Algorithm* is the class *GeneticAlgorithm*, but it can not be activated until the rest of the references (another services, like *Crossover*) are activated first. We need to create XML files for the rest of services to use (i.e. ObjectiveFunction)

```
class GeneticAlgorithm implements Algorithm{
 //Other service references, like Crossover, Parameters...
 ObjectiveFunction obj;

 //Methods to bind/unbind each reference
 public ObjectiveFunction setObjectiveFunction(ObjectiveFunction obj){
  this.obj = obj;
 }

 public void unsetObjectiveFunction(ObjectiveFunction obj){
  this.obj = null;
 }
}
```

We have to say that in future work these kind of files will be automatically created, being this task transparent to future users of the OSGiLiatH framework.

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/geneticalgorithm/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 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/geneticalgorithm/en
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/geneticalgorithm/*"* (the * is a wildcard) inside the <property> tag in the Service Component.
- 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/geneticalgorithm/*"*.

```
class ExampleImpl implements ExampleService,EventHandler{

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

4.4 Distribution

In OSGiLiath 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.

OSGi allows several implementations for the service distribution. ECF[2] has been chosen because it is the most

[2] http://www.eclipse.org/ecf/

mature and accepted implementation [?], 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.. 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
- Service Provider API: Includes protocols to establish the communication (data streams, formats and others): R-OSGi, ActiveMQ/JMS, REST, SOAP, XMPP, ECF Generic

5. OSGILIATH

This section explains the functionality and design of the proposed environment, called OSGiLiatH (*OSGi Laboratory for Implementation and Testing of Heuristics*). This environment is a framework for the development of heuristic optimization applications, not centred 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 R-OSGi 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 and R-OSGi advantages, users can use a powerful tool to synchronize or share data among services.

The source code is available at `http://atc.ugr.es/~pgarcia`, under a GPL license.

[3] `http://www.osgi.org/Release4/Download`

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...
- Evolutionary Algorithm: Evolutionary Algorithm implementation and interfaces to create the rest of Recombinator and Crossover, Mutator and Mutation, TERMINA. It also provides interfaces for the creation of individuals: Individual, Fitness, Gene, Genome,
- Basic Evolutionary Components: Includes several implementations of the previous interfaces:
- Binary Problems: Includes implementation of well-known problems, such as OneMax and MMDP, with INDIVIDUALS tal...
- Function Problems:
- NSGA2:
- OSGiLiART: Service implementation for the creation of Evolutionary Art.
- 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 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 hace PATATIN Y PATATAN.

6. DEVELOPMENT OF SERVICE

This section presents the steps to add services to the existent OSGiLiath core.

6.1 Bundle creation

6.2 Implementing services

7. CONCLUSIONS

8. ACKNOWLEDGMENTS

This work has been supported in part

9. REFERENCES

[1] A. Nonymous. Anonymous framework. *Secret Journal*, 1(1):1–1, 1.