A distributed service oriented framework for metaheuristics using a public standard

P. García-Sánchez, J. González, P.A. Castillo, J.J. Merelo, A.M. Mora, J.L.J. Laredo and M.G. Arenas

Abstract This work presents a Java-based environment that facilitates the development of distributed algorithms using the OSGi standard. OSGi is a plug-in oriented development platform that enables the installation, support and deployment of components that expose and use services dynamically. Using OSGi in a large research area, like the Heuristic Algorithms, facilitate the creation or modification of algorithms, operators or problems using its features: event administration, easy service implementation, transparent service distribution and lifecycle management. In this work, a framework based in OSGi is presented, and as an example two heuristics have been developed: a Tabu Search and a Distributed Genetic Algorithm.

1 Introduction

Nowadays the Metaheuristics Research Area has a wide number of algorithms and problems. There are many implementations of them, using several programming languages, frameworks and architectures, but without using a well-defined plug-in specification.

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 [21]. 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 concreted in the development of algorithms, experimental evaluation, and combination of different optimization paradigms [21].

P. García-Sánchez
Dept. of Computer Architecture and Computer Technology, e-mail: pgarcia@atc.ugr.es

On the other hand, other patterns for integration, like SOA, have emerged. SOA (Service Oriented Architecture) [18] is a paradigm for organizing and utilizing distributed capabilities, called *services*. A service is an interaction depicted in Figure 1.

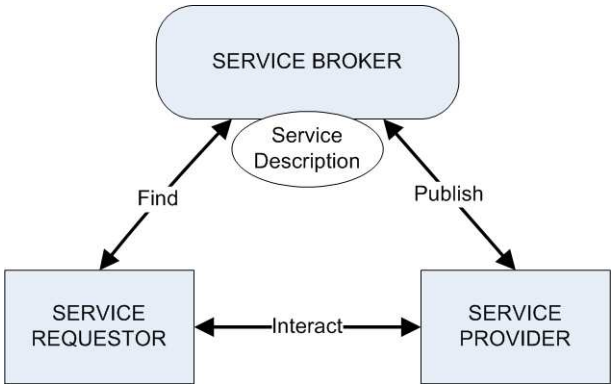

Fig. 1 Service interaction schema. The service provider publish a service description that is used by the requester to find and use services.

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 [5] and GRID [9] for parallel processing. SOA it is also used in this area, using platforms based in Web Services [18], and new standards for this paradigm have emerged, like OSGi.

OSGi (Open Service Gateway Initiative) [2] 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 domotic [15]. Nowadays it defines a specification for a Service Oriented Architecture for virtual machines (VMs). It provides very desiderable features, like packet abstraction, lifecycle management, packaging or versioning, allowing significant reduction of the building, support and deployment complexity of the applications.

OSGi technology allows the components to be dynamically discovered among them to increase the collaboration 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 to the creation of very complex applications.

Therefore, the objective of the proposed environment 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 this work is structured as follows: first the state of the art in similar applications is described (section 2). Section 3 introduces the technologies used in the development of this work. Then, we present (section 4) the design of the proposed architecture (called OSGiLiath) and the development of two computing applications using a Distributed Genetic Algorithm and a Tabu Search. Experiments and yielded results are shown in section 6. Finally the conclusions and future work are presented.

2 State of the art

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++ [10] (focused in Local Search) or SIGMA [11] (in the field of optimization-based decision support systems). Another common problem is that they are just libraries (like ECJ [14], Evolutionary Computation in Java), they have no GUIs, or they are complicated to install and require many programming skills. Another issue could be the lack of comfort, for example, C++ has a more complicate sintaxis than other languages.

Among this great number of frameworks we want to focus in the most widely accepted distributed algorithms frameworks. MALLBA [1] is based in software skeletons 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, it is not based in the plug-in development, so it can not take advantage of features like the life-cycle management, versioning, or dynamic service binding, as OSGi proposes.

Another important platform is DREAM [3], 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 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.

ParadiseEO [6] 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. But it has the same problems that the

previous frameworks, not lifecycle managment 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 [20] is the most important 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. Moreover, the used plug-in system does not uses a public specification like OSGi. And also it is a proprietary software, like their execution environment, the .NET platform [7].

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

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 not have an event administration mechanism and they are not service-oriented, so they not take advantage of this paradigm.

3 Used Technologies

OSGi features can be useful in the development of distributed algorithms, so this section describes the tools and communication protocols employed within the presented framework.

3.1 OSGi

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 framework provides APIs for the management of services that are exposed or used by the bundles.

A **bundle** is a file that contains compiled and packaged classes and a configuration file. This file indicates which classes imports or exports the bundle.

The most important concept in OSGi is the **service**. The services allow to connect *bundles* in a dynamic way, offering a publication-search-connection model. That is, a *bundle* exposes a service by a Java interface, 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*. The Figure 2 shows an example of the OSGi architecture.

It would be useful if this connection could be done out of the source code, so the OSGi also provides **components**. A component is a class inside a *bundle* together with an XML description. This description is interpreted in execution time to create

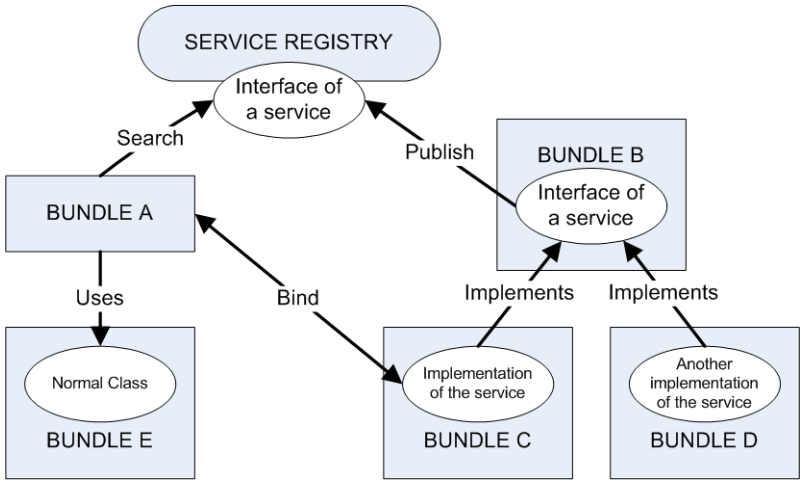

Fig. 2 In OSGi a service can be implemented by several bundles. Other bundles may chose among this implementations using the Service Registry.

and remove services depending the availability of other services, other components or configuration data. The main difference between a component and a normal class inside a *bundle* is that in the second the association between interface and implementation of the service must be defined in the source code, and also the dependency management and the service state detection, being this a tedious work for the programmer. To facilitate this task in OSGi the *Declarative Services* specification [17] arises. It lets that, for example, we could create a class that is not activated until an specific and required service is detected. When this service is active, the class can use it with a *bind* method. It is important to note that implementation will be injected in execution time, not in compilation time.

OSGi also provides event handling with an implementation of the event broker pattern, the **Event Admin**. It is an interbundle communication mechanism based on a publish-and-subscribe model. Some bundles publish events and some other bundles can read this events, being this task transparent for the programmer: the sender does not need to know who are listening their events, and the listener can filter among the events.

3.2 R-OSGi

One of the problems of OSGi today is its inability to invoke remote services and its lack of a distributed module management, so other protocols adapters have been created, like JINI [22] and UPnP [16]. Nevertheless these approximations can be considered invasive, due to their requirement of re-structuring the application. This

is the reason that R-OSGi arises [19]. R-OSGi is a middleware layer inside OSGi that lets a more transparent distribution of the application parts simply distributing its software modules. Inside the OSGi framework the remote and local services are indistinguishable, so the existent OSGi applications can be distributed without modification using R-OSGi. Moreover, this middleware does not imposes client-server assignation because the modules relationship is symmetric. The authors have demonstrated that the R-OSGi is similar to the highly optimized Java 5 RMI implementation and two times faster than UPnP.

R-OSGi creates client *proxies*. For the client of a service, this proxies behave as local services and they also are provided by locally instantiated bundles. However a *proxy bundle* redirects all received calls to the original service that resides in the remote machine, and propagates the result of the call back to the client. An example of this architecture is shown in Figure 3. The events used in the previously explained Event Admin are also transmitted in a transparent manner: the senders and the receivers of the events do not need to add anything to the program code in order to receive the events among distributed nodes, because they do not need to know where the nodes are.

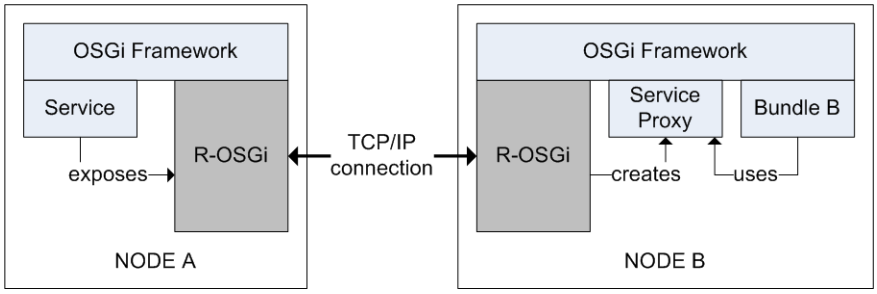

Fig. 3 Architectural overview of R-OSGi. The node B uses *Service Proxy* as a normal service.

4 OSGiLiath Platform

This section dives in 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 usage and offer to programmers the next features:

- Easy interfaces
- Asynchronous data sending/receiving
- Component Oriented Programming

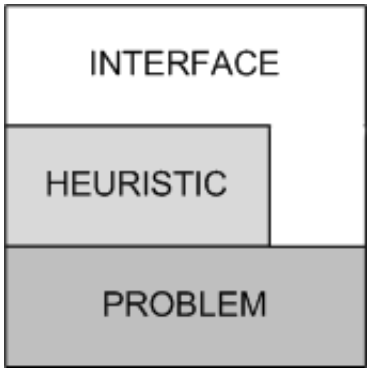

Fig. 4 Defined layers in OSGiLiath

- Client/Server or Distributed Model
- Paradigm independent
- Declarative Services
- Remote event handling

The source code is available in `http://atc.ugr.es/~pgarcia`, under a GPL license. The environment presented in this work lets defining implementations for specific problems using the OSGi benefits. Its architecture is composed by three levels or layers: *Interface*, *Heuristic* and *Problem* (see Figure 4).

The *Interface* layer provides a hierarchy of interfaces defined to develop distributed heuristics. Some examples are *Algorithm*, *Distributed Algorithm*, *Solution*, *Problem*, *Input Data* or *Parameters*. It also provides interfaces and objects for distributed programming, like *Server*, *Node* or *Task*. This class hierarchy, exported as a *bundle* is well-defined, because it will be the basis to construct the full application. As every *bundle*, it can export these interfaces to be used by another bundles. These interfaces must be implemented in the next framework level, the *Heuristic* layer. Using the OSGi Declarative Services Specification [17], the instances of these implementations will be activated when they are necessary and accessed among them. Finally, (*Problem*) layer defines what problems will be executed in the framework.

Furthermore, using the R-OSGi functionality we can add the feature of distributed applications in an undetermined number of nodes. In this case, we have to implement several *Tasks*, whose implementation can be in different nodes. Given the platform architecture the *Heuristic* or *Problem* layers could be in remote nodes, so the user could define new problems or heuristics and automatically bind with the necessary elements to execute.

5 Development Example Using OSGiLiath

As an example of usage of the presented framework, a tabu search and a distributed genetic algorithm have been developed to solve the *Vehicle Routing Problem* (VRP) and the capabilities of the framework have been tested. Due to space restriction we refer the reader to [8], which explains the implemented Tabu Search and a more formal problem approach. The Tabu Search is a sequential algorithm, while the Genetic Algorithm uses a distributed island model: every node executes a separate algorithm and swaps individuals with the other nodes.

5.1 Specifying an application

The first step to develop in OSGiLiath is to implement the interfaces defined in previous sections to build specific implementations. For example, *TabuSearch* and *DistributedGeneticAlgorithm* are implementations of *Algorithm* and *DistributedAlgorithm*. The implementation of each algorithm must be as general as possible, due to the implementation of the problem to solve its developed in the next level. So, in this layer more interfaces are defined, like *StopCriterion*, *TabuList*, *Mutation*, *Fitness* or *IndividualInitialization*. This level uses the feature of Declarative Services in order to obtain automatically the implementation of that interfaces.

5.2 Specifying the problem

Finally in this level the problem to solve is specified in more detail. For example we have implemented the interfaces *Problem*, *Individual*, *Crossover* or *TabuList* with the *ProblemVRP*, *IndividualVRP*, *CrossoverVRP* and *TabuListVRP* classes. Due to they have been exposed as declarative services, when they are activated, the services defined in the previous level also will be activated.

All work developed in this level can be added to the base platform, since all component are clearly differentiated, and other developers could implement their own problems to apply the Genetic Algorithm or Tabu Search, or add new algorithms to solve the VRP problem.

5.3 Adding distributed capacity

Using declarative services implementation of *Task* interfaces are created. In the Tabu Search example, remote nodes could search the best neighbourhood of the current solution, receiving a movement list and the Tabu List, but due to the canonical Tabu

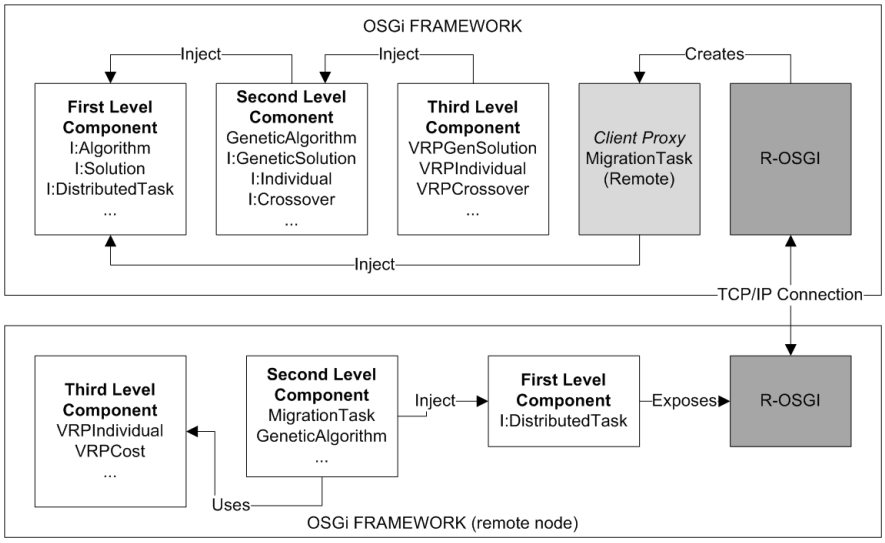

Fig. 5 OSGiLiath architecture. The user can implement heuristics and problems interfaces.

Search is difficult to parallelize because of the latency we only have tested the sequential algorithm.

In the case of the Genetic Algorithm, every certain number of iterations each node receives one of the best individuals of the other nodes, randomly selected. Thanks to the OSGi features, every service can be distributed in a transparent way (operators, algorithms, iniers, schedulers). The programmer does not need how the communication is performed or where the implementation is, he only needs to know the interface of the service.

All the nodes have knowledge of what the other are doing, thanks to the OSGi event handling mechanism. Whenever an iteration or algorithm over, events are published and read by the others, so the algorithms can synchronize or inform to others about their results.

Along with the challenges of OSGi [12], there exists the issue of the loss of abstraction in the development of the interfaces of our framework, so a study to find balance between cohesive and loose coupled hierarchy will be performed in future. In problem-specific algorithms, where exist a tightly coupled association, the usage of events and automatic communication mechanisms will be helpful if they are used properly.

6 Experiments

Once the algorithms development have been explained we present the obtained results. We have to say that the presented work is a proof-of-concept, so these results are shown as example. We have used a 4 nodes cluster, each one of them with a 1.6 GHz, 4 GB RAM and Java version 1.5. The common parameters for the algorithms are a stop criterion of 60 iterations without improve the best solution and random initial solutions. In Tabu Search the Tabu List have 30 moves. The Genetic Algorithm parameters are: 200 individual population with elitism, migration of one of the 10 best individuals, randomly selected every 10 iterations; mutation probability is 0.5 and a tournament selection for crossover of the 50 best individuals. The instances of the two problems have been extracted from [4].

The obtained results are shown in Table 1. As can be seen, the Genetic Algorithm results outperforms the Tabu Search, due to the used crossover swaps complete routes, unlike the Tabu Movements, that moves an unique shop in the routes. The time taken in the sequential Genetic Algorithm also is lower than the sequential Tabu Search.

However, the purpose of this work is not perform an analysis of the presented algorithms, but show the ease of using this framework in the distributed algorithm development.

Table 1 Result table for the experiments (average $\pm$ standard deviation)

Nodes	Cost	Iterations	Time (s)
		Tabu Search	
1	2330.18 ± 86.41	312.13 ± 20.26	224.70 ± 11.61
		Genetic Algorithm	
1	2318.83 ± 72.89	4222.60 ± 435.04	100.43 ± 10.51
2	2268.11 ± 74.53	4759.52 ± 798.54	113.33 ± 87.80
3	2223.18 ± 54.13	4903.66 ± 338.40	128.94 ± 65.67
4	2212.24 ± 29.85	4740.20 ± 278.45	124.85 ± 98.20

Every experiment was executed 10 times

7 Conclusions and future work

This work presents an environment for the development of distributed algorithms extensible via plug-ins and based in a wide-accepted software specification (OSGi). OSGi features (declarative services, dynamic life-cycle management, or package abstraction) are used to easily create algorithms in a layered way. Moreover, it uses R-OSGi to develop distributed services. We have shown the Tabu Search and the Genetic Algorithm implementation as an example.

As future work an automatic generated GUI will be developed to dynamically control which problems, algorithms or parameters to use. A study about scalability using other algorithms (like GRASP, Scatter Search, Ant Colony Optimization and others) will be performed. Also, 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 to centralize all new implementations of problems and algorithms to let the distribution within the base platform, so the users just have to write the level 3 classes to solve particular problems. An 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 chosing one implementation or another depending on several parameters or, for example, using Genetic Programming to evolve and hybridize algorithms.

Acknowledgements Supported by projects AmIVital (CENIT2007-1010) and EvOrq (TIC-3903).

References

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

[2] Alliance O (2004) OSGi alliance Available at: `http://www.osgi.org/`

[3] Arenas M, Collet P, Eiben A, Jelasity M, Merelo JJ, Paechter B, PreußM, Schoenauer M (2002) A framework for distributed evolutionary algorithms. In: Parallel Problem Solving from Nature PPSN VII, pp 665–675

[4] BranchAndCutorg (2003) Vehicle routing data sets Available at: `http://branchandcut.org/VRP/data/`

[5] Buyya R (1999) High Performance Cluster Computing: Architectures and Systems. Prentice-Hall

[6] Cahon S, Melab N, Talbi E (2004) ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. Journal of Heuristics 10(3):357–380

[7] Escoffier C, Donsez D, Hall RS (2006) Developing an OSGi-like Service Platform for .NET. In: 3rd IEEE Consumer Communications and Networking Conference, Vols 1-3, pp 213–217

[8] Esparcia-Alcázar AI, Cardós M, Merelo JJ, Martínez-García A, García-Sánchez P, Alfaro-Cid E, Sharman K (2009) EVITA: An integral evolutionary methodology for the inventory and transportation problem. Studies in Computational Intelligence 161:151–172

[9] Foster I (2002) The Grid: A new infrastructure for 21st Century Science. Phisics Today 55:42–47

[10] Gaspero L, Schaerf A (2001) Easylocal++: an object-oriented framework for the flexible desgin of local search algorithms and metaheuristics. In: Proceedings of 4th Metaheuristics International Conference (MIC'2001), pp 287–292
[11] González JR, Pelta DA, Masegosa AD (2009) A framework for developing optimization-based decision support systems. Expert Systems with Applications 36(3, Part 1):4581 – 4588
[12] Kriens P (2008) Research challenges for OSGi Available at: http://www.osgi.org/blog/2008/02/research-challenges-for-osgi.html
[13] León C, Miranda G, Segura C (2009) Metco: A parallel plugin-based framework for multi-objective optimization. International Journal on Artificial Intelligence Tools 18(4):569–588
[14] Luke S, et al (2009) ECJ: A Java-based Evolutionary Computation and Genetic Programming Research System. Available at http://www.cs.umd.edu/projects/plus/ec/ecj
[15] Marples D, Kriens P (2001) The Open Services Gateway Initiative: An introductory overview. IEEE Communications Magazine 39(12):110–114
[16] Miller BA, Nixon T, Tai C, Wood MD (2001) Home networking with universal plug and play. IEEE Communications Magazine 39(12):104–109
[17] OSGi Alliance (2007) Declarative services specification pp 281–314, available at: http://www.osgi.org/download/r4-v4.2-cmpn-draft-20090310.pdf
[18] Papazoglou MP, Van Den Heuvel W (2007) Service oriented architectures: Approaches, technologies and research issues. VLDB Journal 16(3):389–415
[19] Rellermeyer JS, Alonso G, Roscoe T (2007) R-osgi: Distributed applications through software modularization. vol 4834 LNCS, pp 1–20
[20] Wagner S, Affenzeller M (2004) Heuristiclab grid - a flexible and extensible environment for parallel heuristic optimization. In: Proceedings of the International Conference on Systems Science, vol 1, pp 289–296
[21] Wagner S, Winkler S, Pitzer E, Kronberger G, Beham A, Braune R, Affenzeller M (2007) Benefits of plugin-based heuristic optimization software systems. vol 4739 LNCS, pp 747–754
[22] Waldo J (1999) The Jini architecture for network-centric computing. Communications of the ACM 42(7):76–82
[23] Wall B (1996) A genetic algorithm for resource-constrained scheduling, Ph.D. thesis, MIT Available at: http://lancet.mit.edu/ga