I.J. Intelligent Systems and Applications, **2014, 04, 11-25**
Published Online March 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2014.04.02

The Conception of the New Agent-Based Platform for Modeling and Implementation of Parallel Evolutionary Algorithms

Sara Sabba, Salim Chikhi
Department of Fundamental Computing and its Applications, Faculty of New Technologies of Information and Communication, MISC Laboratory, Constantine 2 University, Algeria
E-mail: sara.sabba,chikhi@umc.edu.dz

***Abstract*—** Evolutionary algorithms (EAs) are a range of problem-solving techniques based on mechanisms inspired by biological evolution. Nowadays, EAs have proven their ability and effectiveness to solve combinatorial problems. However, these methods require a considerable time of calculation. To overcome this problem, several parallelization strategies have been proposed in the literature. In this paper, we present a new parallel agent-based EC framework for solving numerical optimization problems in order to optimize computation time and solutions quality.

***Index Terms*—** Evolutionary Algorithm, Combinatorial Problems, Agent-Based Software, Ontology, Parallelization Strategies, JADE Framework

I. Introduction

Combinatorial optimization holds a substantial place in the field of computer sciences and many other engineering sciences. This field of study results from the intersection between operational research, theoretical computing and mathematical programming. Combinatorial optimization aims at finding the best solution among a finite number of choices by minimizing or maximizing an optimization function with or without constraints. Thus, combinatorial optimization problems are often complex and NP-hard, they are characterized by the exponential number of combinations to be explored, which requires relatively considerable time to find a good solution, especially when the number of entries is large.

The necessity to quickly find reasonable solutions to many of these problems has led to the development of several approximation algorithms which include Evolutionary Computing (EC) algorithms. An Evolutionary Algorithms (EAs) are a stochastic population-based search techniques inspired from the evolutionary process and natural selection of the biological populations. These algorithms combine Genetic Algorithms (GA) [18], Evolutionary Strategies (ES) [32, 35], Evolutionary Programming (EP) [27] and Genetic Programming (GP) [22, 23]. They all share the same conceptual based natural evolution simulation, but differ in the breeding strategy and representation on which EA operate [21].

The EC algorithms cannot significantly reduce the exploration time of the search space. Since, they are often expensive in calculation time, specially for large problems. Consequently, the parallelism concept is an indispensable for executing algorithms in parallel machines. In fact, the parallelism aims to address both issues. The first is to reduce the search processing time of large problem instances. The second is to improve the solutions quality by exploiting more promising search regions in reasonable time. Several parallelization strategies of the EAs have been proposed in [9, 8]. Whose idea is to launch concurrent execution of the inner loops of the algorithms, to decompose the problem domain (research space) into sub-spaces where each one is executed in different processors, or to launch several search process simultaneously (Multi-threads search) with different degrees of synchronization and cooperation.

For the last few years, a larger number of optimization frameworks have been developed such as: DREAM [29], MALLBA [2], ECJ [36], BEAGEL [6, 12], J-DEAL [19], EASY LOCAL + + [15], MAFRA [25], ParadisEO [5]. According to certain criteria these frameworks can support the class of evolutionary algorithms (EA) [36, 6, 19, 29, 25, 2, 5], and/or the class of Single-Solution based Algorithms (SSL) [15, 29, 25, 2, 5]. Actually, each software tool uses its own strategy to run its various methods, starting with the choice of programming language (most are object-oriented JAVA or C++) then specifying the cooperation and hybridization strategy between the resolution methods (EA/EA) [36, 6, 19, 29, 25, 2, 5], (EA/SSL) [2, 5], (SSL/SSL) [15, 2, 5], and finally, the choice of execution mode: sequential execution [15, 25, 2, 5] and / or parallel execution [36, 6, 19, 29, 2, 5].

In fact, most frameworks implement the parallelism as communicating objects using the following parallelism supports: threads [36, 29, 5], sockets TCP/IP [6, 19, 29], PVM/MPI [5], Netstream [2] and Condor /MW [5]. Up to our knowledge, there has been no literature applications works concerning the field of EC frameworks and agent-based approach.

In our proposition, we adopt the multi-agent paradigm to develop a framework of parallel evolutionary algorithms. In this context, we introduce a new agents-based framework architecture for modeling and implementing the parallel evolutionary algorithms. The proposed idea is developed under JADE middleware at aiming to facilitate the implementation of any agent components.

The paper is structured as follows: we analyze at first some considerations about the Evolutionary Computing framework design. Thus, we provide the technical background used for the development. Then, we explain our idea showing the basic motivations and goals. Therefore, we detail the agent-based framework architecture. Afterwards, we describe system operation in serial and parallel executions. After this presentation, we illustrate some functionality used in the implementation then we finalize with a conclusion.

II. Evolutionary Computing Framework

2.1. EC framework Design

From a software engineering point of view, an EC can seen as an abstract class of algorithms, and its different variants like GP, GA, and ES can be seen as concrete instantiations [11]. In fact, the best way of modeling EC algorithms it is in the form of a framework. A framework is a set of cooperating classes that make up a reusable design for a specific software domain [14]. The development of such generic software is very complex. Since, it requires too much effort to developers which must be taken multiples aspects into account. Gristan Gagn é and Marc Pariscan [11] are mentioned six criteria to qualify the genericity of the EC frameworks. They cite: generic representation, generic fitness, generic operations, generic evolutionary model, parameters management and configuration output. In the same paper, the authors compare some existing frameworks according to the previous criteria where they are judged ECJ 1.3 [36] and Open Beagle 2.2.0 [12] as real generic EC tools compared to EO [20], GAlib [42], lil-gp [31] and GPLAB[37, 38] software.

Actually, each EC framework developer groups use their own strategy to design and implement their generic software. However, most of them agree that their tools must meet the following criteria:

1) *The representation of all EC variants:* The framework must define all EA individual structures (vector, tree) of all possible types (bit, string, double, integer).
2) *The genericity of the solution methods:* The generic EC software must provide the full control structure of the invariant part of the algorithms. The design and the implementation of the solution methods should be completely independent to the problems they are dedicated to solve. As consequence, the user will only develop a minimal code of the specific problem (fitness function). The framework may also implement some specific problem operators such as the crossover and the mutation operators of the traveling salesmen problem (TSP).
3) *The flexibility and adaptability of the software components design:* The framework architecture must be easily extended to define new components and add new paradigms without modifying the base framework structure.

2.2. Related Works

For the last few years, several EC software tools have been developed to solve the complex problems. Some proposed frameworks are specialized in one EC variant like: GAPS [24], lil-gp [31] and GPLAB [38] for genetic programming, GAlib [42] for genetic algorithm, JGAP [28] for both genetic algorithms and genetic programming, MOEA [17] for multi-objective evolutionary algorithm. Some other are the generic tools, that is, they ensure most genericity criteria mentioned in the previous section. This category includes : JCLEC [41], ECJ [36], Open Beagle [6, 12], ParadisEO [5], MALLBA [2], ect. We present below the characteristics of some successful open source frameworks:

ECJ: is an open source EC software tools written in Java, it is one of the most popular EC frameworks. ECJ is highly flexible, it is created to simplify and facilitate the implementation of any kind of EC algorithms using the configuration file state (i.e., the algorithm parameters can be dynamically modifying at runtime). Furthermore, ECJ implements several interesting features including multi-objective optimization, particle swarm optimization and several parallel strategies executions like asynchronous island models over TCP/IP, Master/Slave evaluation over multiple processors, with support for generational, asynchronous steady-state, and co-evaluation distribution.

Open Beagle: is a C++ framework for developing any variant of EC. Open Beagle follows the same mechanism utilized in ECJ. Since, it bases on XML configuration files to configure dynamically the algorithm parameters. Moreover, Open Beagle has a strong flexible structure based on object-oriented extension of C++ and the standard library (STL) to build different modules of EC flavors. Indeed, its extension Distributed Beagle offers an environment for parallel and distributed ECs using a new Master/Slave architecture [13] target toward powerful clusters over TCP/IP sockets.

ParadisEO: is a successful metaheuistics tools developed in C++ to support a broad range of algorithm including: EAs, local search (LS) algorithms, exact methods (EM) and Particle Swarm Optimization (PSO).

ParadisEO offers a high level of flexibility and adaptability and openness, it was designed to support several execution models namely serial, parallel, multi-objectives and hybrid applications. Furthermore, these models are portable on distributed memory machine as well as shared memory through the Posix threads multiprogramming or PVM/MPI communication libraries.

MALLBA: as ParadisEO tools MALLBA is a generic framework for serial, parallel and hybrid metaheuristics (EA, LS and EM). This library was developed as software skeletons implemented as a set of C++ classes to represent object abstractions of the resolution techniques. The main feature of this framework is the integration of all skeletons under the same design principal in order to facilitate the switch from sequential to parallel optimization engines, and also to provide more powerful hybrid between skeletons. MALLBA has a specific middleware called Netstream. It is a C++ library implemented on MPI to facilitate the communication of parallel applications in LAN and WAN environments. The different parallel implementation strategies of exact, heuristic and hybrid algorithms are clearly explained in [1].

JCLEC: is a recent EC framework developed in Java language. It is characterized by a strong architecture that was designed of a high-level of reusability and adaptability using several design patterns. In fact, JCLEC gives more intention to the graphical user interface (GUI). Accordingly, it offers to the user two ways to execute its applications. The first is by using XML configuration files. The second is by using GenLab which is a graphical user interface allowing inexperienced users to configure and develop easily its algorithms. In addition, JCLEC contains also several implementations of evolutionary algorithms namely: classic EAs, multi-objective algorithms, memetic algorithms and niching algorithms.

As we can see, each existing framework adds new interesting features compared to the other software in order to facility and simplify the development of EC algorithms (namely XML configuration file, GUI, middleware, ect.). Moreover, when we focus our attention on a frameworks structure design, we can see that all frameworks use probably the same classes to modeling and implement the basic EC software components. First, they define individual type and structure «Genotype». Secondly, they define EA operators using «abstract classes» (i.e., selection, replacement and Recombination). Finally, they define evaluator class which must be an «interface» implementable by the user to specify the objective function of the problem dedicated to solve.

III. Technical Backgrounds

3.1. Agent-Based Software Engineering

The agent-based software engineering (ABSE) technology has become a new programming paradigm based on"a social view of computing" [44]. It is naturally more suitable to develop certain types of applications namely distributed, dynamic, mobile and autonomous computing systems. ABSE focuses on the collective behaviours of multiple autonomous and flexible entities which have the ability to cooperate, negotiate and communicate with each other by sending and receiving messages in order to achieve a common goal. Furthermore, agent technology is certainly offering any advance in software development [43], which can help to enhance the modularity, reusability flexibility of applications. Therefore, several platforms have been invented to facilitate the development of agent-oriented system, these environments are usually used a specific protocols, language and technology conforming to FIPA specification [10].

3.2. JADE Framework

One of the important decision before the implementation of an agent software is to find the most suitable development environment for the construction and the deployment of the application. Actually, there are three successful agent platforms: ZEUS [26], AGENT BUILDER [40] and JADE [33]. In our proposition we have oriented towards JADE framework. In fact, JADE (Java Agent Development Framework) [10] is an open-source software framework developed in Java language and distributed by TILAB (Telecom Italian Laboratory). It is devoted to simplify the implementation of multi-agent applications conforming to FIPA (Foundation for Intelligent Physical Agents) specifications. The agent platform keeps a high performance for development of distributed system. It provides effective and light-weight communication between agents inherently distributed through Java runtime environment.

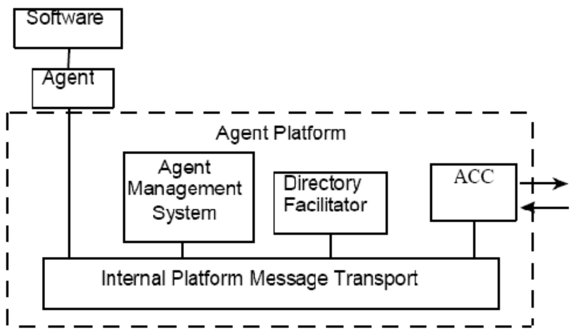


Fig. 1: Reference architecture of FIPA agent Platform [11]

JADE platform is composed of agent containers that can be distributed over the network. There is a special container, called the main container, which represents the bootstrap point of a platform. When the main container is launched, three special agents are automatically started (1) Agent Management System

(AMS) that manages the operation of an Agent Platform (AP) such as: the creation deletion of agents, the registration (obtain an AID) of agents and the migration of agents to and from the AP, (2) Directory Facilitator (DF) that provides a yellow pages service to other agents and, (3) Agent Communication Channel (ACC) that provides Message Transport Service (MTS) between agents using Message Transport Protocol (MTP), i.e., supports inter-agent communication and inter-operability within and across different platforms.

In addition, each agent developed under JADE framework performs a task that is called "Behaviour" [3], where it is implemented as an object of a class that extends jade.core.behaviours.Behaviour. According to the task to be achieved, each agent can run several behaviours of different types. Indeed, there are several types of behaviours [3], in our approach we are interested to define:

1) *One-shot Behaviour*: where agent executes its task only once then it is destroyed.
2) *Cyclic Behaviour:* where agent runs its task in a repetitive way until the satisfaction of a stopping criterion specified by the programmer.
3) *Sequential Behaviour:* where agent has the ability to execute several behaviours of different types one by one.
4) *Parallel Behaviour:* where agent allows executing several behaviours in parallel.

IV. AFPEA: Agent Framework for Parallel Evolutionary Algorithms

AFPEA is an architecture of a new agent-based EC framework (implemented in JAVA under JADE middleware). The proposed system is devoted for the design and the implementation of parallel evolutionary algorithms. Actually, it aims to benefit from the advantages of some interesting features of the agents' paradigm missing in the classical approaches in order to facilitate and accelerate the development of parallel algorithms.

4.1. Motivations

Our motivations are based on interesting services provided by JADE:

1) *Portability:* AFPEA is developed under JADE framework which implements agents as one Java thread and Java events. Therefore, it can be supported by the different material architectures and any operating systems where a Java Virtual Machine exists.
2) *Directory Facilitator DF:* The DF is a centralized registry of entries, it is allows saving the ID name and the service provided by each agent connected to the system (in the host and in the distributed machines). In fact, it helps agents to exploit required services.
3) *Persistent-Delivery:* This service allows buffering and persistent storage of undelivered messages [4].
4) *Fault tolerance:* This is activated on each node host it allows replicating the Main Container for fault tolerance purposes [4].
5) *Ease of programming:* Depending on programmatically view, the development of parallel applications is very easy using agent technology then object-oriented technology. Actually, agent networking in JADE is implemented at high abstraction. The main information needed by agent to accomplish its service or its communication (as the address of the destination agent) can be easily recovered by DF or AMS. On the other hand, JADE packages facilitate the development of the main parallel strategies using a few lines of code. For instance, the agent responsible for asynchronous distribution can be simply used blockingReceive() instruction in order to block its future behaviour (the next distribution) until receives the responses. However, if the agent did not use the instruction, the distribution will be synchronous, whenever a machine completes its work it reassigns directly another one.
6) *Performance execution in minimum time:* This characteristic is the results of the previous one, it is proved in [7] where the authors compare the Round Trip Time of the testbed implemented using RMI mechanism (without the facilities offered by the JADE middleware) with the other implemented under JADE, the obtained results showed good JADE message system performance for developing distributed applications.

4.2. Goals

This section contains a synthesis of the main proprieties of AFPEA which are related to the design objectives scheduled for its development.

1) *Modular:* The EC library was designed in a generic way using several design patterns. The objective is to maintain the maximum separation between the solution methods and the problem to be solved in order to secure the flexibility of the system.
2) *Flexibility:* This feature is the results of the previous one. It allows the user to reuse or change existing operator methods or add other ones in order to implement the new algorithms.
3) *Scalable parallelization strategies supported:* AFPEA is not restricted to implement the different parallelization evolutionary algorithms strategies existing in the literature but also to implement other strategies proposed in the academic researches.

4) *Dynamic execution and Performance calculation:* Our first aim in this proposal is to ensure the intelligent and the dynamic execution of parallel evolutionary algorithms. Indeed, several agents have been proposed in order to insure a high quality of the final results in minimum run time (the role of these agents is explained in the next section).

V. System Architecture

The framework structure can be divided in four parts:

5.1. Framework Agents

AFPEA is composed of a set of heterogeneous agents which allow cooperating with each other to achieve the main goal which is minimizing the computation time of the evolutionary algorithms. These agents are regrouped as follows:

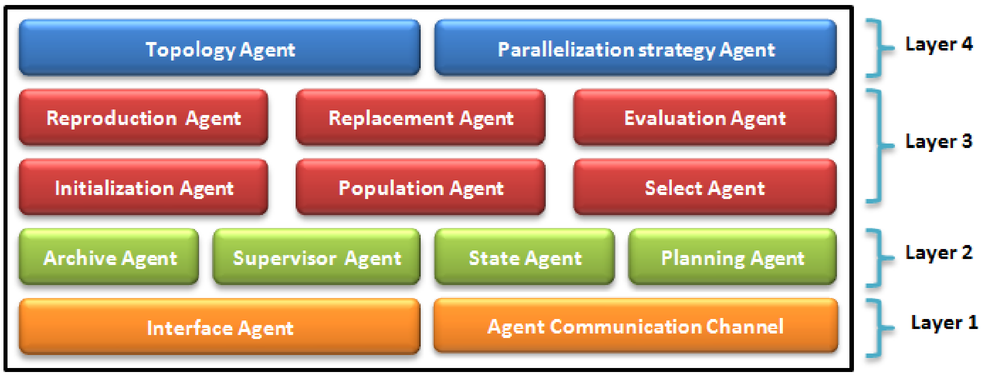


Fig. 2: AFPEA multi-layers architecture

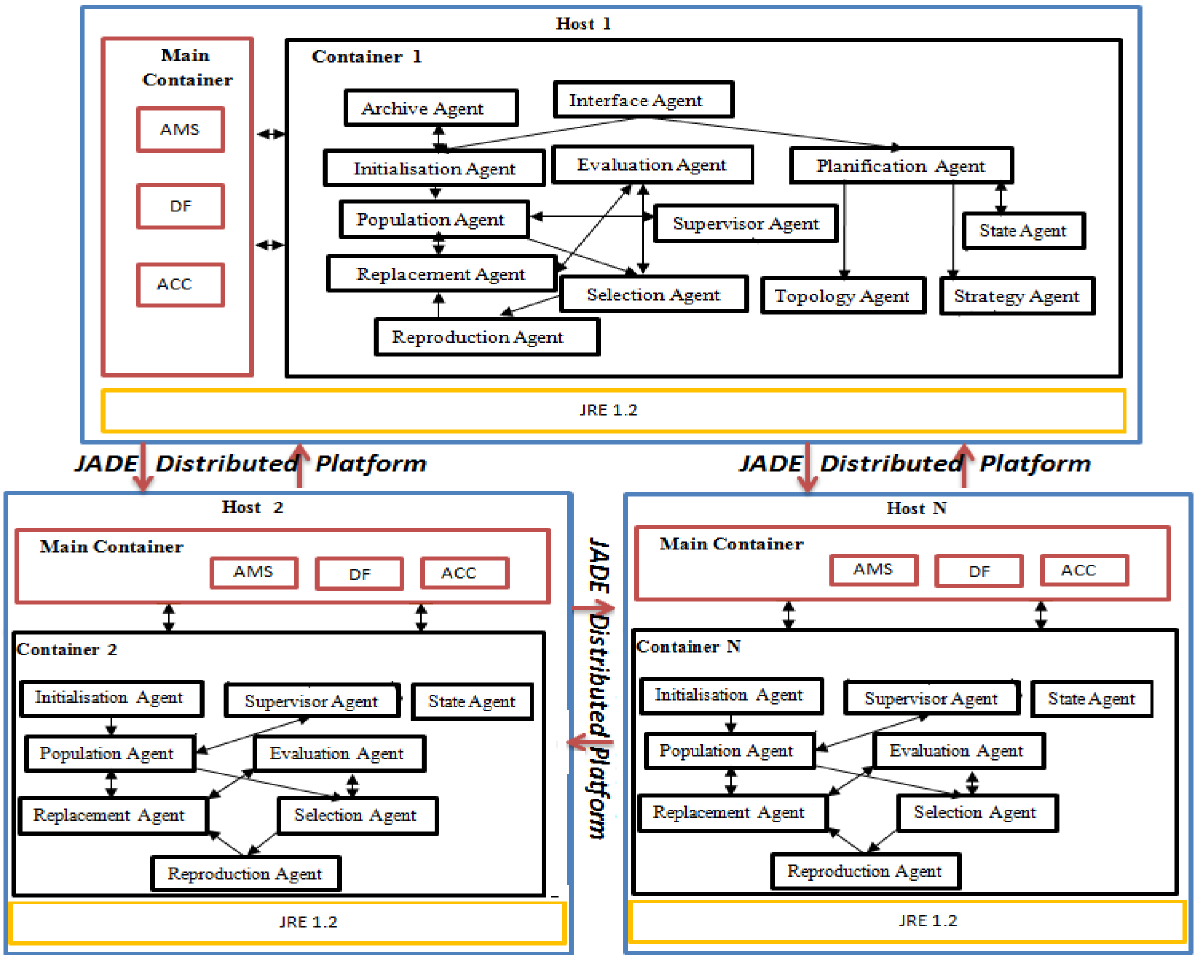


Fig. 3: Agent resolution in parallel mode

Layer 1: is responsible for all communication performed in the system, i.e., between the user and the system and between the distributed agents. This layer is composed of two agents: Interface Agent and Agent Communication Channel ACC (provided by JADE).

Layer 2: is composed of four agents: Archive Agent, supervisor Agent, State Agent and Planning Agent which cooperate to ensure and keep the performance, the reliability and the rapidity of the performed calculations.

Layer 3*:* is loaded to execute evolutionary operators through the six agents namely: Initialization Agent, Population Agent, Select Agent, Reproduction Agent, Replacement Agent and Evaluation Agent.

Layer 4: is responsible for monitoring parallel execution via its two agents: Parallelization Strategy Agent and Topology Agent.

The proposed agents are distributed in two levels. The first tranche is located in the “Master” and it is responsible for monitoring. The second tranche is located in “Slavers” (Work/stations) and it is responsible for executing algorithm operators. Figure 3 illustrates the global architecture design of the AFPEA under JADE multi-agent platform.

We describe below the role of each agent:

1) Interface Agent

Generally, to exploit any EC framework the inexperienced user utilizes directly the graphical user interface, or the XML configuration file to specify the parameters of the EC algorithm. Except the experienced programmer, that may access to the framework code source for implementing new operators to create new algorithms.

2) Initialization / Selection / Reproduction / Replacement Agents

They are reactive agents, responsible to execute the different operators of the evolutionary algorithms using its local memories (that contain the implementation of the most operator methods).

3) Evaluation Agent

It is a reactive agent, which uses an objective function (implemented by the user) to calculate the fitness of individuals (solutions).

4) Population Agent

It is a reactive agent that allows to:

- Represent the population that will be reproduced in every generation.
- Save the best solution in every generation.
- Monitor the evolution of the population, if the population has not changed (the algorithm is stuck in a local minimum) after K generations, it sends a report to the supervisor agent to change the current parameters.
- Control the stopping criterion of execution.

5) State Agent

It is a reactive agent which is activated only at slave stations. This agent allows following the change of the machine’s state in order to send its report to the Planning Agent.

6) Planning Agent

Planning Agent is an autonomous agent that reacts only when the user (or the system) chooses parallel execution mode, its main purpose is to select the least loaded machines to participate in the calculation in order to avoid falling on the wrong distribution. Consequently, this agent:

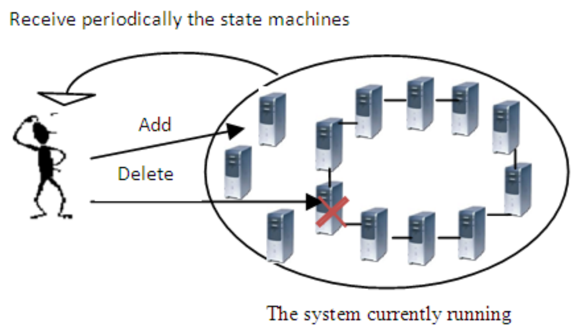

Receive periodically the state machines
Add
Delete
The system currently running

Fig. 4: The Planning Agent that ensures the dynamic system

- Uses the DF agent to know the list of the state agent connected to the network.
- Sends to each state agent a request to get machine’s state (load rate, failure).
- Focus on a threshold loading rate to construct a list of the most adequate work stations to participate in the calculation.
- During execution, it can add more machines when they are unoccupied (or loaded is less than threshold), it can also delete stations in case of problems.
- It sends the list and the number of the participant machines to Topology Agent and Strategy Agent.

7) Supervisor Agent

It is an intelligent agent that ensures the performance of the final results; it is responsible for:

- Changing the execution mode from the sequential mode to the parallel mode if it is necessary (the problem size is very large).
- Changing the parameters of the running algorithm if it is receive a message from Population Agent. In fact, to ensure this task it can be based on special methods (heuristic) [30] or the parameters stored by archive agent.

8) Archive Agent

It is a reactive agent which allows to:

- Save at the end of each execution the parameters used in the resolution of the combinatorial problem (instance problem name, resolution algorithm, population size, crossover rate, crossover method, etc.).
- If the system finds other parameters which are led to the best solutions the agent will change the old settings. If the system finds other parameters which are led to the best solutions the agent will change the old settings.
- The recorded information can be used by Supervisor Agent to change the current parameters.

9) The Agents Responsible of Parallelization/Distribution Strategies

There are several strategies can be used for parallelizing/distributing the evolutionary algorithms. We cite: the cooperative island model (synchronous or asynchronous, parallel or distributed), the parallelization of the evaluation phase (synchronous or asynchronous), the parallelization of the objective function, etc. In our system, each strategy is represented by an agent, which will be the administrator of the parallelization/distribution processes.

Example 1: Parallelization strategy of evaluation phase

In this strategy the master manages the whole evolutionary sequential process and the slaves evaluate only the new individuals. We explain below the principle tasks of the agent responsible of this strategy.

- Recovers the ID list of Evaluator Agents through DF Agent.
- Divides the population into subPop. The subPop size is an important parameter, since if the individuals are large then a large job size won ́t have any efficiency benefit, however, if they ́re very small the job size will have a huge benefit. Consequently, we can define the Max value of this parameter to ensure the performance execution.
- Sends the subPops to the Evaluator Agents.
- Receives the results
- Sends the evaluated population to the Replacement Agent.

Example 2: Island cooperative strategy

In this strategy the same or the different AEs are executed concurrently to cooperate (exchange the individuals) in order to identify the best solution (each island execute the all EA process).Therefore, the agent responsible of this strategy should send to each island the important parameters including:

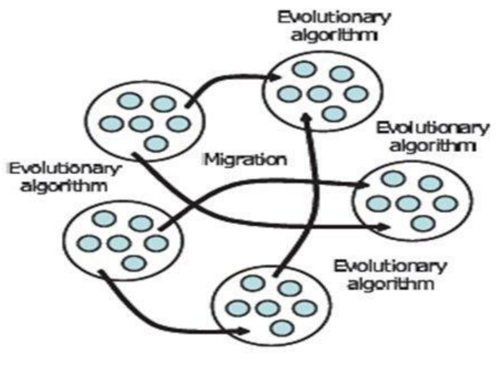


Fig. 5: The island model of EAs

- The population size that will emigrate.
- The select method to choose the emigrate individuals.
- The replacement method to include the immigrant individuals in the current population.
- The migration time of solutions [8]:
  - If the communication between agents is synchronous, the solutions migration is periodically performed according to a specific time interval (a constant number of iteration).
  - If the communication is asynchronous, it associates to each island algorithm a decision criterion for immigration this latter can be re-evaluated in each iteration according to the state of the current population.
- The interconnection ID list to define the agents that will be connected with them to send and receive the solutions (this parameter is defined by topology agent).

The other detail of this strategy is explained in the next section.

10) Topology Agent

It is a reactive agent which is activated only in the cooperative execution mode in the host machine.

- This agent is responsible for the construction of the interconnection lists (between the agents located in different islands) according to the type of network topology (ring, hypercube topology, etc.). These lists contain the addresses of the agents with which must communicate (send and receive the individuals) during the exchange process.
- During execution process the interconnection lists can be changed if the Planning Agent has modified the list of participant work stations.

5.2. Libraries

Each agent existing in the framework uses its local memory to execute its behaviour. Indeed, the local memory is a set of class that defines agent methods.

These classes are designed and implemented in a generic and flexible way in order to create new methods in the future. The main library existing in AFPEA is Java Evolutionary Object Library (JEOL) it contains the generic classes of EA operators including initialization, selection, reproduction and replacement. In fact, each operator is represented by an «abstract class» that implements the most existing operations. As an example, the «abstract selection class» implements *rank, elitist, roulette-wheel and tournament* selection methods. Moreover, the reproduction class is defined by two abstract classes. The first, for representing crossover methods and the second for representing mutation methods. Notice that, the Reproduction Agent may use one or the both classes to execute its behaviour (this choice depends on the category of evolutionary algorithm executed).

5.3. The Problem Representation and Ontology

1) The problem representation

As all existing frameworks the representation of the problem to be solved is structured as follows:

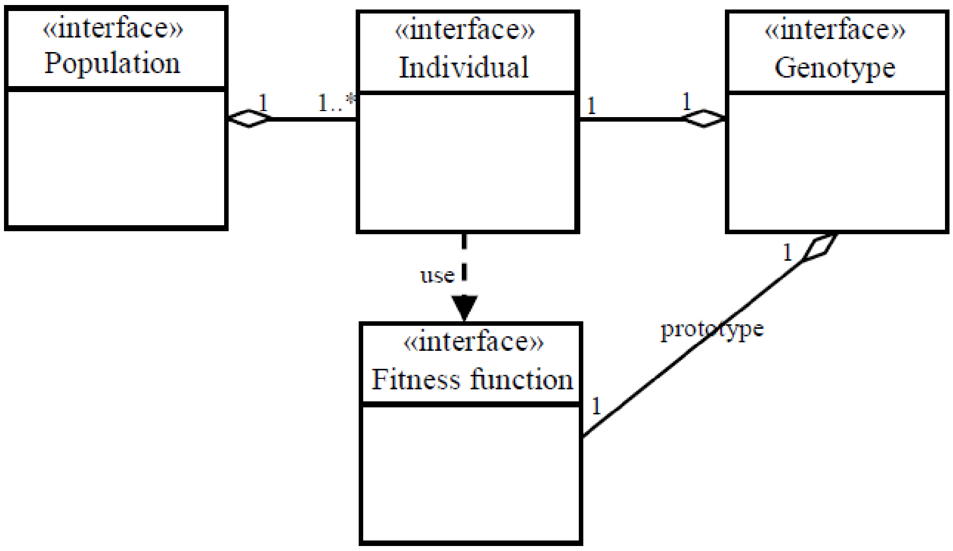


Fig. 6: The problem representation

In the evolutionary algorithm strategy, the research space is defined by the population. This latter is composed of a set of individuals which represent the possible solutions to the optimization problem. The individual structure is completely dependent to the problem type; it can be represented by array (bit, integer or double), tree (syntax or expression) or other type defining by the user by implementing individual «interface» (figure 6). In addition, to calculate the fitness associated to each solution, the user must implement the Evaluator «interface» to specify the objective function to the optimization problem.

2) Ontology

In agent-based software engineering, the only method for communication between agents is by sending and receiving messages which contain a small amount of information. Thus, this conversation is assured by Agent

Communication Language (ACL) that uses a string and byte sequence to represent the exchanged messages. It can be clearly seen that this representation is not appropriate to handle all type of information (where data is usually stored as object form) and it cannot be considered convenient for internal purpose of an agent. To overcome this problem and facilitate the comprehension and the communication between the different agents, JADE framework uses the notion of ontology [16] in order to represent the content of the complex messages inside an agent.

In our agent framework the pivotal information exchanged between agents is the population, for initializing, selecting, reproducing, evolving, replacing, migrating, and/or distributing. Therefore, this complex information (individual list) must be considered as ontology to be shared between the agents framework.

5.4. Parameters Configuration

Before the agent framework is ready to run it is necessary to define all operation parameters to ensure that each agent is aware of its future behaviour. In our system, the user parameters (choose through the interface) will be stored an object and get-at-able through accessor methods (getter and setter). However, in the future we intend to use XML file configuration to avail the advantage offered by this technique which is the dynamic reconfiguration, i.e., the user can change the parameters during execution without recompiling.

VI. The Implementation of the Serial Execution

In the serial execution the agents are situated in one container of the local machine. Only the initialization Agent, the population Agent, the Reproduction Agent and the Replacement Agent will be activated to achieve their mission which is to find the best solution of the optimization problem. The order of the exchanged messages between agents and the type of their behaviours is schematized in figure 7:

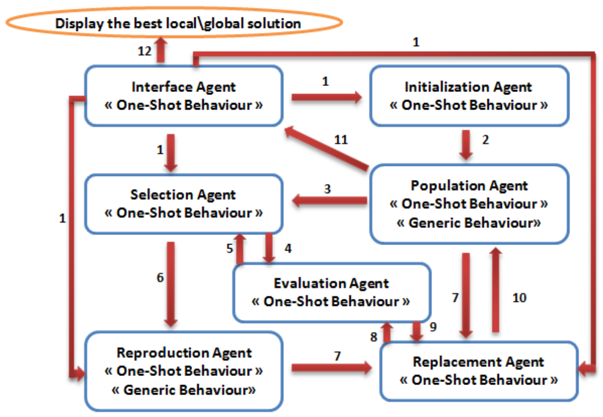

Display the best local\global solution
Interface Agent « One-Shot Behaviour »
Initialization Agent « One-Shot Behaviour »
Selection Agent « One-Shot Behaviour »
Population Agent « One-Shot Behaviour » « Generic Behaviour»
Evaluation Agent « One-Shot Behaviour »
Reproduction Agent « One-Shot Behaviour » « Generic Behaviour»
Replacement Agent « One-Shot Behaviour »
1
2
3
4
5
6
7
8
9
10
11
12

Fig. 7: The sequence of the exchanged messages between agents

By analyzing the behaviour of each agent:

- The Interface Agent has two simple behaviours, the first sends the user's choice of parameters (and methods) to the different agents and the second displays the system state which is currently running (best local/global solution, calculation time, notifications, etc.).
- The Initialization Agent has a simple behaviour which allows initializing the first population using the method and the parameters selected, in order to send the result to the population agent.
- The Population Agent has three simple behaviours: the first receives at each generation the new population produced by a set of genetic agent operators, the second sends the current population to the Selection Agent and Replacement Agent, and the third saves the best global solution at each generation. Finally, the three behaviours are executed cyclically until the satisfaction of the stopping criterion of the evolutionary process.
- The Selection Agent has two simple behaviours that are activated when it receives a message from the Population Agent (current population). The first behaviour performs a simple communication with the Evaluation Agent in order to evaluate the individuals of the current population, and the second uses the results of the previous behaviour to select a sub-population to be sent to the Reproduction Agent.
- The Reproduction Agent has also two behaviours, the first is a cyclic which allows reproducing a new population (through agent operators) until the number of new individuals equals to the desired size. The second is a simple that allows sending the new population to the Replacement Agent.
- The Replacement Agent starts its first behaviour when it receives the current population and the new population. It is allows performing a simple communication with the Evaluation Agent to evaluate the new individuals. The second behaviour of this agent uses the replacement method to find the new population of the next generation.

It is can be clear that, the serial implementation does not add any significant advantage to the EA execution compared with object oriented implementation. However, the run time is same in the both implementations (this is what we will prove in the experimental part).

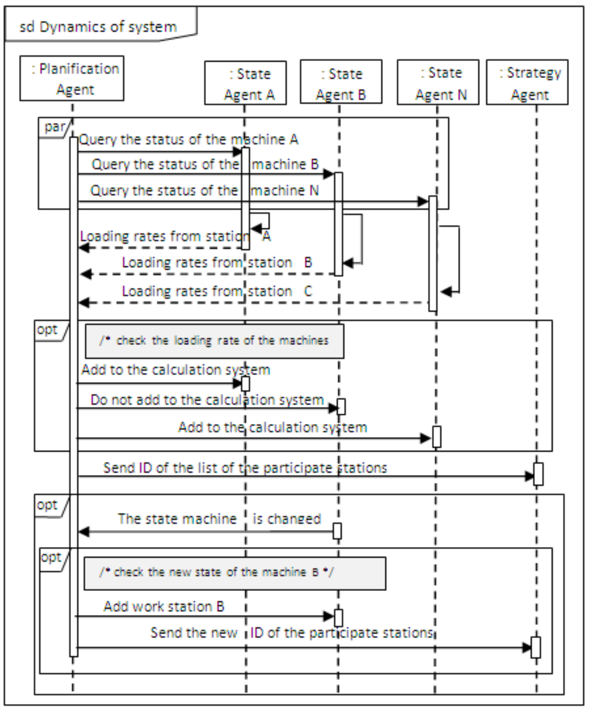

sd Dynamics of system
: Planification Agent
: State Agent A
: State Agent B
: State Agent N
: Strategy Agent
par
Query the status of the machine A
Query the status of the machine B
Query the status of the machine N
Loading rates from station A
Loading rates from station B
Loading rates from station C
opt
/* check the loading rate of the machines
Add to the calculation system
Do not add to the calculation system
Add to the calculation system
Send ID of the list of the participate stations
opt
The state machine is changed
opt
/* check the new state of the machine B */
Add work station B
Send the new ID of the participate stations

Fig.8: The dynamic system modeling

VII. The Implementation of the Parallel Execution

The parallel implementation was designed to be supported by Master/Slave model. Only one machine must be declared as «Host» (see figure 3) so the others as «Slave» machines.

Actually, the execution of the framework in parallel mode is completely dynamic (figure 8). The selection of the participate work stations will be specified during execution through the collaboration between AMS Agent, DF Agent, Planning Agent and State Agents. So, the number of containers (of course agents) may change while the system is operating.

We present below the scenario established between the distributed agents in parallel/distributed population evaluation strategy and synchronous Island model.

7.1. The Implementation of Parallel/Distributed Population Evaluation

At the beginning of execution, the Planning Agent selects the less loaded stations to participate of the calculation where it recovers the addresses of the Evaluator Agents (from the DF and AMS) in order to establish the communication flow with DistPopEval Agent. At the same time, the Initialization Agent initializes the population and sends it to the Selection Agent to choose some individuals for recombining by the Reproduction Agent. This latters, sends the new population to the DistPopEval Agent to distribute the evaluation phase (figure 9).

The DistPopEval Agent will run when it receives the addresses of the Evaluator Agents and the new individuals. It has three beahviours. The first is simple which allows dividing the population into subPops according to the load charge of each machine. The second is cyclic that allows sending the subPops to the Evaluator agents until the completion of the evaluation phase, and the third is simple which allows sending the results to the Replacement Agent in order to replace the current population. Notice that, each Evaluator Agent executes the objective function implemented by the user.

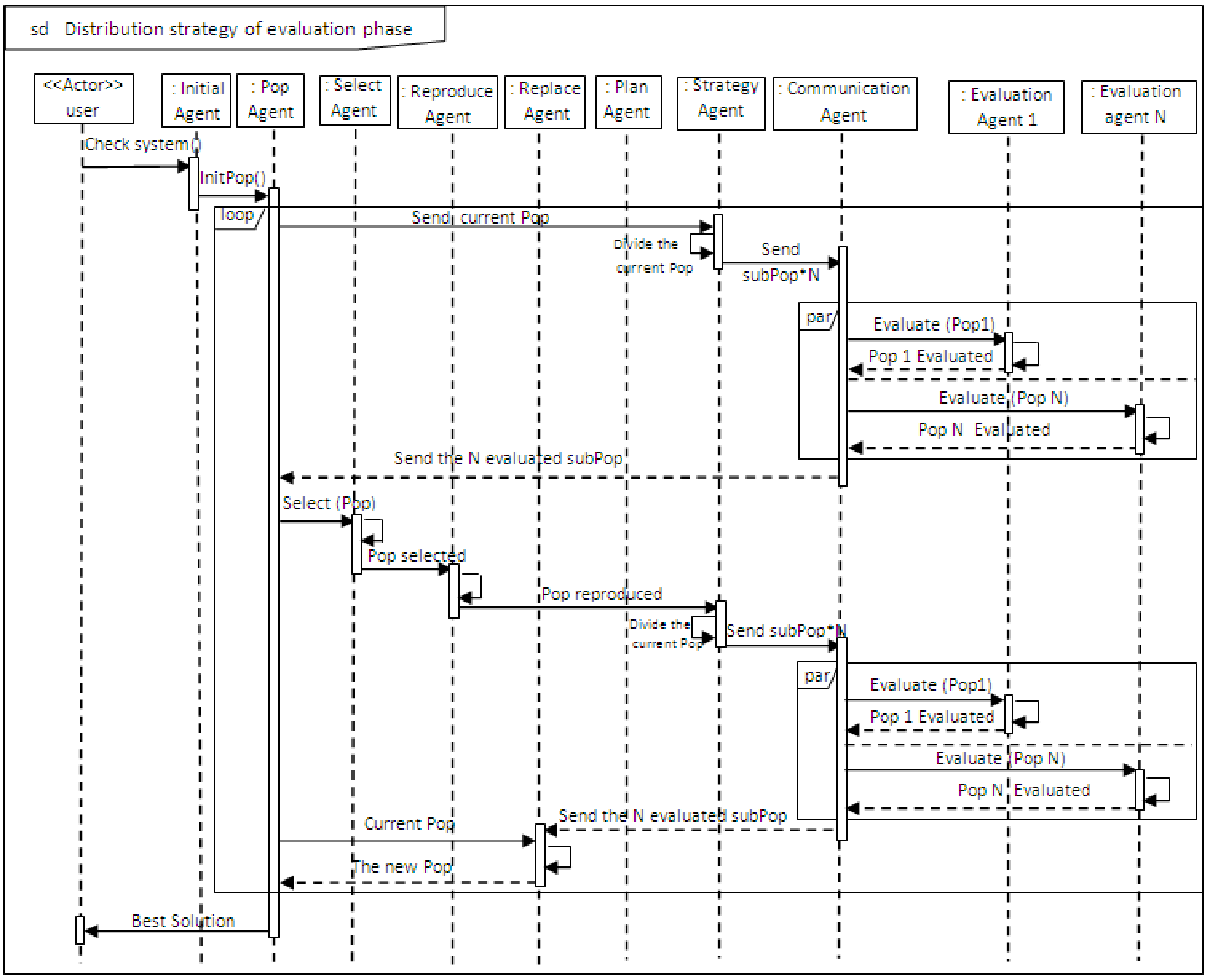


Fig. 9: The sequence diagram for modeling the distribution strategy of evaluation phase

7.2. The Implementation of Island Model

In this strategy, the Planning Agent searches the most adequate machines for the calculation in order to recover the addresses of the Population and Replacement Agents (from DF and AMS). The addresses of the Population Agents are used by DistSynIsland Agent to send the subPops to the islands. The addresses of the Replacement Agents are used by Topology Agent to define the interconnection list (with which must exchange the solutions) of each island.

When the DistSynIsland Agent receives the initialized population, the addresses of the Population Agents and the interconnection lists, it begins by dividing the population into subPops. Then, it sends to each island the necessary parameters namely: the subPop, the stopping criteria, the ID of the selection, recombination and replacement methods used to execute the evolutionary process, the ID of the selection and replacement methods used to select the emigrant solutions and replace the immigrant solutions, the interconnection list, and the time for exchanging the individuals (that is generally measured by a predefined number of iterations).

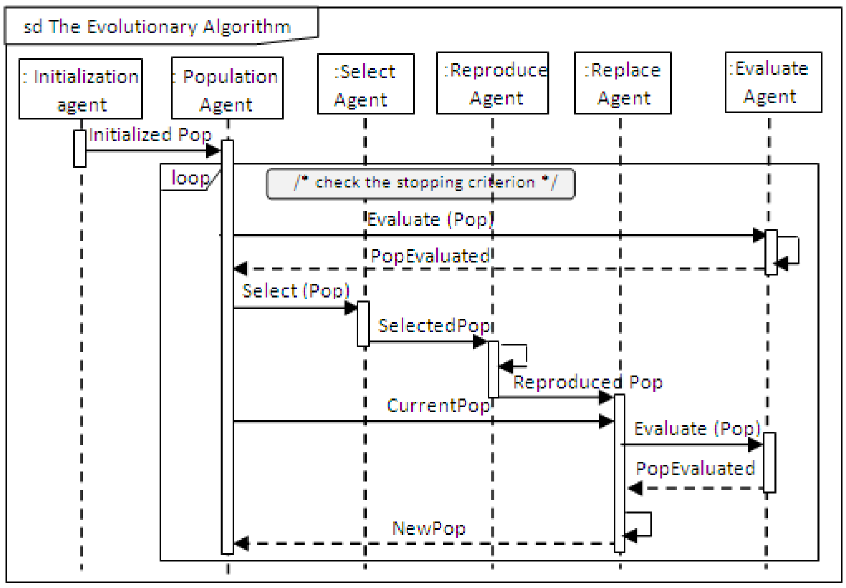


Fig. 10: The serial Evolutionary Algorithm executing in each island

At each island, the agents execute the sequential evolutionary process explained in section 6 (figure 10). After N iterations, the execution is suspended to exchange some individuals between the different islands. At the end of execution (the satisfaction of the stopping criteria) each island sends its best founded solution to the DistSynIsland Agent this latter transfers all results to the Evaluator Agent (located in the host machine) to define the best global solution.

VIII. The Experimentation Phase

After the explication of the implementation phase, this step is devoted to show some important functionalities used by agents' framework to ensure the dynamic system and to facilitate the distribution task.

```
void registrationDF() {
    //******* registration in DF *******
    DFAgentDescription dfd = new DFAgentDescription();
    dfd.setName(getAID());
    ServiceDescription sd = new ServiceDescription();
    sd.setType("selection_operator");
    sd.setName(getLocalName() + "Selection");
    dfd.addServices(sd);
    try {
        DFService.register(this, dfd);
    } catch (FIPAException fe) {
        fe.printStackTrace();
    }
    System.out.println(getLocalName()+" is Lunched ");
}
```

Fig. 11: The registration of selection agent

8.1. The Registration of Agents

Each agent connects in the system must register its services (e.g., agent ID, service name, service type, etc.) with the DF using the following code (figure 11):

```
void SearchingDF (String service)
{
    DFAgentDescription dfd = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    sd.setType( service );
    dfd.addServices(sd);
    try
    {
        DFAgentDescription[] result = DFService.search(this, dfd);
        if (result.length>0){
         for (int i = 0; i < result.length; i++) {
            System.out.println("Agent"+i+":"+ result[0].getName()) ;
    }
    }else{
     System.out.println("Search1 returns null");
    }
    }catch (FIPAException fe) {
        fe.printStackTrace();
    }

 }
```

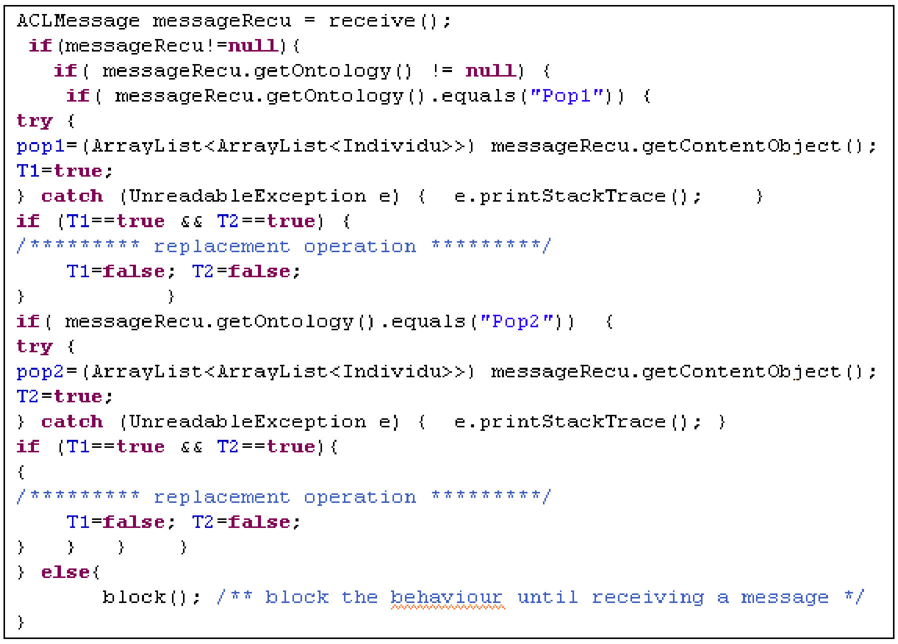
Fig. 12: Searching the DF to find the agents that provide selection service

Accordingly, the system agents may query the DF to find out what services are offered by other agents in order to provide the services they desire. The search operation returns an array which contains the ID of suitable agents. Each ID is used to extract the name and the transport address of the agent to be communicated. In fact, in our proposal this search strategy is usually used in distribution strategy by Planning Agent to find the State Agents, the Evaluator Agents, the Population Agents and the Replacement Agents.

8.2. Asynchronous Communication

The main characteristic of multi-agent systems is that the ability of agents to communicate and to interact through the exchange of messages. The sent messages are routed to the destination agent and are placed in its message queue (this latter will notified when the message arrives). The receiving agent has the ability to control its behaviours according to the type of messages (or semantic) or to the identity of senders. For example, the receiving agent may suspend all its behaviours until the arrival of the desired messages. This feature can be easily performed by using a simple instructions (blockingReceive() or block()). This strategy facilitates the implementation of asynchronous applications. It is exactly used in our framework to implement the behaviour of the agents responsible for asynchronous distribution and asynchronous replacement. Otherwise, if the agent did not use this instruction then it executes its behaviour in synchronous way, i.e. when it receives a new message it executes directly its behaviour(s).

Figure 13 shows the code used by Replacement Agent to replace the current population. In this example, the agent receives two new sub-pops evaluated by two distributed processors:



```
ACLMessage messageRecu = receive();
 if(messageRecu!=null){
   if( messageRecu.getOntology() != null) {
    if( messageRecu.getOntology().equals("Pop1")) {
try {
pop1=(ArrayList<ArrayList<Individu>>) messageRecu.getContentObject();
T1=true;
} catch (UnreadableException e) {  e.printStackTrace();    }
if (T1==true && T2==true) {
/********* replacement operation *********/
    T1=false; T2=false;
}            }
if( messageRecu.getOntology().equals("Pop2"))  {
try {
pop2=(ArrayList<ArrayList<Individu>>) messageRecu.getContentObject();
T2=true;
} catch (UnreadableException e) {  e.printStackTrace(); }
if (T1==true && T2==true){
{
/********* replacement operation *********/
    T1=false; T2=false;
}   }   }    }
} else{
       block(); /** block the behaviour until receiving a message */
}
```

Fig. 13: Asynchronous communication control

Table 1: The run time comparison between GOBA and GABA

TSP Instances	Best Time		Average Time		Worst Time		Best TSPLIB
	GOBA	GABA	GOBA	GABA	GOBA	GABA	
Wi29	0.01	0.01	0.015	0.02	0.13	0.12	27604
Eil51	1.4	1.19	1.71	1.69	2.66	2.80	426
Berlin52	0.21	0.14	0.31	0.24	0.49	0.4	7542
St70	1.26	1.36	1.92	2.02	2.74	2.73	675
Eil76	5.1	4.77	5.09	5.88	7.04	7.68	538
Pr76	1.61	1.45	3.08	3.26	4.8	4.56	108159
KroA100	3.38	2.96	5.25	4.52	7.04	5.8	21282
Lin105	3.6	3.83	5.2	5.14	7.03	5.44	14379
Pr107	5.31	3.75	6.43	5.41	8.82	6.66	44303
Bier127	11.34	8.72	18	14.53	22.52	21.22	118282
Pr144	5.01	4.76	8.81	7.22	12.51	9.44	58537
U159	15.02	15.74	22.56	22.21	30.5	28.25	42080
Ts225	65.62	58.66	78.90	70.29	95.69	80.83	126643
Pr226	27.26	26.67	48.87	46.28	66.84	69.03	80369
Pr264	39.29	35.55	60.82	51.33	79.20	64.84	49135

8.3. Performance Comparisons

Based in the above discussion (section 6), this section is devoted to compare the serial execution time of EA in agent and object implementations. For this purpose, we have implemented the genetic algorithm to solve traveling salesman problem (TSP) (proposed in [35]) using the both techniques. The two algorithms Genetic Object-Based Algorithm (GOBA) and Genetic Agent-Based Algorithm (GABA) were tested on instances of symmetric TSP from 29 to 264 cities. The execution results are displayed in table 1. The first column shows the different instances of TSP used in the tests; the second and the third and the fourth columns display the best, the average and the worst communication time (measured by seconds) estimated by GOBA and GABA to find the best solutions, these letters are obtained after 20 runs of algorithm. The last column shows the best results published in the TSPLIB [40] library.

According to the displayed results we can see that the run time obtained by GOBA and GABA is in the some cases is almost same but in the most cases is better in GABA.

In order to show agents communication under JADE framework we have implemented the parallel version GABA using a distribution strategy of reproduced and evaluation phases. Parallel GABA (PGABA) was implemented as a set of agents communicated through remote method with an ACL message. The communication between agents is shown in figure 14.

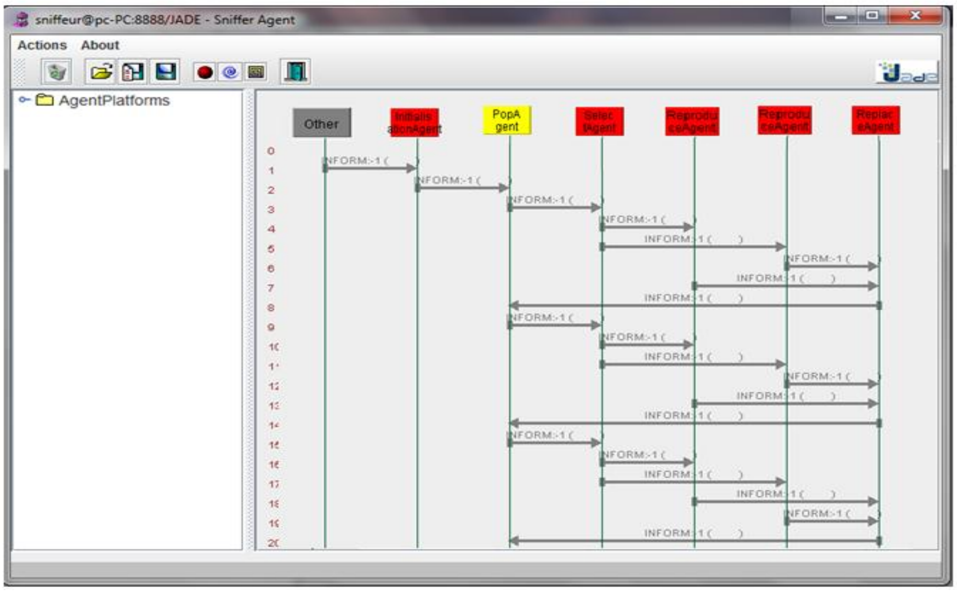


Fig. 14: Jade sniffer agent. This screenshot shows a communication between PGABA agents'

The order of the exchanged messages between the agents is the same explained in section (VII-7.1).

IX. Conclusion

Combinatorial optimization methods are in the most cases the stochastic algorithms that take a considerable time to find a reasonable solution to the NP-hard problems. Consequently, the necessity to quickly find optimal solutions in minimum runtime has led to the development of several parallelization strategies of the EC algorithms.

In the past few decades, Agent paradigm became one of the most attractive approaches that represent an exciting new means of analyzing, designing and building software systems. It is included in several areas as the artificial intelligence, the distributed computer systems and the software engineering. The agent approach has added many aspects that do not exist in the classical paradigms (Object-oriented, component-oriented) namely the autonomy, the adaptation, the semantic and the intelligent decision. In fact, to benefit from these advantages we presented in this paper a new agents-based framework architecture for developing and modeling parallel evolutionary algorithms.

For this purpose, we have begun by analyzing the principal concepts of EC framework design. Thus, we have presented some technical backgrounds used in the development. Then, we have described the overall architecture of the proposed system where we have detailed the role of each agent. Afterword, we have explained the order of the agent behaviours in serial and parallel executions.

In the illustration phase, we have demonstrated that the running time of the serial evolutionary algorithms is same in the agent and object approaches, we have also showing how each agent system inscribes in the system in order to publish its services which are requisite for other agents and finally, we have explained how the agent can easily control asynchronous and synchronous executions.

In order to accomplish the implementation of our proposal, we have divided our future work in three parts. The first one is dedicated to the implementation of the basic evolutionary operators. The second part is devoted to enhance the behaviour of each agent in order to create an intelligent and dynamic system, and the third is dedicated to evaluate the overall architecture in the computing cluster.

References

[1] Enrique Alba, Francisco Almeida, Maria J. Blesa, Carlos Cotta, M. Díaz, Isabel Dorta, Joaquim Gabarró, Coromoto León, Gabriel Luque, Jordi Petit. Efficient parallel LAN/WAN algorithms for optimization. The mallba project. Parallel Comput. Elsevier Science Publishers B. V, v32, n(5-6), 2006, pp. 415-440.

[2] Enrique Alba, Francisco Almeida, Maria J. Blesa, J. Cabeza, Carlos Cotta, M. Díaz, Isabel Dorta, Joaquim Gabarró, Coromoto León, J. Luna, Luz Marina Moreno, C. Pablos, Jordi Petit, Angélica Rojas, Fatos Xhafa. MALLBA: A Library of Skeletons for Combinatorial Optimisation (Research Note). Proceedings of the 8th International Euro-Par Conference on Parallel Processing. Springer, 2002, pp. 927–932.

[3] Bellifemine Fabio Luigi, Caire, Giovanni, Greenwood, Dominic. Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology). ISBN 0470057475, John Wiley \ Sons, 2007.

[4] Bellifemine Fabio, Caire Giovanni, Trucco Tiziana, Rimassa Giovanni, Mungenast Roland. Jade administrator's guide, 2010.

[5] Sébastien Cahon, Nordine Melab, El-Ghazali Talbi ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. Journal of Heuristics. Kluwer Academic Publishers, v10, n3, 2004, pp. 357-380.

[6] Christian Gagné, Marc Parizeau, Marc Dubreuili.Distributed JDEAL: An Environment for Parallel and Distributed Evolutionary Computations. In Proc. of the 17th Annual International Symposium on High Performance Computing Systems and Applications (HPCS), 2003, pp. 11-14.

[7] Elisabetta Cortese, Filippo Quarta, Giosue Vitaglione. Scalability and Performance of JADE Message Transport System. 2002.

[8] Gabriel Crainic, Michel Toulouse**.** Parallel Metaheuristics. Fleet Management and Logistics. T. G. C. a. G. Laporte. Kluwer Academic. 1998, pp. 205-251.

[9] Van-dat Cung, Simone L. Martins, Celso C. Ribeiro, Catherine Roucairol. Strategies for the parallel implementation of metaheuristics. Essays and Surveys in Metaheuristics. Kluwer, 2001, pp. 263-308.

[10] Fabio Bellifemine, Giovanni Caire, Agostino Poggi, Giovanni Rimassa. JADE: A software framework for developing multi-agent applications. Lessons learned. v50, n1-2, 2008, pp.10-21.

[11] Christian Gagné, Marc Parizeau. Genericity in Evolutionary Computation Software Tools: Principles and Case Study. International Journal on Artificial Intelligence Tools. v15, n(2), 2006, pp. 173-194.

[12] Christian Gagné, Marc Parizeau. Open BEAGLE: An evolutionary computation framework in C++. http://beagle.gel.ulaval.ca, 2006.

[13] Christian Gagne, Marc Parizeau, Marc Dubreuil. Distributed Beagle: An Environment for Parallel and Distributed Evolutionary Computations. Proceedings of the 17th Annual International Symposium on High Performance Computing Systems and Applications (HPCS). 2003, pp. 201-208.

[14] Gamma Erich, Helm Richard, Johnson Ralph, Vlissides John. Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc, 1995.

[15] Di Gaspero L, A. Schaerf. Easylocal++: An Object-Oriented Framework for the Design of Local Search Algorithms and Metaheuristics. In MIC'2001 4th Metaheuristics International Conference, Porto, Portugal, 2001, pp. 287-292

[16] Giovanni Caire, David Cabanillas. Application-Defined Content Languages and Ontologies, JADE documentation. 2010.

[17] David Hadka. MOEA Framework, a Java library for multiobjective evolutionary algorithm. http://www.moeaframework.org/index.html. 2009.

[18] John H. Holland. Adaptation in natural and artificial systems. , Ph.D. thesis. The University of Michigan Press. Ann Arbor, MI, USA, 1975.

[19] J. Costa, N. Lopes, P. Silva. JDEAL: The Java Distributed Evolutionary Algorithms Library. http://laseeb.isr.istutl.pt/sw/jdeal/home.html. 2000.

[20] Keijzer Maarten, Guervos, Juan J. Merelo, Romero, Gustavo, Schoenauer Marc. Evolving Objects: A General Purpose Evolutionary Computation Library. Selected Papers from the 5th European Conference on Artificial Evolution. Springer-Verlag. 2002, pp. 231–244.

[21] Kicinger Rafal, Arciszewski Tomasz, Jong Kenneth De. Evolutionary computation and structural design: A survey of the state-of-the-art.

Comput. Struct. Pergamon Press, Inc. v83, n(23-24), 2005, pp. 1943-1978.

[22] John Koza. Genetic programming. Bradford / MIT Press, 1992.

[23] John Koza. Genetic programming. Bradford / MIT Press, 1994.

[24] Kramer Michael D, Zhang Du. GAPS: A Genetic Programming System. International Journal on Artificial Intelligence Tools. v12, 2003, pp. 187-206.

[25] Natalio Krasnogor, Jim Smithr. MAFRA}: A Java Memetic Algorithms Framework. IIn Alex A. Freitas, William Hart, Natalio Krasnogor, and Jim Smith, editors, Data Mining with Evolutionary Algorithms. v8, 2000, pp. 125-131.

[26] Hyacinth S. Nwana, Divine T. Ndumu, Lyndon C. Lee. ZEUS: An Advanced Tool-Kit for Engineering Distributed Multi-Agent Systems. In Proceedings of the 3rd International Conference on Practical Applications of Intelligent Agents and Multi-Agent Technology. 1998, pp. 377-391.

[27] L.J. Fogel, A.J. Owens, M.J. Walsh, Artificial Intelligence Through Simulated Adaptation. Wiley, New-York. 1966.

[28] Meffert Klaus et al. JGAP- Java Genetic Algorithms and Genetic Programming Package. http://jgap.sf.net. 2002.

[29] M.G. Arenas, P. Collet, A.E. Eiben, M. Jelasity, J.J. Merelo, B. Paechter, M. Preuß, and M. Schoenaue. A framework for distributed evolutionary algorithms. In Proceedings of PPSN VII. 2002.

[30] Pilat Marcin L, White Tony. Using Genetic Algorithms to Optimize ACS-TSP. ANTS'02: Proceedings of the Third International Workshop on Ant Algorithms. Springer-Verlag, 2002, pp. 282-287.

[31] Bill Punch, Douglas Zongker. lil-gp 1.1 beta. http://garage.cse.msu.edu/software/lil-gp. 1998.

[32] Rechenberg, I. Evolutions strategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. TU Berlin, 1971.

[33] Rimassa G, Bellifemine F, Poggi A. JADE -A FIPA Compliant Agent Framework. Proceedings of the Practical Applications of Intelligent Agent, 1999, pp. 97-108.

[34] Sara Sabba, Salim Chikhi. Integrating the Best 2-Opt method to enhance the genetic algorithm execution time in solving the traveler salesman problem. Complex System and Dependability. Advances in Intelligent and Soft Computing. Springer-Verlag, v170, 2011, pp. 195-208.

[35] Schwefel Hans-Paul. Numerical Optimization of Computer Models. John Wiley \ Sons, Inc. 1981.

[36] Sean Luke, Liviu Panait, Gabriel Balan, Sean Paus, Zbigniew Skolicki, Jeff Bassett, obert Hubley, Alexander Chircop. ECJ: A java-based evolutionary computation and genetic programming research system, http://cs.gmu.edu/˜eclab/projects/ecj, 2005.

[37] Sara Silva. GPLAB: A genetic programming toolbox for MATLAB. http://gplab.sourceforge.net. 2005.

[38] Sara Silva, Jonas Almeida. GPLAB: A genetic programming toolbox for MATLAB. In Proc. of the Nordic MATLAB Conference, 2003, pp. 273–278.

[39] TSPLIB(1995).Http://www.iwr.uni-heidelberg.de/groups/ comopt/software/TSPLIB95/STSP.html.

[40] AgentBuilder U.G. An Integrated Toolkit for Constructing Intelligent Software Agents, 2000.

[41] Ventura Sebastian, Romero Cristobal, Zafra Amelia, Delgado Jose A, Hervas Cesar. JCLEC: a Java framework for evolutionary computation. Soft Comput.Springer, 2007, v11, n4, pp. 381–392.

[42] Matthew Wall. GAlib: A C++ library of genetic algorithm components, http://lancet.mit.edu/ga, 2000.

[43] Michael J. Wooldridge, Nicholas R. Jennings. Software engineering with agents: pitfalls and pratfalls. IEEE Internet Computing, v3, n3, 1999, pp. 20-27.

[44] Yoav Shoham, Artificial Intelligence, v60, n1, 1993, pp. 51-92.

Authors' Profiles

Sara Sabba received her Master's degree from Mentouri University of Constantine, Algeria in 2009 and she is a PhD student from the year 2010 in Mentouri University of Constantine, Algeria. Her current research interests include optimization methods and their applications to solve several combinatorial optimization problems.

Salim Chikhi has done his PhD in Computer Science from the University of Constantine, in 2005. He is currently a Professor at the Mentouri University, Constantine, Algeria. He is the Leader of the SCAL team of MISC Laboratory. His research areas include soft computing and artificial life techniques and their application in several domains.