Parallel Cooperative Optimization Research Group

Particle Swarm Optimization with ParadisEO

http://paradiseo.gforge.inria.fr

Laboratoire d'Informatique Fondamentale de Lille

ParadisEO (1)

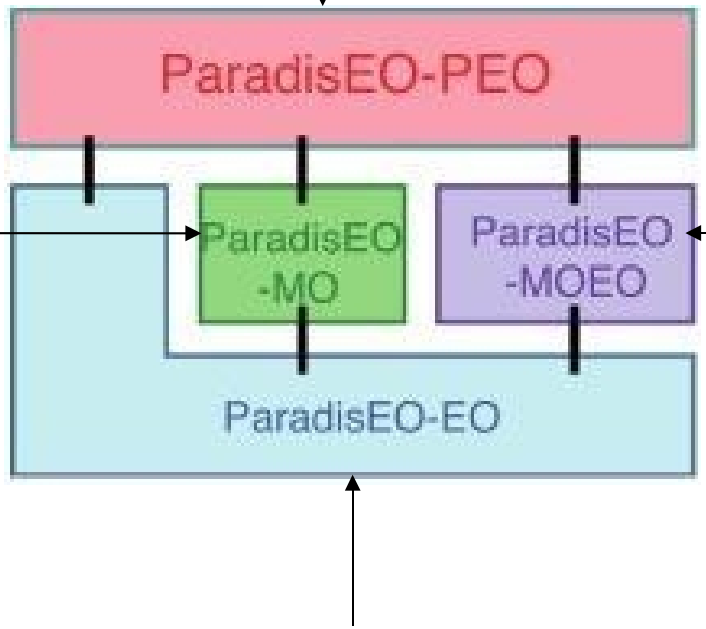
- A templates-based, ANSI-C++ compliant Metaheuristic Computation Framework
- GForge Project by INRIA Dolphin Team
- Paradigm Free (genetic algorithms, genetic programming, particle swarm optimization, local searches …)
- Hybrid, distributed and cooperative models

ParadisEO (2)

- Flexible / a considered problem
- Generic components (variation operators, selection, replacement, termination, particle behaviors …)
- Many services (visualization, managing command-line parameters, saving/restarting, …)

ParadisEO : Module-based architecture

Parallel and distributed metaheuristics



Tabu Search, Simulated Annealing, Hill-Climbing: single solution based metaheuristics

Multi-objective metaheuristics : NSGAII…, metrics…

Evolutionary computation, Swarm intelligence : population-based metaheuristics

The main steps to build a particle swarm optimization algorithm

1. Design a representation
2. Decide how to initialize a population (=swarm)
3. Design a way of evaluating an individual
4. Design suitable velocity operator
5. Decide the flight operator
6. Decide how to manage the population
7. Decide the “best updating” strategy
8. Decide the continuation criterion

Framework and tutorial application

- Framework dedicated to metaheuristics

  → Paradiseo *Parallel and Distributed Evolving Objects*

- Tutorial application

  → Norm optimization problem

  (Euclidean norm minimization)

Application to the Norm Optimization Problem (NOP)

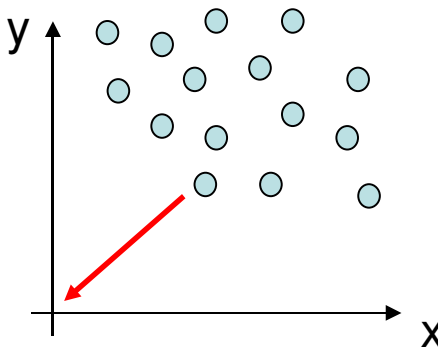
- $f : R^n \rightarrow R$ $\qquad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, x_i \in R$

$$f(x_1, x_2, \ldots, x_n) = \sqrt{\sum_{i=0}^{N} x_i^2}$$

- Example :

$$f(x_1, x_2) = \sqrt{x_1^2 + x_2^2}$$

Minimization of the Euclidean norm



Designing a representation

- Representing an individual as a position
- Maybe several ways to do this. The representation must be relevant regards the tackled problem
- When choosing a representation, we have to bear in mind how the positions will be evaluated and how the flight operators will be used

Real-valued representation

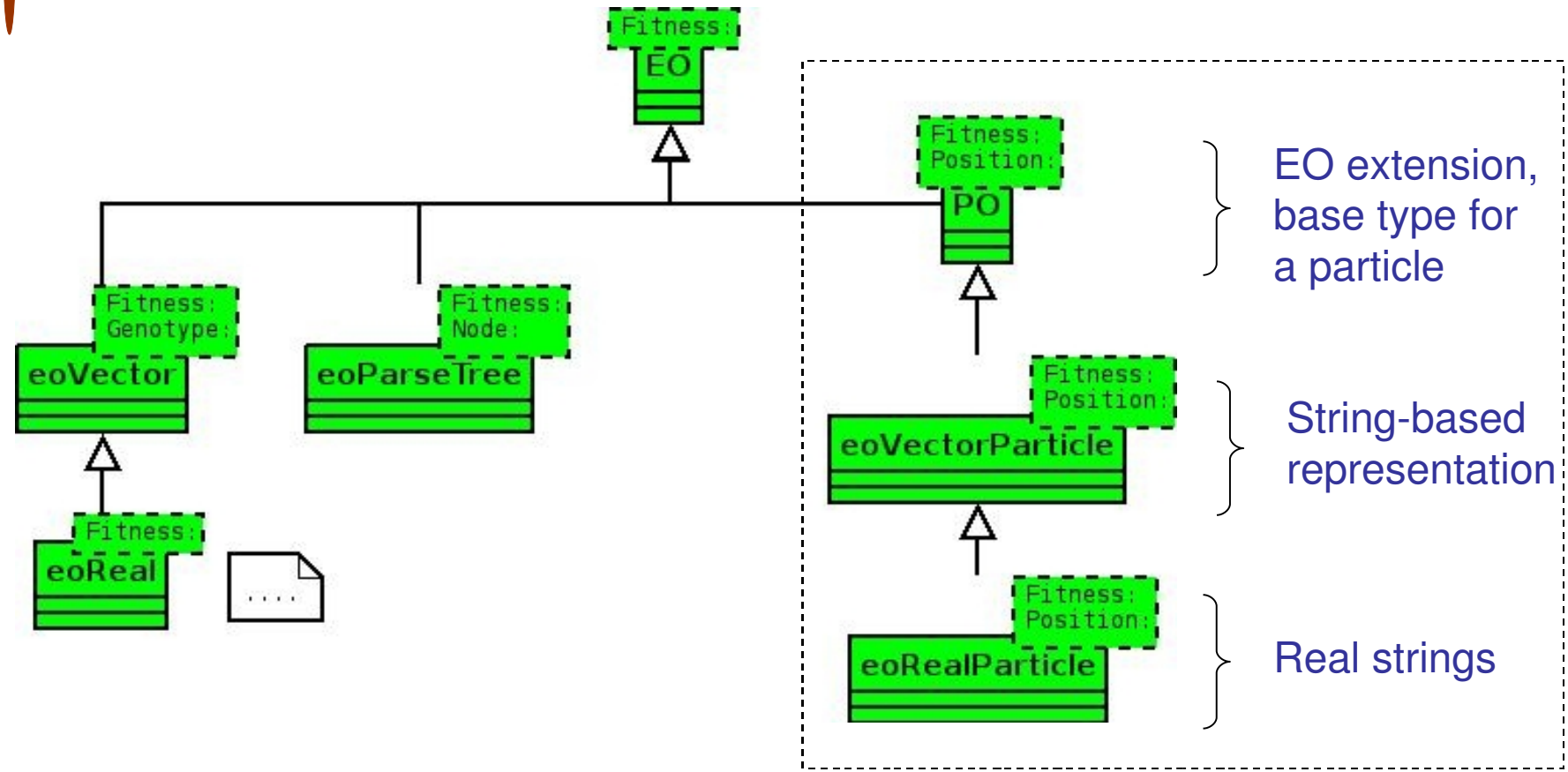
- Individuals are represented as a tuple of n real-valued numbers

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, x_i \in R$$

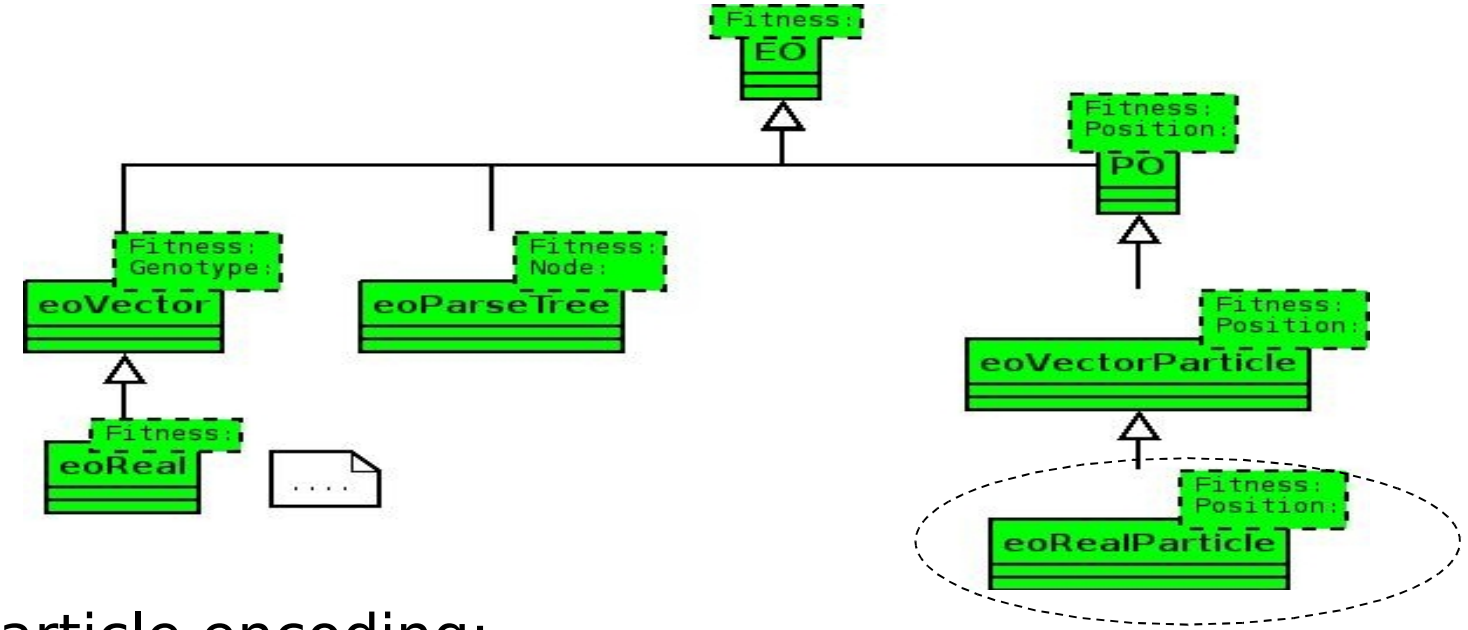
- The fitness function maps tuples of real numbers to a single real number

$$f : R^n \rightarrow R$$

Existent basic particle representations in ParadisEO



Application to the NOP

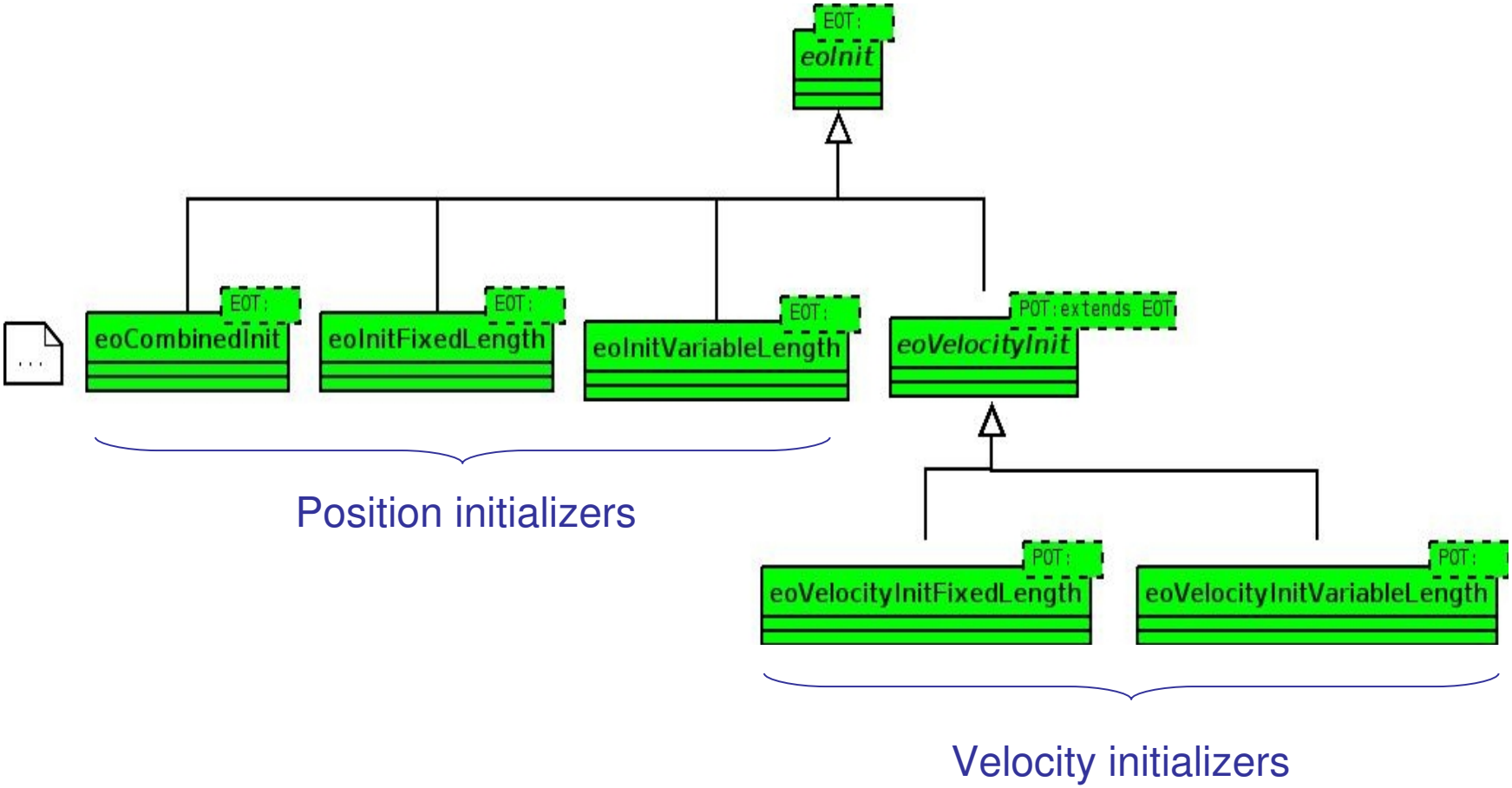


- Particle encoding:
  - Each component of the position is a real
  - Each component of the best position is a real
  - Each component of the velocity is a real

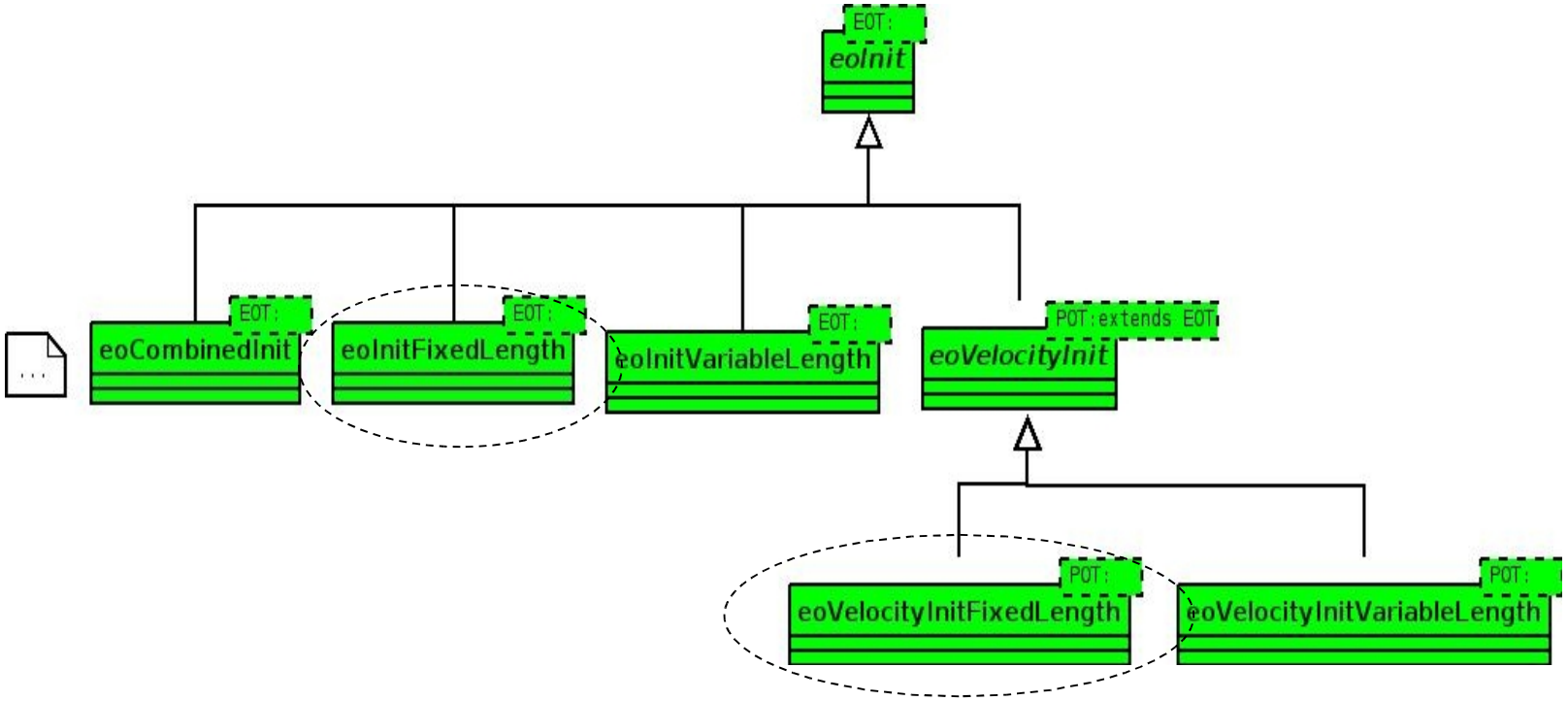
Initialization of the swarm

- Initialize
  - Positions Standard
  - Velocities
  - Best positions of each particle
  - The global best
- Standard strategies
  - Position + Velocities: Random
  - Initial global best = initial best particle

Existent swarm initializers in ParadisEO

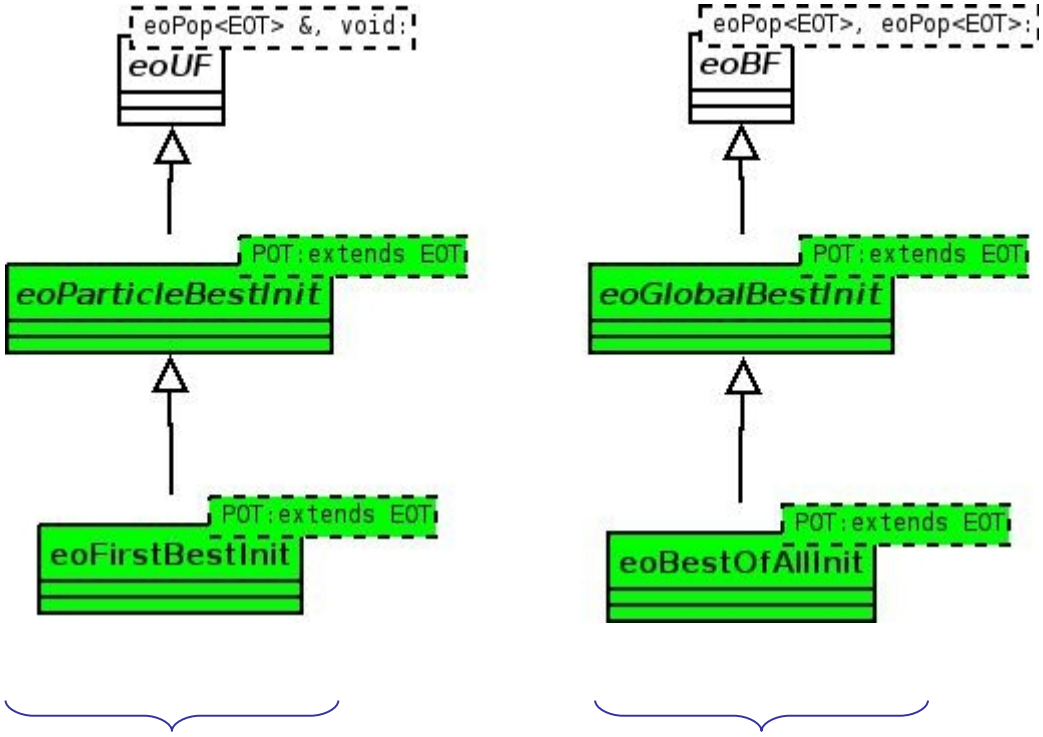


Application to the NOP



- Generate the initial positions and velocities at random between bounds

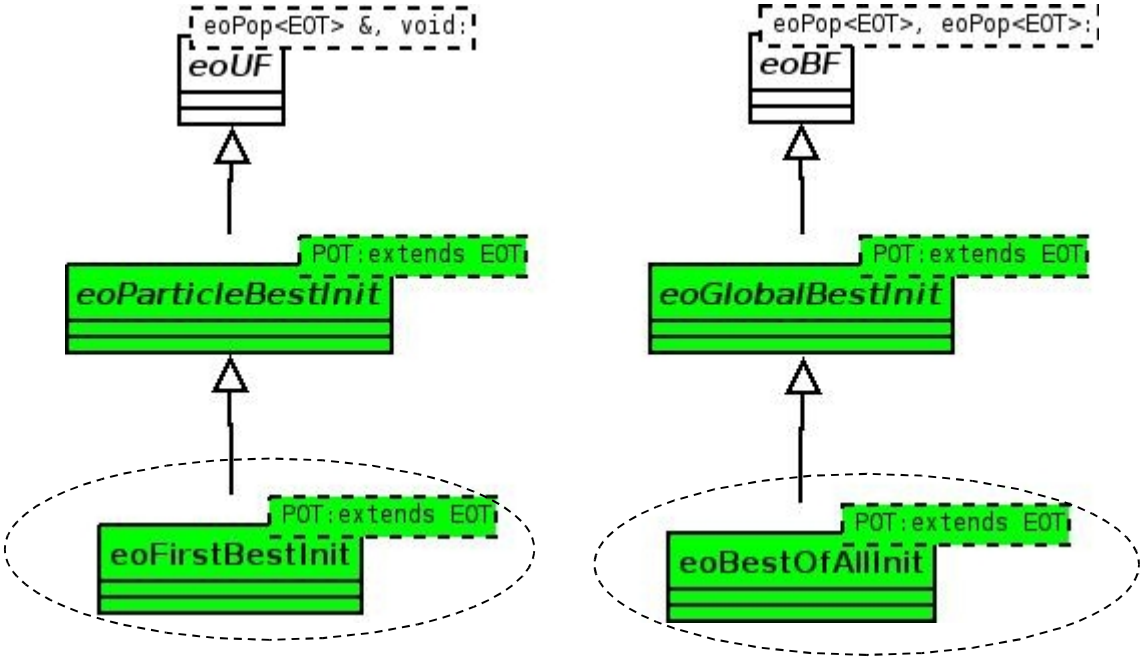
Existent swarm initializers in ParadisEO (2)



Particle best initializers

Global best initializers

Application to the NOP



- The first position is the initial best
- The initial global best is the initial best of the swarm

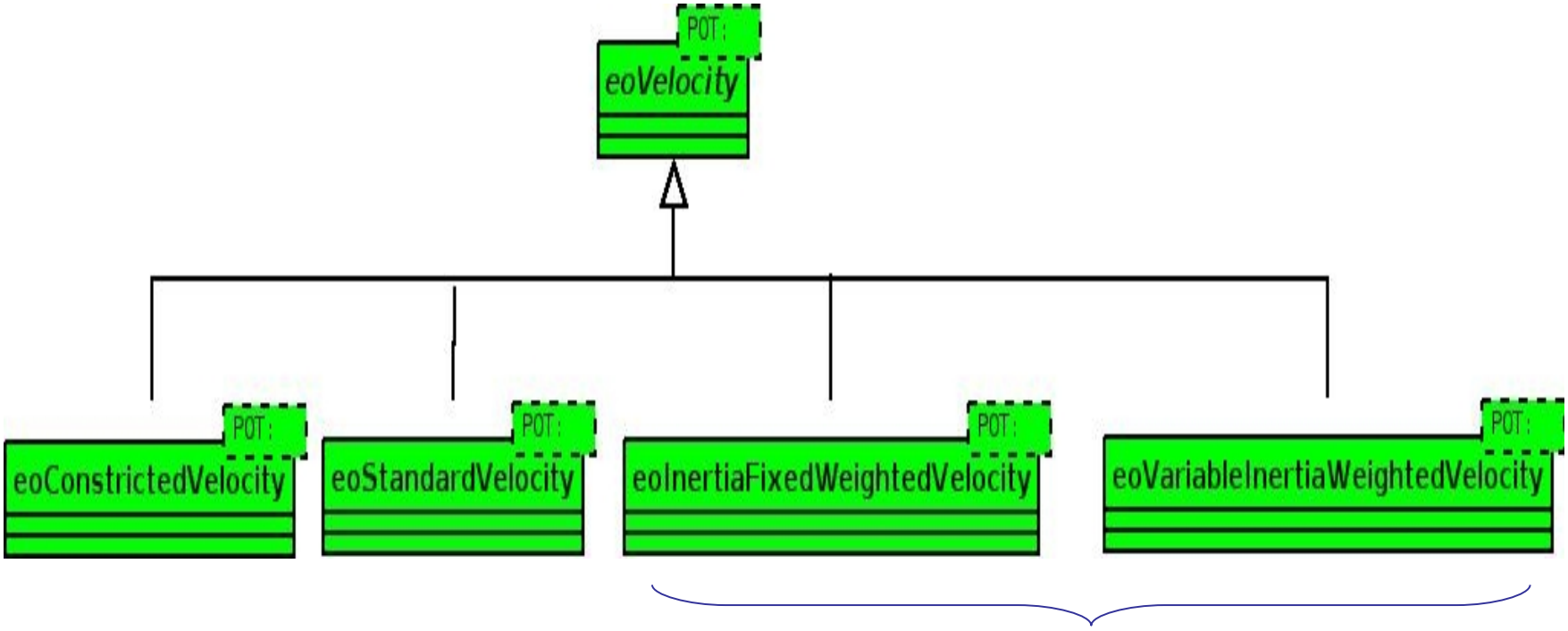
The evaluation of an individual

- This is by far the most costly step for real applications
- It might be a subroutine, a black-box simulator, or any external process (e.g. robot experiment)
- Fitness could be approximated

Designing a velocity

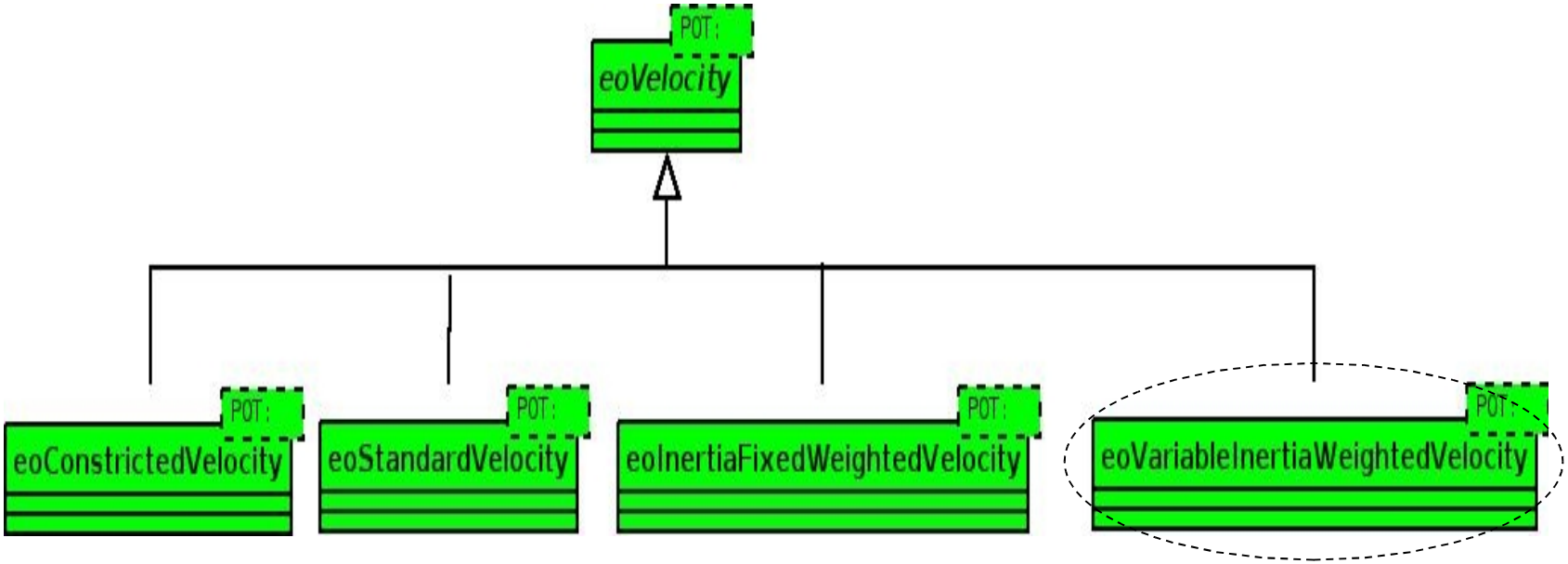
- The velocity gives the direction
- Learning factors and memory
  - Follow the global best ?
  - Follow the particle's best ?
- Global version faster
  - May converge to local optimums
- Local version slower
  - Not easy to be trapped into local optimums

Existent velocities in ParadisEO



Inertia weight based velocities

Application to the NOP

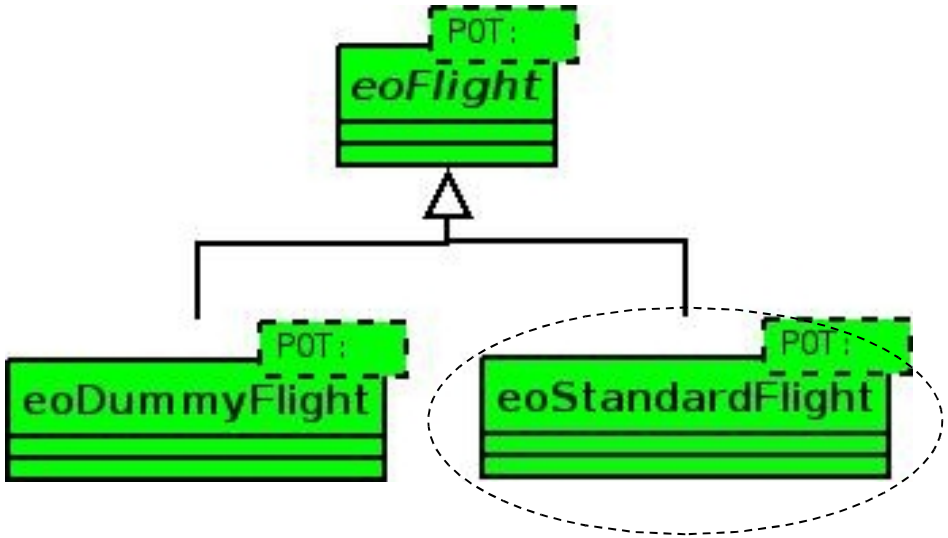


- Inertia weight factor which decreases with the number of generations

Vi = K * [Vi + c1 x (Pi best – Pi) + c2 x (Pgbest – Pi)]

Flight operator

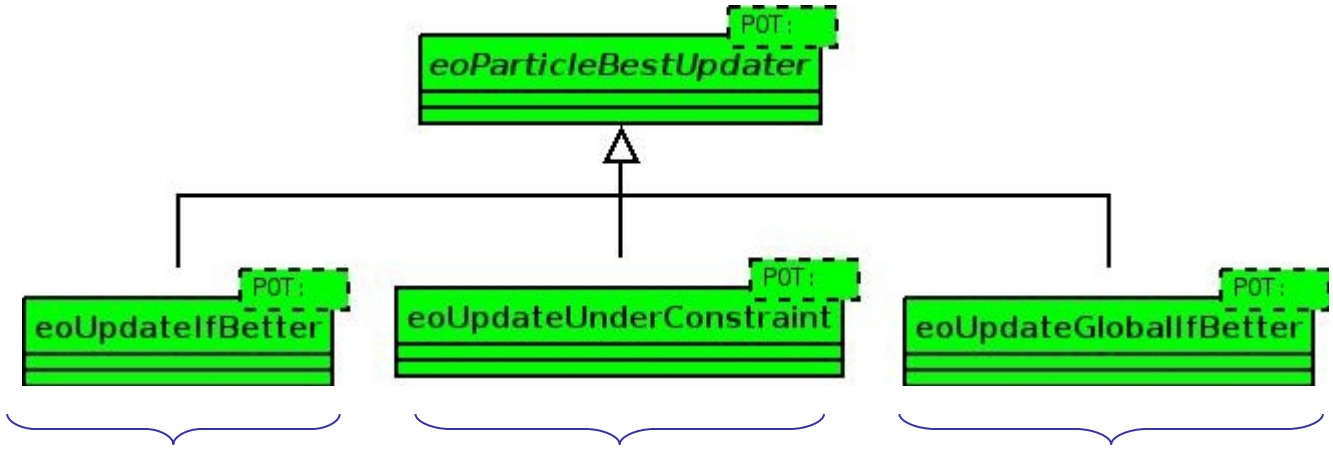
- The flight consists in adding the velocity to the current position (standard)
- ParadisEO provides this standard approach but is opened



Designing a “best updating” strategy

- The global/local directions depends on the updating strategy
- Standard approach: update the best position of a particle if the new fitness value is better
- Constraint handling - what if the solution breaks some constraint of the problem
  - penalize the fitness
  - specific methods
- Multi-objective optimization gives a set of compromise solutions

Existent updating strategies in ParadisEO

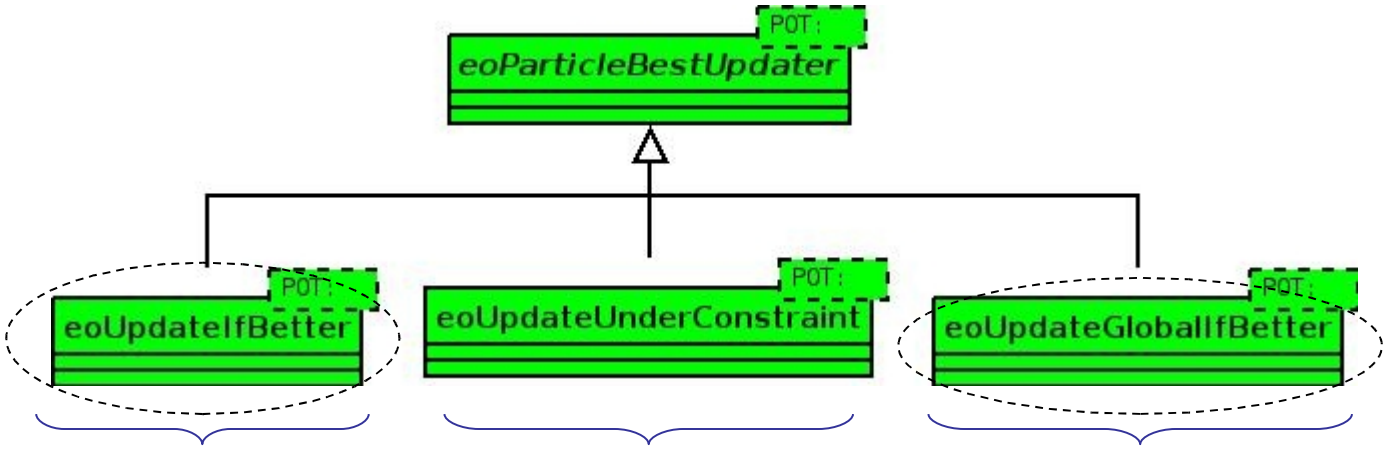


Standard particle's best update

Update the fitness if it's ε-better

Standard global best update

Application to the NOP



Standard particle's best update

Update the fitness if it's ε-better

Standard global best update

The continuation strategy

- The optimum is reached !
- Limit on CPU resources: Maximum number of fitness evaluations
- A given number of generations without improvement: for the swarm or for the global best

Core classes

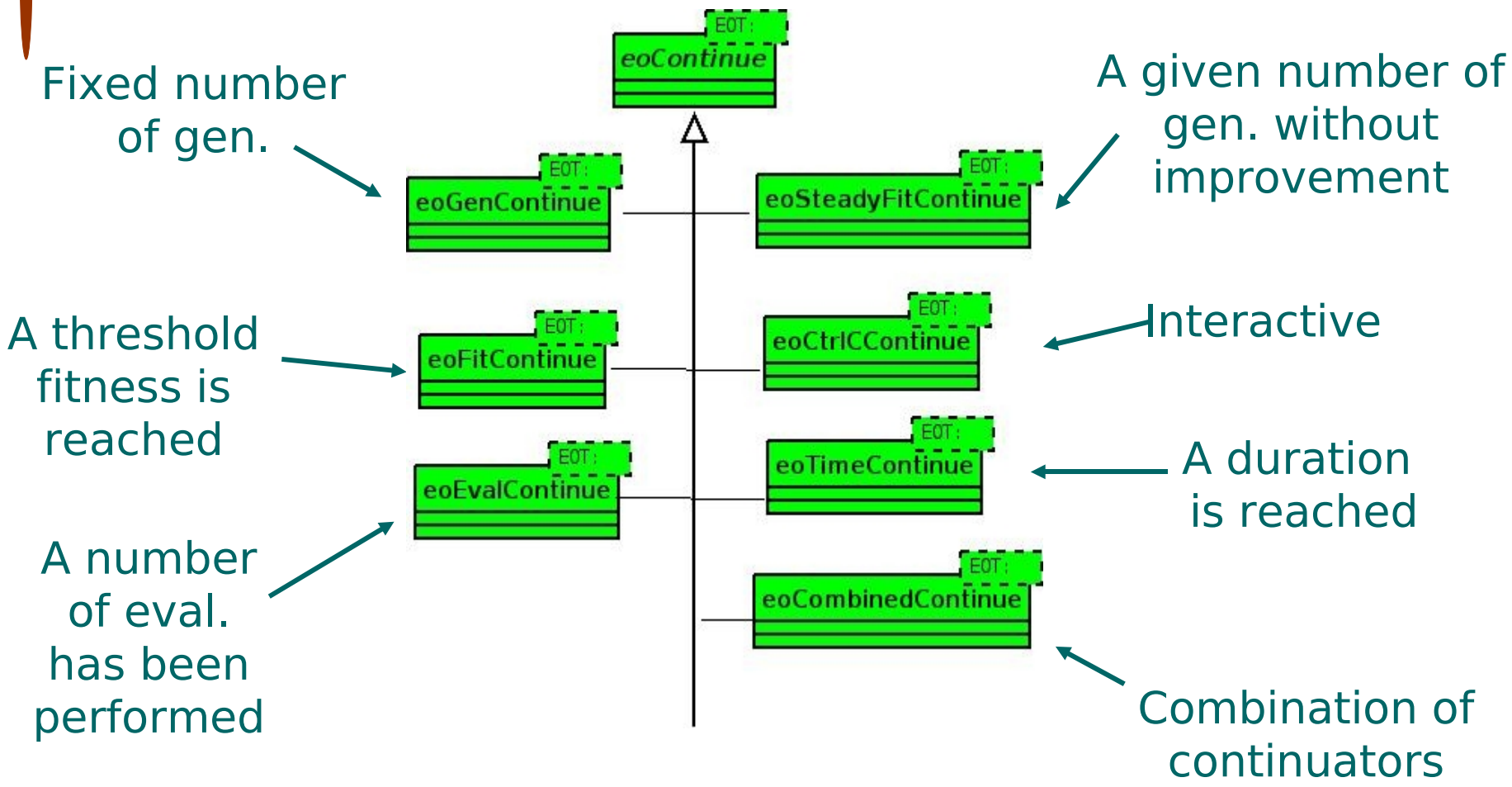
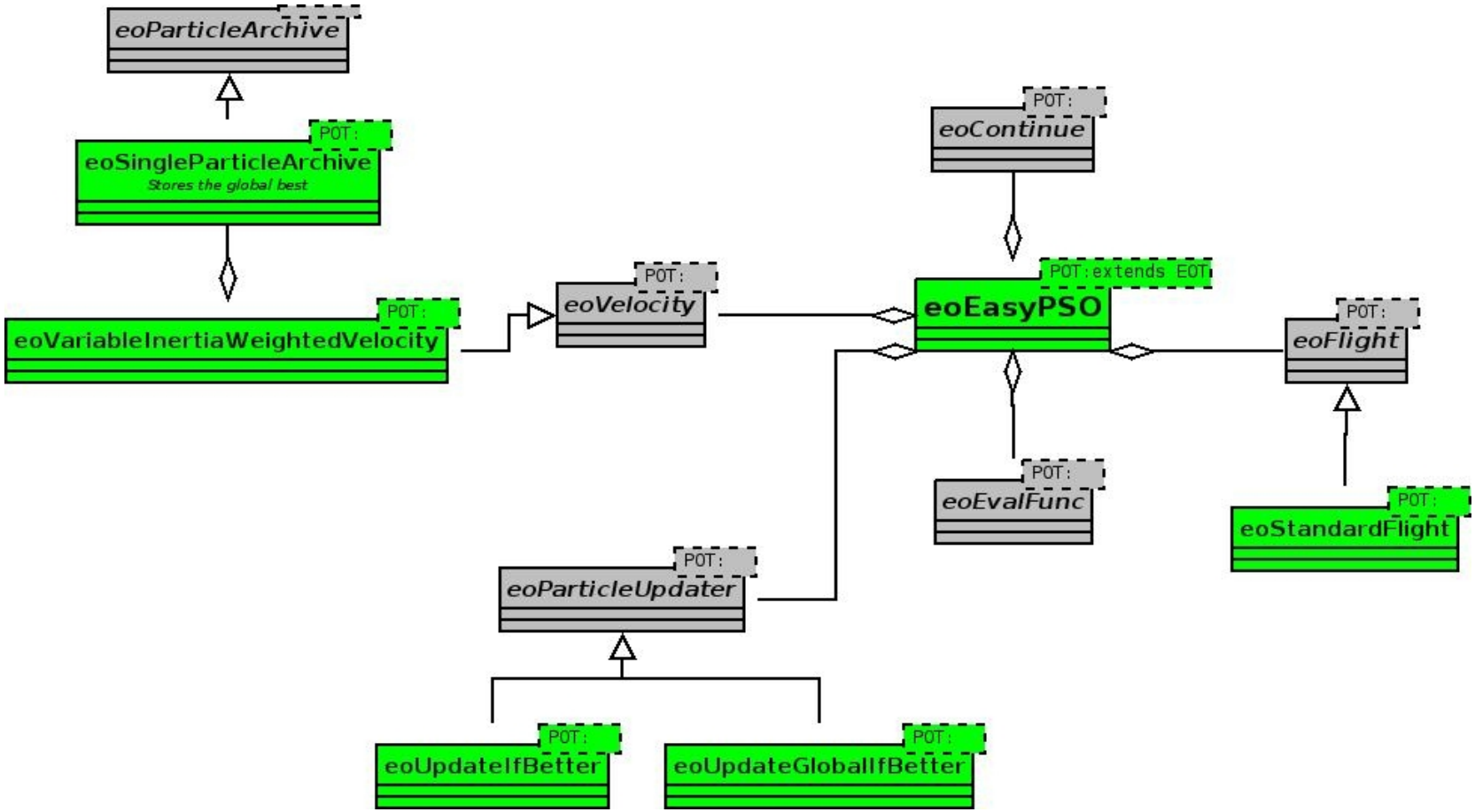


Illustration. Core classes of the Particle Swarm Algorithm



Implementation

- eoPop < Indi > pop; /* population */
- eoEvalFuncPtr<Indi, double, const Indi& > eval(real_value); /* evaluator */

- eoUniformGenerator < double >uGen (-1.0, 1.0);
- eoInitFixedLength < Indi > random (VEC_SIZE, uGen);
- eoVelocityInitFixedLength < Indi > speedRandom (VEC_SIZE, uGen);
- eoFirstIsBestInit < Indi > localInit;
- eoSingleParticleArchive < Indi > archive;
- eoBestOfAllInit < Indi > globalInit (archive); /* initializers */

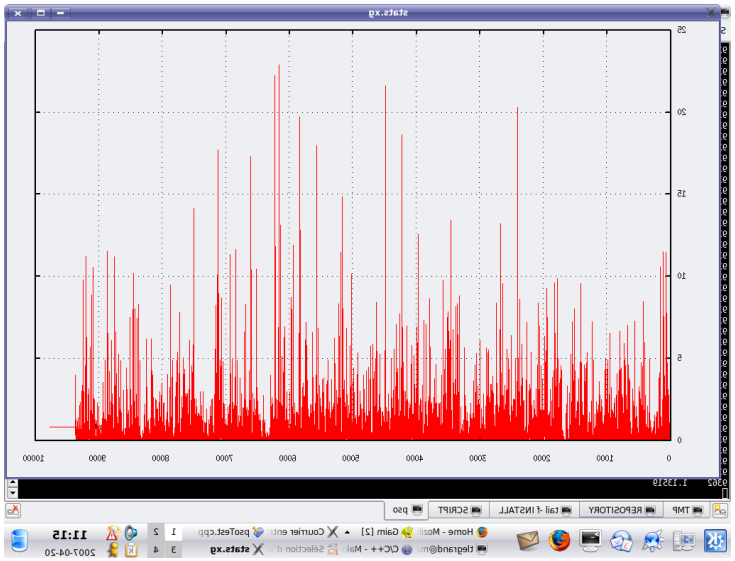
- eoVariableInertiaWeightedVelocity < Indi > velocity (archive, generationCounter,MAX_GEN, C1, C2);
- eoStandardFlight < Indi > flight; /* main operators */

- eoUpdateIfBetter < Indi > updater;
- eoUpdateGlobalIfBetter < Indi > globalUpdater (archive); /* updaters */

- eoGenContinue < Indi > genCont (MAX_GEN);
- **eoEasyPSO < Indi > psa (genCont , eval, velocity, flight, updater, globalUpdater); /* PSO */**
- **psa (pop); /* run it */**

Execution and visualization

- Advanced runtime control
  - Termination condition on the swarm
  - Termination condition on the global best
- Advanced statistics and utilities



Download, test and enjoy

- http://paradiseo.gforge.inria.fr
- Free download of **paradiseo-ix86-1.0-alpha**
- Future work (beta and major releases)
  - Parallel and distributed PSO
  - Advanced features