Several frameworks for metaheuristics have been proposed in the literature. Most of them have the following limitations:

- **Metaheuristics:** most of the exiting frameworks focus only on a given metaheuristic or family of metaheuristics such as evolutionary algorithms (e.g., GAlib [809]), local search (e.g., EasyLocal++ [301], Localizer [550]), and scatter search (e.g., OPTQUEST). Only few frameworks are dedicated on the design of both families of metaheuristics. Indeed, a unified view of metaheuristics must be done to provide a generic framework.
- **Optimization problems:** most of the software frameworks are too narrow, that is, they have been designed for a given family of optimization problems: nonlinear continuous optimization (e.g., GenocopIII), combinatorial optimization (e.g., iOpt), monoobjective optimization (e.g., BEAGLE), multiobjective optimization (e.g., PISA [81]), and so on.
- **Parallel and hybrid metaheuristics:** Moreover, most of the existing frameworks either do not provide hybrid and parallel metaheuristics at all (Hotframe [262]) or supply just some parallel models: island model for evolutionary algorithms (e.g., DREAM [35], ECJ [819], JDEAL, distributed BEAGLE [291]), independent multistart and parallel evaluation of the neighborhood (e.g., TS [79]), or hybrid metaheuristics (iOpt [806]).
- **Architectures:** Finally, seldom a framework is found that can target many types of architectures: sequential and different types of parallel and distributed architectures, such as shared-memory (e.g., multicore, SMP), distributed-memory (e.g., clusters, network of workstations), and large-scale distributed architectures (e.g., desktop grids and high-performance grids). Some software frameworks are dedicated to a given type of parallel architectures (e.g., MALLBA [22], MAFRA [481], and TEMPLAR [426,427]).

Table 1.8 illustrates the characteristics of the main software frameworks for metaheuristics[52]. For a more detailed review of some software frameworks and libraries for metaheuristics, the reader may refer to Ref. [804].

1.8.3 ParadisEO Framework

In this book, we will use the ParadisEO[53] framework to illustrate the design and implementation of the different metaheuristics. The ParadisEO platform honors the criteria mentioned before, and it can be used both by no-specialists and by optimization method experts. It allows the design and implementation of

- Single-solution based and population-based metaheuristics in a unifying way (see Chapters 2 and 3).

[52] We do not claim an exhaustive comparison.
[53] ParadisEO is distributed under the CeCill license.

TABLE 1.8 Main Characteristics of Some Software Frameworks for Metaheuristics

Framework or Library	Metaheuristic	Optimization Problems	Parallel Models	Communication Systems
EasyLocal++	S-meta	Mono	-	-
Localizer++	S-meta	Mono	-	-
PISA	EA	Multi	-	-
MAFRA	LS, EA	Mono	-	-
iOpt	S-meta, GA, CP	Mono, COP	-	-
OptQuest	SS	Mono	-	-
GAlib	GA	Mono	Algo-level Ite-level	PVM
GenocopIII	EA	Mono, Cont	-	-
DREAM	EA	Mono	Algo-level	Peer-to-peer sockets
MALLBA	LS EA	Mono	Algo-level Ite-level	MPI Netstream
Hotframe	S-meta, EA	Mono	-	-
TEMPLAR	LS, SA, GA	Mono, COP	Algo-level	MPI, threads
JDEAL	GA, ES	Mono	Ite-level	Sockets
ECJ	EA	Mono	Algo-level	Threads, sockets
Dist. BEAGLE	EA	Mono	Algo-level Ite-level	Sockets
ParadisEO	S-meta P-meta	Mono, Multi COP, Cont	Algo-level Ite-level Sol-level	MPI, threads Condor Globus

[S-meta: S-metaheuristics; P-meta: P-metaheuristics; COP: combinatorial optimization; Cont: continuous optimization; Mono: Monoobjective optimization; Multi: multiobjective optimization, LS: local search; ES: evolution strategy; SS: scatter search; EA: evolutionary algorithms; GA: genetic algorithms; Algo-level: algorithmic level of parallel model; Ite-level: iteration level of parallel models; Sol-level: solution level of parallel models. Unfortunately, only a few of them are maintained and used!.]

- Metaheuristics for monoobjective and multiobjective optimization problems (see Chapter 4).
- Metaheuristics for continuous and discrete optimization problems.
- Hybrid metaheuristics (see Chapter 5).
- Parallel and distributed metaheuristics (see Chapter 6).

ParadisEO is a white box object-oriented framework based on a clear conceptual separation of the metaheuristics from the problems they are intended to solve. This separation and the large variety of implemented optimization features allow a maximum code and design reuse. The separation is expressed at implementation level by splitting the classes into two categories: provided classes and required classes. The provided classes constitute a hierarchy of classes implementing the invariant part of the code. Expert users can extend the framework by inheritance/specialization. The

required classes coding the problem-specific part are abstract classes that have to be specialized and implemented by the user.

The classes of the framework are fine-grained and instantiated as evolving objects embodying one and only one method. This is a particular design choice adopted in ParadisEO. The heavy use of these small-size classes allows more independence and thus a higher flexibility compared to other frameworks. Changing existing components and adding new ones can be easily done without impacting the rest of the application. Flexibility is enabled through the use of the object-oriented technology. Templates are used to model the metaheuristic features: coding structures, transformation operators, stopping criteria, and so on. These templates can be instantiated by the user according to his/her problem-dependent parameters. The object-oriented mechanisms such as inheritance, polymorphism, and so on are powerful ways to design new algorithms or evolve existing ones. Furthermore, ParadisEO integrates several services making it easier to use, including visualization facilities, online definition of parameters, application checkpointing, and so on.

ParadisEO is one of the rare frameworks that provides the most common parallel and distributed models. These models concern the three main parallel models: algorithmic level, iteration level, and solution level. They are portable on different types of architectures: distributed-memory machines and shared-memory multiprocessors as they are implemented using standard libraries such as message passing interface (MPI), multithreading (Pthreads), or grid middlewares (Condor or Globus). The models can be exploited in a transparent way, one has just to instantiate their associated ParadisEO components. The user has the possibility to choose by a simple instantiation for the communication layer. The models have been validated on academic and industrial problems. The experimental results demonstrate their efficiency. The experimentation also demonstrates the high reuse capabilities as the results show that the user redo little code. Furthermore, the framework provides the most common hybridization mechanisms. They can be exploited in a natural way to make cooperating metaheuristics belonging either to the same family or to different families.

ParadisEO is a C++ LGPL open-source framework (STL-Template)[54]. It is portable on Windows, Unix-like systems such as Linux and MacOS. It includes the following set of modules (Fig. 1.31):

- **Evolving objects (EO):** The EO library was developed initially for evolutionary algorithms (genetic algorithms, evolution strategies, evolutionary programming, genetic programming, and estimation distribution algorithms) [453]. It has been extended to population-based metaheuristics such as particle swarm optimization and ant colony[55] optimization.
- **Moving objects (MO):** It includes single-solution based metaheuristics such as local search, simulated annealing, tabu search, and iterated local search.

[54]Downloadable at `http://paradiseo.gforge.inria.fr`.
[55]The model implemented is inspired by the self-organization of *Pachycondyla apicalis* ant species.

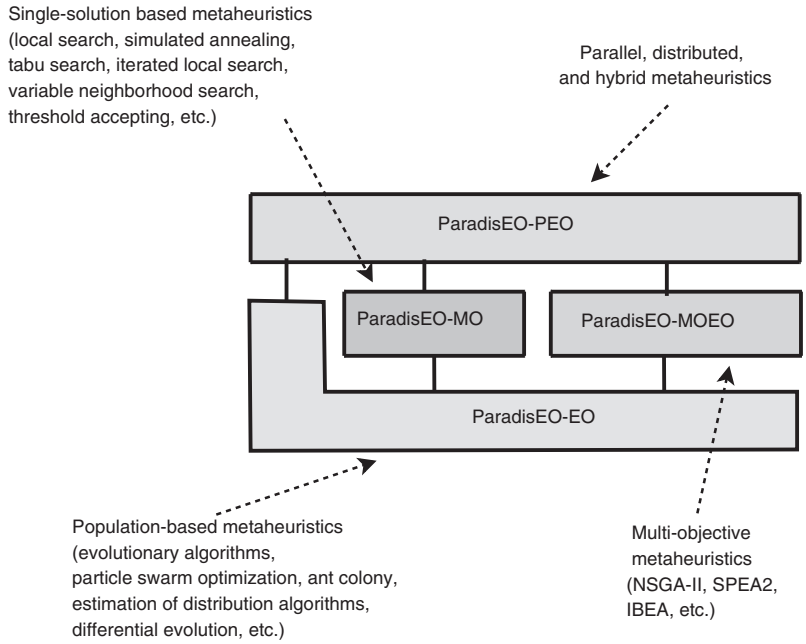

FIGURE 1.31 The different unified modules of the ParadisEO framework.

- **Multiobjective evolving objects (MOEO):** It includes the search mechanisms to solve multiobjective optimization problems such as fitness assignment, diversification, and elitism. From this set of mechanisms, classical algorithms such as NSGA-II, SPEA2, and IBEA have been implemented and are available.
- **Parallel evolving objects (PEO):** It includes the well-known parallel and distributed models for metaheuristics and their hybridization.

1.8.3.1 ParadisEO Architecture The architecture of ParadisEO is multilayered and modular allowing to achieve the objectives quoted above (Fig. 1.32).

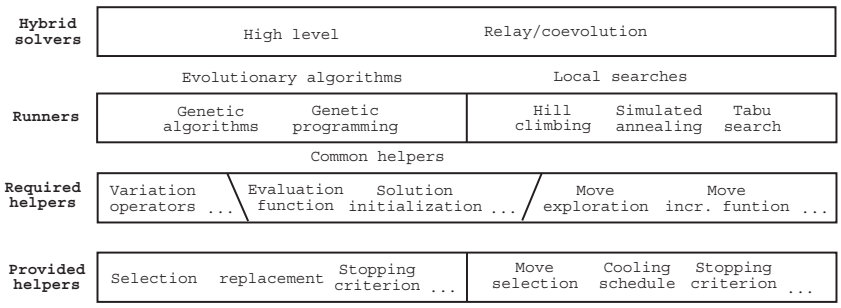

FIGURE 1.32 Architecture of the ParadisEO framework.

This allows particularly a high genericity, flexibility, and adaptability, an easy hybridization, and code and design reuse. The architecture has three layers identifying three major classes: *Solvers*, *Runners*, and *Helpers*.

- **Helpers:** Helpers are low-level classes that perform specific actions related to the search process. They are split into two categories: *population helpers* (PH) and *single-solution helpers* (SH). Population helpers include mainly the transformation, selection, and replacement operations, the evaluation function, and the stopping criterion. Solution helpers can be generic such as the neighborhood explorer class, or specific to the local search metaheuristic such as the tabu list manager class in the tabu search solution method. On the other hand, there are some special helpers dedicated to the management of parallel and distributed models, such as the communicators that embody the communication services. Helpers cooperate between them and interact with the components of the upper layer, that is, the runners. The runners invoke the helpers through function parameters. Indeed, helpers do not have not their own data, but they work on the internal data of the runners.
- **Runners:** The *Runners* layer contains a set of classes that implement the metaheuristics themselves. They perform the run of the metaheuristics from the initial state or population to the final one. One can distinguish the *population runners* (PR) such as genetic algorithms, evolution strategies, particle swarm, and so on and *single-solution runners* (SR) such as tabu search, simulated annealing, and hill climbing. Runners invoke the helpers to perform specific actions on their data. For instance, a PR may ask the fitness function evaluation helper to evaluate its population. An SR asks the movement helper to perform a given movement on the current state. Furthermore, runners can be serial or parallel distributed.
- **Solvers:** Solvers are devoted to control the search. They generate the initial state (solution or population) and define the strategy for combining and sequencing different metaheuristics. Two types of solvers can be distinguished: *single metaheuristic solvers* (SMS) and *multiple-metaheuristic solvers* (MMS). SMS are dedicated to the execution of a single metaheuristic. MMS are more complex as they control and sequence several metaheuristics that can be heterogeneous. They use different hybridization mechanisms. Solvers interact with the user by getting the input data and by delivering the output (best solution, statistics, etc.).

According to the generality of their embedded features, the classes of the architecture are split into two major categories: *provided* classes and *required* classes. Provided classes embody the factored out part of the metaheuristics. They are generic, implemented in the framework, and ensure the control at run time. Required classes are those that must be supplied by the user. They encapsulate the problem-specific aspects of the application. These classes are fixed but not implemented in ParadisEO. The programmer has the burden to develop them using the object-oriented specialization mechanism.

At each layer of the ParadisEO architecture, a set of classes is provided (Fig. 1.32). Some of them are devoted to the development of metaheuristics for monoobjective and multiobjective optimization, and others are devoted to manage transparently parallel and distributed models for metaheuristics and their hybridization.

There are two programming mechanisms to extend built-in classes: function substitution and subclassing. By providing some methods, any class accepts that the user specifies his own function as a parameter that will be used instead of the original function. This will avoid the use of subclassing, which is a more complex task. The user must at least provide the objective function associated with his problem.

1.9 CONCLUSIONS

When identifying a decision-making problem, the first issue deals with modeling the problem. Indeed, a mathematical model is built from the formulated problem. One can be inspired by similar models in the literature. This will reduce the problem to well-studied optimization models. One has also to be aware of the accuracy of the model. Usually, models we are solving are simplifications of the reality. They involve approximations and sometimes they skip processes that are complex to represent in a mathematical model.

Once the problem is modeled, the following roadmap may constitute a guideline in solving the problem (Fig.1.33).

First, whether it is legitimate to use metaheuristics for solving the problem must be addressed. The complexity and difficulty of the problem (e.g., NP-completeness, size, and structure of the input instances) and the requirements of the target optimization problem (e.g., search time, quality of the solutions, and robustness) must be taken into account. This step concerns the study of the intractability of the problem at hand. Moreover, a study of the state-of-the-art optimization algorithms (e.g., exact and heuristic algorithms) to solve the problem must be performed. For instance, the use of exact methods is preferable if the best known exact algorithm can solve in the required time the input instances of the target problem. Metaheuristic algorithms seek good solutions to optimization problems in circumstances where the complexity of the tackled problem or the search time available does not allow the use of exact optimization algorithms.

At the time the need to design a metaheuristic is identified, there are three common design questions related to all iterative metaheuristics:

- **Representation:** A traditional (e.g., linear/nonlinear, direct/indirect) or a specific encoding may be used to represent the solutions of the problem. Encoding plays a major role in the efficiency and effectiveness of any metaheuristic and constitutes an essential step in designing a metaheuristic. The representation must have some desired properties such as the completeness, connexity, and efficiency. The encoding must be suitable and relevant to the tackled optimization problem. Moreover, the efficiency of a representation is also related to the search