3
Overview of the Evolutionary Design System

3.1 Introduction

When applying a genetic algorithm to any new application, four main elements must be considered. First, the phenotype must be specified, i.e. the allowable solutions to the problem must be defined by the specification and enumeration of a search space. Second, the genotype (or coding of the allowable solutions) must be defined. Third, the type of genetic algorithm most suitable for the problem must be determined. Fourth, the fitness function must be created, in order to allow the evaluation of potential solutions of the problem for the GA.

Since it is proposed that a genetic algorithm is to be used to form the core of the system introduced in this thesis, these four elements can be identified in the generic evolutionary design system. Designs are searched for using a multiobjective genetic algorithm as the 'search-engine' to evolve solutions. To achieve this, the GA manipulates hierarchically organised genotypes (or coded solutions). The genotypes are mapped to phenotypes (or designs) defined by a low-parameter spatial-partitioning representation. These phenotypes are analysed by modular evaluation software, which provides the GA with multiple fitness values for each design. Figure 3.1 illustrates how these four elements are combined to allow the evolution of a range of different solid object designs from scratch.

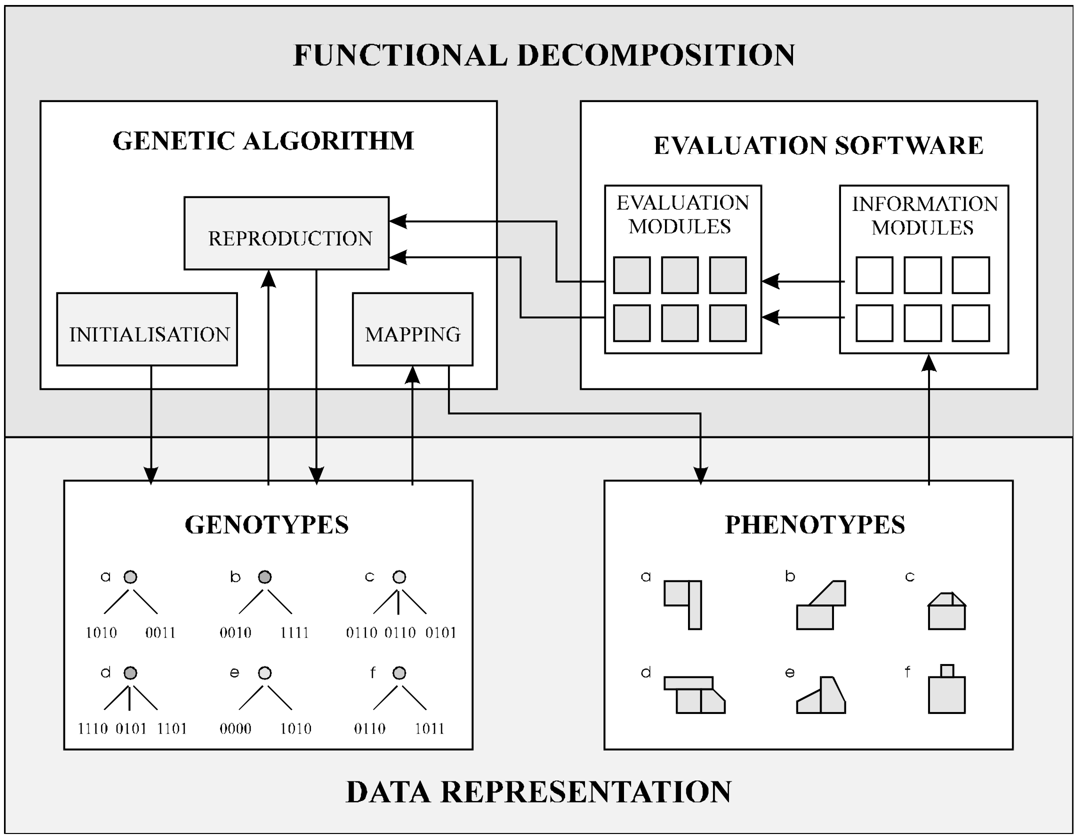

Fig. 3.1 Block diagram of the evolutionary design system.

3.2 System Organisation

This section outlines the organisation of the generic evolutionary design system and gives a brief overview of the elements of the system. Each of these four elements are fully explained and justified in the following four chapters.

3.2.1 Phenotype

Designs, or phenotypes, are defined by a spatial-partitioning representation. This representation combines methods from constructive solid geometry (CSG) and traditional spatial partitioning representations, to allow the definition of a wide range of solid objects using a number of primitive shapes in combination (Bentley & Wakefield, 1996b). Primitive shapes consist of a rectangular block or cuboid with variable width, height and depth, and variable three dimensional position. Every block can also be intersected by a plane of variable orientation (see fig. 3.2), to allow the approximation of curved surfaces. Blocks require nine parameters to fully define their geometry. Designs are defined by a number of non-overlapping blocks or primitives.

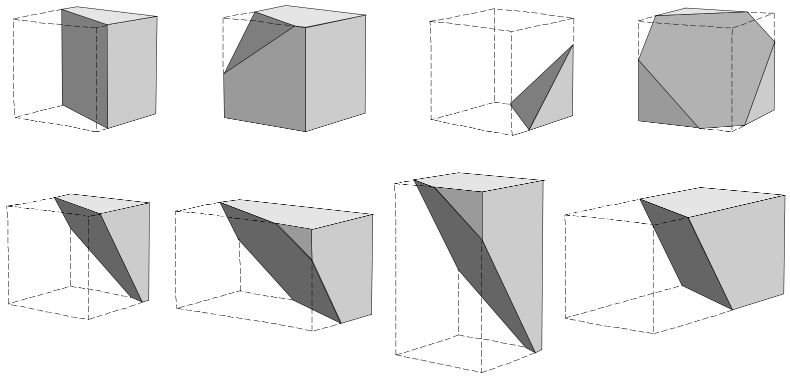

Fig. 3.2 Examples of primitive shapes used to represent designs.

A phenotype simply consists of a flat list of ordered parameters which define the geometry of every primitive making up the design. Figure 3.3 shows the list of parameters in an evolved phenotype and the corresponding design that they specify.

```
40, -15, 25, 80, 50, 50, 2.8594, -2.258, 16.80

-25, -15, 25, 50, 50, 50, -0.703, -0.703, -2.398

40, -15, 25, 80, 50, 50, 0.2822, -8.510, 16.80

-25, -15, 25, 50, 50, 50, 3.8447,-6.986, -2.398
```

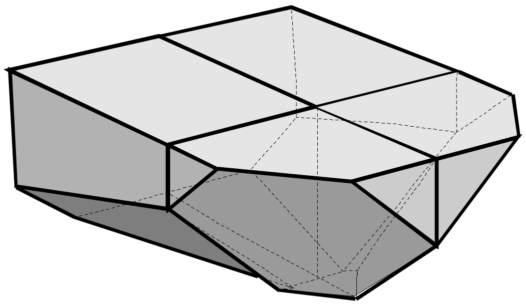

Fig 3.3 An example of a simple four-primitive evolved phenotype (the hull of a small boat).

3.2.2   Genotype

The genetic algorithm within the system never directly manipulates phenotypes. Only coded designs, or genotypes are actually modified by the genetic operators of the GA. Every genotype consists of a single chromosome arranged in a hierarchy consisting of multiple blocks of nine genes, each gene being defined by sixteen bits, see fig. 3.4. This arrangement corresponds to the spatial partitioning representation used to define the phenotypes, with each block of genes being a coded primitive shape and each gene being a coded parameter.

A mutation operator is used within the genetic algorithm to vary the number of primitives in a design by adding or removing new blocks of nine genes from chromosomes. This permits evolution to optimise the number of primitives in addition to the geometries of primitives in designs. However, varying the length of chromosomes in this way can cause the crossover operator to produce meaningless offspring. To overcome this, a new type of crossover operator, known as hierarchical crossover, was developed. This new version of the genetic operator uses the hierarchical arrangement of the chromosomes to find points of similarity between two chromosomes of different sizes. Once such points are found, hierarchical crossover uses the tree-structure of the chromosomes to efficiently generate new offspring without loss of meaning (Bentley & Wakefield, 1996d).

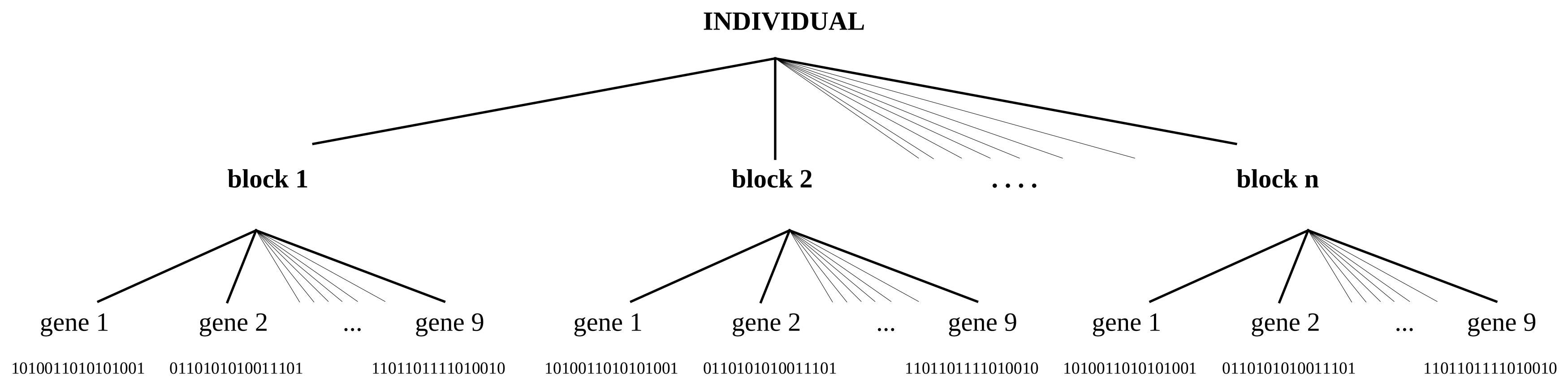

Fig 3.4 Hierarchical genotype of a single individual in the population.

3.2.3 Genetic Algorithm

The genetic algorithm used within the system is slightly different compared to the simple GA outlined in the previous chapter (see fig. 3.5). For example, two populations of solutions are maintained: the main *external population*, and the smaller *internal population*. All new solutions are held in the internal population where they are evaluated. They are then moved into the external population (i.e. 'born' into the 'real world'), replacing only the weakest members of the external population. Other different features include the use of an explicit mapping stage between genotypes and phenotypes, and the use of multiobjective techniques within the GA.

To begin with, the GA has the internal population of solutions initialised with random values to allow the evolution of designs from scratch (see fig. 3.5, box 1). However, if required, a combination of random values and user-specified values can be used to allow the evolution of pre-defined components of designs, or of selected parts of designs (Bentley & Wakefield, 1996e).

The GA then uses an explicit mapping stage to map the genotypes to the phenotypes (fig. 3.5, box 2). This resembles nature, i.e. the DNA of an organism is never 'evaluated' directly; first the phenotype must be grown from the 'instructions' given in the DNA, then the phenotype is evaluated (Dawkins, 1986). By performing this process explicitly, the system is able to gain some advantages. For example, should a symmetrical design be required, only half a design needs to be coded in the genotype and hence evolved by the GA. This partial design can then be reflected during the mapping stage to form a complete design, which is then evaluated (Bentley & Wakefield, 1995b).

Next, the GA calls the relevant user-specified evaluation software to analyse the phenotypes and obtain multiple fitness values for each individual solution (most design problems are multicriteria problems), see fig. 3.5, box 3. The GA must then determine from these multiple fitness values which individuals are fitter overall than others. In other words, the GA has to be able to calculate a single overall fitness value (fig. 3.5, box 4), using multiobjective

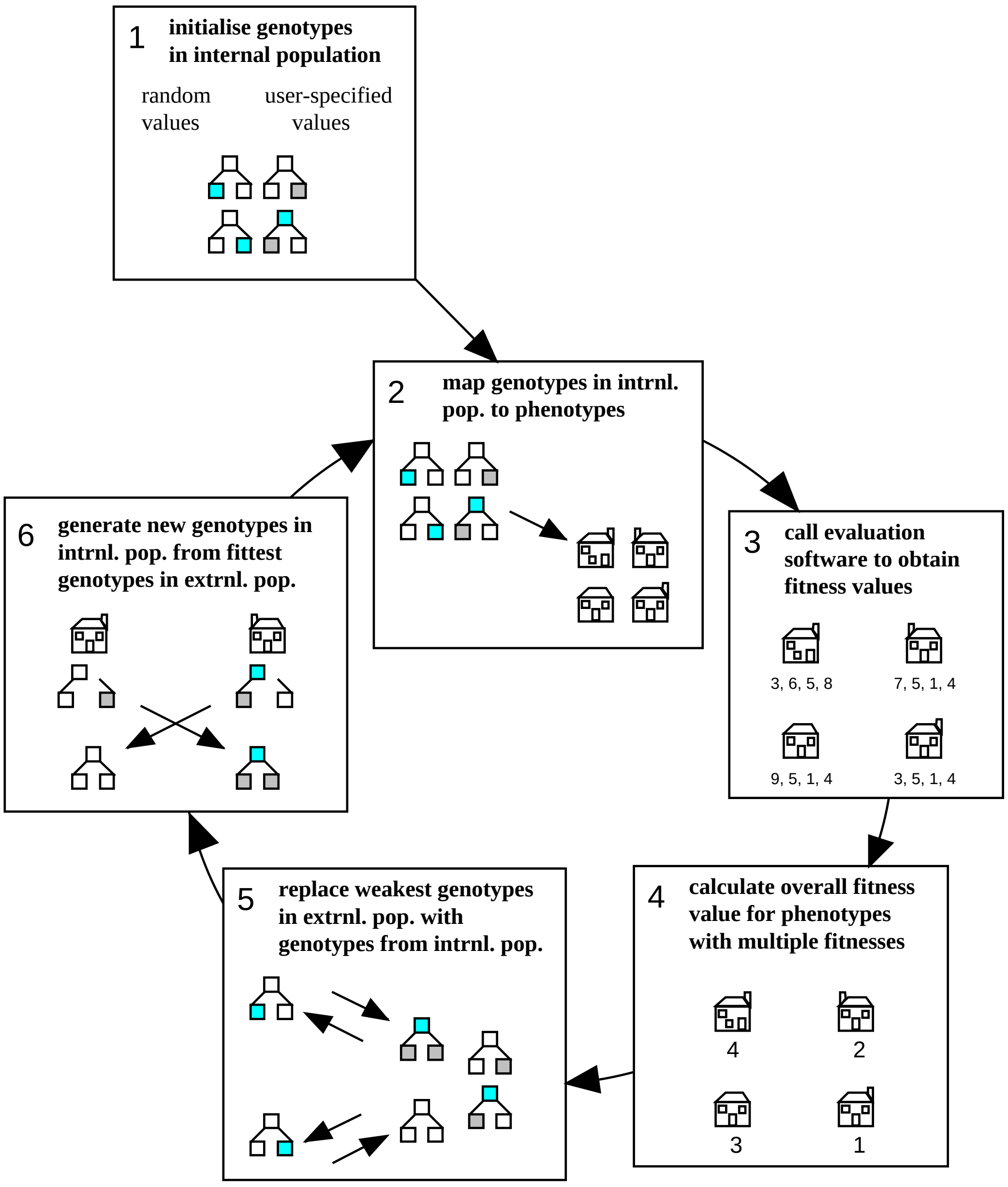
1 initialise genotypes in internal population
random values
user-specified values
2 map genotypes in intrnl. pop. to phenotypes
3 call evaluation software to obtain fitness values
3, 6, 5, 8
7, 5, 1, 4
9, 5, 1, 4
3, 5, 1, 4
4 calculate overall fitness value for phenotypes with multiple fitnesses
4
2
3
1
5 replace weakest genotypes in extrnl. pop. with genotypes from intrnl. pop.
6 generate new genotypes in intrnl. pop. from fittest genotypes in extrnl. pop.

Fig. 3.5 The genetic algorithm used as the core of the system.

optimisation techniques to handle the many separate fitness values produced by the evaluation software (Bentley & Wakefield, 1996f). The new method used to calculate this overall fitness value also incorporates the concept of 'importance', allowing a user to increase or decrease the relative importance of any criteria. (Note that the words 'multiobjective', and 'multicriteria' are used interchangeably throughout this thesis.)

The GA then moves the individuals from the internal population where all new individuals are held, into the main external population (fig. 3.5, box 5). However, unlike the simple GA, this GA does not replace an entire population of individuals with new individuals every generation. In a similar way to the steady-state GA (Syswerda, 1989), this GA only replaces the weakest (less fit) individuals in the external population with new individuals from the smaller internal population, allowing the fittest individuals to remain in the external population over multiple generations (Bentley & Wakefield, 1996c). Unusually, the GA also prevents very fit individuals from becoming immortal (i.e. never being replaced by offspring) by giving every individual in the external population a pre-defined lifespan. Once the individual reaches this lifespan, they become very unfit and thus are quickly 'killed' by new individuals taking their places.

Finally, the GA favours individuals with higher overall fitnesses when picking 'parents' from the external population. The randomly chosen parent solutions (with fitter solutions preferentially selected) are then used to generate a new internal population of offspring using crossover and mutation operators (fig. 3.5, box 6). These operators are more advanced than those found in the simple GA. For example, mutation can not only alter the values of genes in individuals, but also the number of genes in individuals. Moreover, the crossover operator can efficiently generate new chromosomes from two parent chromosomes of different sizes without loss of meaning (Bentley & Wakefield, 1996d).

The GA then maps the new genotypes to the phenotypes, evaluates the new phenotypes, and continues the same process as before. This iterative process continues until either a specified number of generations (i.e. loops) have passed, or until an acceptable solution has emerged.

3.2.4 Evaluation Software

All parts of the system described so far are generic, i.e. they can be applied to a wide range of different solid-object design problems. However, there is an element of the system that will inevitably be specific to individual design applications: the evaluation software. As mentioned previously, designs must be evaluated to instruct the GA how fit they are, i.e. how well they perform the desired function described in the design specification. Hence, the evaluation software is a software version of the design specification, which must be changed for every new design task.

In an attempt to reduce the time needed to create evaluation software for a new design problem, all parts of the various different types of evaluation software created so far have been implemented as re-usable modules (Bentley & Wakefield, 1996c). In other words, it is proposed that many designs can be specified by using a number of existing evaluation modules in combination. Moreover, wholly new design tasks will only require the creation of modules of evaluation software that do not already exist, thus dramatically shortening the time needed to apply the system to a new application. Over time a large library of such modules could be developed, to reduce the future need for new modules. Figure 3.7 shows some of the existing modules in the library of evaluation software developed as part of this project.

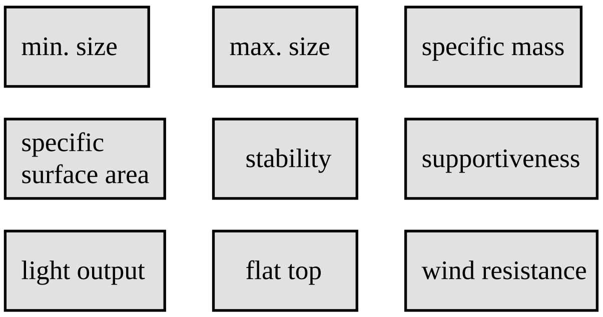

Fig 3.6 Examples of evaluation software modules in current library.

In addition to a library of different evaluation software modules (or fitness functions), a library of phenotype information modules is maintained. This is necessary because many modules of evaluation software require specific information about a design in order to calculate how fit that design is. Using a distinct information module to calculate, say, the mass of a design, allows all evaluation modules that need this value to share the information generated. In other words, such information on phenotypes need only be generated once, to supply all evaluation modules that require it. Figure 3.7 shows some of the information modules in the library developed as part of this project.

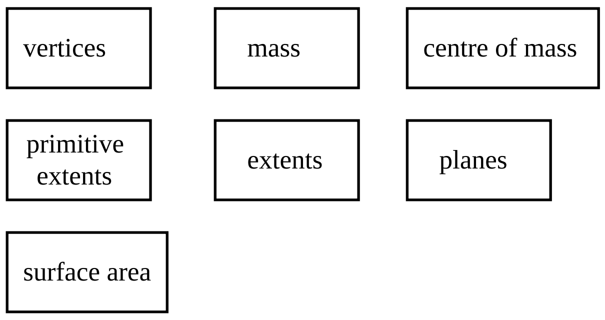

Fig 3.7 Examples of phenotype information modules in current library.

For every evaluation module selected by a user to partially specify a design application, there will be corresponding information modules that are automatically used by the system. Additionally, every evaluation module requires the user to input a number of desired parameter values. For example, if a design was required which was streamlined in shape, the 'wind resistance' module would be selected. This would automatically enable the information modules 'primitive extents', 'extents' and 'planes', with the user then needing to input parameter values to specify the forces (generated by the air-flow on the design) that were required. When evaluating phenotypes during evolution, the module would then provide the GA with fitness value(s) that correspond to how well the phenotype allows air to flow past itself, see fig. 3.8.

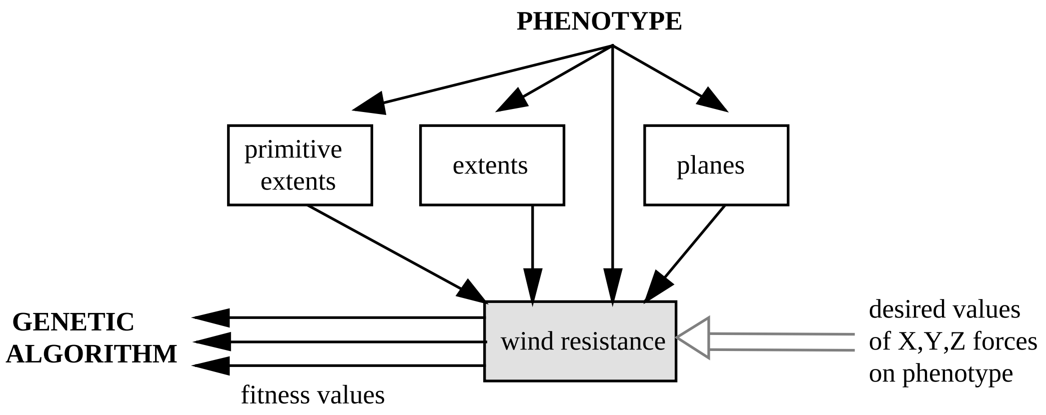

Fig. 3.8 Using an evaluation software module with phenotype information modules.

Consequently, complete design applications are specified to the evolutionary design system by the selection of a combination of modules of evaluation software, and the input of desired parameter values. The system then enables the appropriate information modules which supply all of the evaluation modules with the necessary information on the phenotype. A number of separate fitness values are generated by the evaluation modules for each design, which are used by the GA to guide evolution to good solutions, see fig. 3.9.

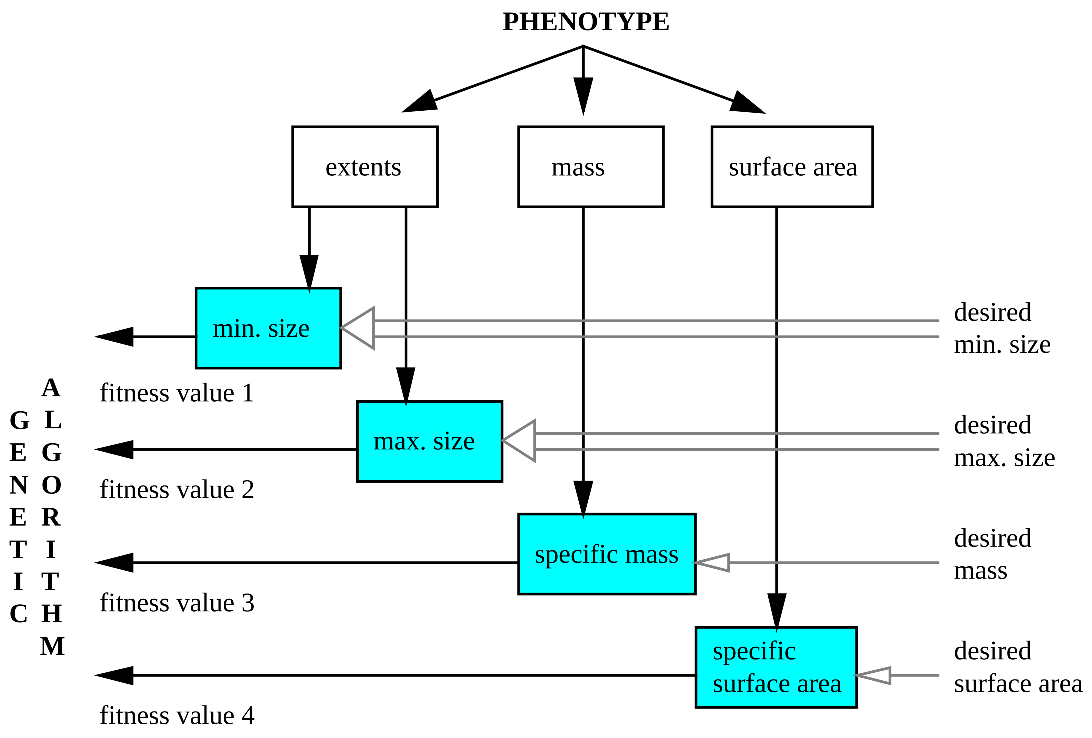

Fig 3.9 Specifying a simple application using modules of evaluation software, phenotype information modules and user-specified parameter values.

3.3 Summary

This chapter has given an overview of the generic evolutionary design system. Each of the four distinct elements of the system was identified and summarised. First, the new spatial-partitioning representation used within the phenotypes to define solid object designs was described. Second, the hierarchical structure and coding of the genotypes was shown, and it was explained that this structure allows a novel hierarchical crossover operator to efficiently generate meaningful new chromosomes from two chromosomes of different lengths. Third, it was explained that the core of this system consists of a multiobjective genetic algorithm with a distinct mapping stage between genotypes and phenotypes, overlapping populations, preferential selection of parents, and other advanced features such as chromosomes of variable lengths, and lifespans. Fourth, the evaluation software was outlined, with an explanation of how information modules are used in combination with user-definable evaluation modules to provide the fitness values of phenotypes to the GA.

These four elements: phenotype, genotype, GA, and evaluation software, will be fully explained in the following four chapters of this thesis.