Agent–based Support within an Interactive Evolutionary Design System

Dragan Cvetković
Soliton Associates Ltd.
44 Victoria Street 2100
Toronto ON M5C 1Y2
Canada
email: dragan@soliton.com
Phone: +1 (416) 364 9355

Ian Parmee
University of the West of England
Frenchay Campus
Bristol BS16 1QY
United Kingdom
email: ian.parmee@uwe.ac.uk
Phone/Fax +44 (0)117 344 3137

9th September 2002

Number of pages (without figures): 24, number of figures: 6, number of tables: 0.

Agent–based Support within an Interactive Evolutionary Design System

Abstract

This paper describes the use of software agents within an interactive evolutionary conceptual design system. Several different agent classes are introduced (search agents, interface agents and information agents) and their function within the system is explained. A preference modification agent is developed and an example is given illustrating the use of agents in preference modelling.

Keywords: Software Agents, Conceptual Design, Multi–objective Optimisation, Preferences.

1 Introduction

Although application of evolutionary and adaptive computing technologies for design optimisation is now well-established, there is little recognition of their potential for design exploration through appropriate integration with conceptual design processes. Such integration supports search across predefined design spaces, whilst also allows exploration outside of initial constraint and variable parameter bounds. Close designer interaction allows exploration involving off-line processing of initial results which leads to a redefinition of the design space. Further designer/evolutionary search of redefined space can lead to the discovery of innovative or even creative solutions (Parmee, 1999; Parmee, 1998).

Entirely machine–based conceptual design is not suggested here, nor is currently considered viable. Best utility can be achieved from systems that enhance the designer's inherent capabilities. Appropriate integration can result in the development of prototype evolutionary design tools that provide powerful extensions to design team activity by supporting rapid, extensive exploration and stimulating innovative reasoning. This paper therefore discusses the use of agent based methods for both search/exploration and for the support of the designer in the design process. It represents a synopsis of the second part of PhD thesis research presented in (Cvetković, 2000), dealing with application of agents.

The idea of using preferences and agents in Conceptual Engineering Design is not new. Some aspects of the research are presented by D' Ambrosio and Birmingham (D'Ambrosio and Birmingham, 1995), Wellman (Wellman and Doyle, 1991; Wellman, 1995; Wellman and Walsh, 2000) and many other researchers.

The paper is organised in the following manner: Section 2 briefly describes the interactive evolutionary conceptual design system (IEDS), section 3 introduces preferences, whereas section 4 introduces agents. In section 5 the use of agents within the system is discussed and some classes of agents are introduced. Section 6 provides an example of agent use. Finally, section 7 provides conclusions, discussion and pointers to future work. More details, in a wider context, are given in (Cvetković, 2000). The IEDS is described in (Parmee et al., 2000; Parmee et al., 2001) and the preferences are described in more details in (Cvetković, 2000; Cvetković and Parmee, 2002).

2 The IEDS

Conceptual design represents the initial phase of a design process (Pahl and Beitz, 1996). The research presented here is based on whole system airframe design in cooperation with British Aerospace (BAE) Systems Ltd. Some design issues have been presented elsewhere (Parmee and Purchase, 1997; Cvetković et al., 1998). A major characteristic of conceptual design relates to innovation and creativity which is very well encapsulated by the following quote (Goel, 1997):

> "...problem formulation and reformulation are integral parts of creative design. Designers' understanding of a problem typically evolves during creative design processing. This evolution of problem understanding may lead to (possible radical) changes in the problem and solution representations."

Therefore, an Interactive Engineering Design System (IEDS) (Parmee et al., 2000; Parmee et al., 2001) has been developed in order to assist the designer during conceptual design. The core of the IEDS is described in Figure 1. It consists of the following modules:

[Figure 1 about here.]

Information gathering processes: (e.g. cluster oriented genetic algorithms COGAs (Parmee, 1996; Bonham and Parmee, 1998; Parmee and Bonham, 2000)) a module that constantly extracts relevant information from the search processes and presents it to the design team via machine-based agents. COGAs support the rapid decomposition of complex, multi-variate design space into regions of high performance and the extraction of relevant design information from such regions through good solution cover.

Preference module: a module for specifying the relative importance of objectives and constraints (Cvetković and Parmee, 1999b; Cvetković and Parmee, 2002). Its task is to help the designer in this process by introducing several categories of importance of objectives (much less important, less important, equally important, more important and much more important) linguistically and to transfer these values, using concept of leaving score and induced ordering (Fodor and Roubens, 1994), into weights used throughout the optimisation process.

- **Distributed co–evolutionary genetic algorithm:** a module for multi–objective optimisation (Parmee and Watson, 1999; Parmee et al., 2000), supporting the identification of high performance regions of a multi–dimensional Pareto frontier. Each objective is assigned a separate optimisation process with the task of optimising (minimising or maximising) that objective only. At the beginning all optimisation processes are independent of each other but as the run progresses a penalty, relating to maximal allowable Euclidean distance between the variables of each evolutionary process, is used to lead obtained solutions towards a common region. If a variable is outside a range defined by a range constraint map, the associated solution fitness is adjusted by a penalty function. The communication between processes is implemented using Parallel Virtual Machine (PVM) software package (Geist et al., 1994).

- **Problem decomposition module:** a part of the distributed co–evolutionary module that uses Taguchi methods (Peace, 1993) for identifying the sensitivity of several differing objectives to individual variable parameters;
- **Database module:** a module for storing interesting and promising solutions and training data.

The two components under consideration within the paper are the distributed co–evolutionary genetic algorithm and preference module. Preferences are used in a co–evolutionary context to change the value of penalties depending on importance factors: more important objectives are penalised less, the less important objective are penalised more. In that way, processes compete for the best solution, since the penalty function discourages the solutions that are far apart.

Examples of different algorithms and different design aspects are illustrated through the joint research project with British Aerospace (BAE) Systems. The details of the project are described in more details in (Cvetković et al., 1998; Cvetković, 2000; Parmee et al., 2000; Parmee et al., 2001) and also below in section 5.

An example of the integration of these two modules is presented in Figure 2 showing the influence of preference settings on the co–evolutionary optimisation processes. It shows the optimisation (maximisation) of 2 objectives: both processes S_0 and S_1 work on objectives y_3 (specific excess power, SEP1) and y_9 (ferry range, FR), but the process S_0 tries to maximise objective y_3, whereas process S_1 tries to maximise objective y_9 (the objectives are conflicting). The plots show the values of objective y_3. Figure 2(a) shows the optimisation results using preference $y_3 \ll y_9$ (i.e. "objective y_3 is much less important than y_9"), Figure 2(b) with equal preferences ($y_3 \approx y_9$) and Figure 2(c) shows the optimisation results using preference $y_3 \gg y_9$ (i.e. "objective y_3 is much more important than y_9"). These preferences direct search towards different regions of y_3 vs. y_9 values: if $y_3 \ll y_9$ the search processes will converge towards smaller values for y_3 (and larger values for y_9), as illustrated in Figure 2(a), if they are considered equally important, they will converge towards compromise regions where both objectives are "average" (Figure 2(b)). Similar results are obtained by plotting objective y_9 instead of y_3.

[Figure 2 about here.]

Figure 2 demonstrates how preferences control the search process driving the compromise region towards the one with better values for the more important objective. It can also be noted in Figure 2(a) that the results of the two optimisation processes do not converge to the same extent as they do with equal preferences (Figure 2(b)). This behaviour (i.e. the result difference) could be explained by noting that the more important objective is penalised less: the solution that is usually penalised if the Euclidean distance between variables is more than for instance 10%, will now be penalised if the distance is more than for instance 20%.

The penalty–factor corrected value of objective y_i, $f_i(\boldsymbol{x}, t)$ was calculated using the following formula:

$$f_i(\boldsymbol{x},t) = f_i^0(\boldsymbol{x}) \cdot \prod_{j=1}^{k} \chi_d(x_j, x_i, t), \quad \text{where}$$
$$\chi_d(x,y,t) = \begin{cases} \varphi_p \cdot \theta_p(x,y), & |x-y| \geq d_p(t); \\ 1, & \text{otherwise.} \end{cases} \tag{1}$$

for

$$\theta_p(x,y) = \min\{1, w_x/w_y\}, \tag{2}$$

w_s is the weight of objective s, $f_i^0(\boldsymbol{x})$ is the original value of objective y_i for a given set of inputs $\boldsymbol{x}$, t is the generation number, $d_p(t)$ is the monotonically decreasing function specifying minimal non-penalised distance and φ_p is the original penalty factor (usually 0.5).

During the design process, the designer is able to change the preferences and objectives to optimise and to dynamically add, modify and delete constraints (scenarios) with an almost immediate feedback from the system, showing the influence of the changes on the solutions generated.

During the later phases of design few objectives tend to be in evidence, whereas during conceptual design the designer requires a global picture and, therefore, routinely deals with many possible objectives. The vast number of parameters involved can confuse the designer. In order to reduce cognitive overload and to help the designer in mundane and less creative tasks, a set of agents has been developed that are an integral part of the IEDS. The components relevant to the research described in this paper are marked by dotted lines in Figure 1. They are described in section 5 after an introduction to preferences and agents.

3 Preferences

The notion of preferences is not now, and different authors propose different preference systems in engineering design (Wellman and Doyle, 1991; D'Ambrosio and Birmingham, 1995; Greenwood et al., 1996). The authors have developed a preference systems for specifying the relative importance of objectives. The following predicates have been introduced (Cvetković and Parmee, 1999b; Parmee et al., 2000; Cvetković and Parmee, 2002; Cvetković, 2000):

relation	intended meaning
$\approx$	is equally important
$\prec$	is less important
$\ll$	is much less important
$\neg$	is not important
$!$	is important

and the set of axioms specifying the following properties of these relations:

- $\approx$ is an equivalence relation;
- $\prec$ and $\ll$ are strict orders;
- $\approx$ is *congruent* with $\ll$ and $\prec$;
- $\ll$ is subrelation of $\prec$;
- misc. properties:

$$!x \vee \neg x \tag{3}$$

$$!y \wedge \neg x \Rightarrow x \ll y \tag{4}$$

$$\neg x \wedge \neg y \Rightarrow x \approx y \tag{5}$$

$$x \prec y \wedge y \ll z \Rightarrow x \ll z \tag{6}$$

The corresponding, “more important” ($\succ$) and “much more important” ($\gg$) relations are defined as

$$x \gg y \overset{\text{def}}{\Leftrightarrow} y \ll x \tag{7}$$

$$x \succ y \overset{\text{def}}{\Leftrightarrow} y \prec x \tag{8}$$

The above cited work describes the use of preferences within multi–objective optimisation (weighted sums optimisation, weighted Pareto optimisation etc). The current paper tries to merge preferences with agents i.e. to use agents to automate preference estimation.

4 Agents

Defining the notion of an agent is a very difficult task as the following quote demonstrates. According to Watt (1996, p. 89):

> "...'Agent' is a difficult word for a difficult concept; covering a rag-bag of concepts that span a whole gamut of different kinds of behaviour, including, for example, autonomy, learning and social interaction; but there is a common ground. An agent will set out to do something, and do it; therefore it has competences for intending to act, for action in an environment, and for monitoring and achieving its goals. Of course, the adequate performance of these, other competencies, such as learning, negotiation, and planning, may be helpful or even necessary."

An overview of theoretical aspects of agents (including topics such as belief, intention, default reasoning, possible world semantics etc.) is given in (Wooldridge and Jennings, 1995).

In the most general sense, agents can be classified into the following categories (Stenmark, 1999): interface agents, system agents, advisory agents, filtering agents, retrieval agents, navigation agents, monitoring agents, recommender agents and profiling agents. However, for the purposes of our application, conceptual design, the following classes of agents appear to offer utility:

Interface agents: agents that help the designer deal with a system and which (if designer wishes it) hide some low–level non-interesting details from the designer;

Search agents: agents that cover the process of optimisation, cooperation, population monitoring, jumping out of regions, constraint questioning etc.

Information agents: agents that deal with information obtained, look for interesting solutions, filter uninteresting ones, make decision with regard to what and where to explore, resolve conflicts etc.

Figure 3 classifies the agents used throughout the project. A similar classification is used by Sycara et al. (1996).

[Figure 3 about here.]

Since the role of agents necessitates collaboration (negotiations) and interaction, these two concepts are important and they will be described in the following two subsections.

4.1 Negotiations

According to Sycara (1991), there are four conflict situations where negotiation is used in design. These conflicts are (Berker, 1995):

- Different agents make conflict recommendations for a parameter value;

- A value proposed by one agent makes it impossible for another agent to offer consistent values for other attributes;
- A decision of one agent adversely affects the optimality of other agents;
- Alternate approaches achieve similar functional results.

The negotiation process proceeds as follows:

1. Generation of proposal;
2. Generation of counter proposal based on feedback from dissenting agents;
3. Communication of justifications and supporting evidence.

The paper by Nwana et al. (1996) gives an overview of different coordination techniques.

4.2 Agent communication

Agents need a common language in order to be able to communicate. The first developed methods used a blackboard architecture (Hayes-Roth, 1985; Brenner et al., 1998), as presented in Figure 4(a), where all agents are able to read from and write to a shared memory area. The other method utilises directed message passing from agent to agent, as shown in Figure 4(b) using message transport methods (e.g. PVM, MPI, etc). More modern agent communication languages (ACL) are described in (Labrou et al., 1999)).

[Figure 4 about here.]

5 Use of agents in design processes

A very successful agent–based application is described in (Ygge and Akkermans, 1999): it describes a climate control of large buildings with many office rooms using a "market based agent approach". Agents buy and sell cooling power resources (Huberman and Clearwater, 1995). A very comprehensive review of computer supported cooperative environments for engineering design is given in (Shen and Norrie, 1999). Wellman describes "market–oriented programming" in (Wellman, 1995; Wellman, 1996). Some issues are also discussed in (Lander, 1997). An agent based system for conceptual design is described in (Campbell et al., 1999; Campbell, 2000).

In our development of agents, it has been decided to follow the philosophy of simple agents: an agent performs only one function, similar to single function agents (SIFA) (Brown et al., 1995; Berker, 1995). Single function agents are designed to perform one function only and they have the following parameters:

Function: what kind of work it performs;

Target: on what parameter or object the agent has an immediate effect;

Point of view: the perspective that the agent takes in performing its function on its target. The point of view can be cost, strength etc.

Brown et al. (1995) argue that in this way it is much easier to construct new agents and (equally important) it is much easier to debug agents.

Following the classification of agents given in section 4, the agents developed for the BAE Systems conceptual engineering design system are schematically presented in Figure 3. They mostly follow the above SIFA philosophy: single function per agent.

The conceptual airframe design project (or miniCAPS project, explained below) has been developed in cooperation with BAE Systems. The details of the project are not relevant for this paper and it is not possible to describe the model in detail. However, a brief summary follows.

The miniCAPS model (Webb, 1997) is a version of CAPS (Computer Aided Project Studies), BAE Systems software used by designers during the earliest investigation stages of a new aircraft. MiniCAPS reproduces the general characteristics of CAPS but without the computational complexity. MiniCAPS models a variety of disciplines and consists of three modules: **Aerodynamics** (lift and drag coefficients, flight envelope etc.); **Performance** (ferry range, sustained turn rate, take-off distance, cruise height etc.) and **Configuration** (wing position, wing shape, canard position, number of engines, mass estimation etc.). A high degree of interaction is incorporated between these disciplines and many of the objectives are thus highly conflicting.

One of the goals of the project was a development of an (semi–)intelligent and (semi–)autonomous system that will utilise preferences and agents in order to reduce designer's burden through the performance of more mundane tasks, enabling designer to concentrate on more creative aspects of the design.

At present miniCAPS utilises 9 variable parameters producing a total of 13 outputs, each of which may be considered an objective.

5.1 Interface agents

Interface agents are used to reduce the complexity of increasingly sophisticated and overloaded conceptual design systems. They build an (user friendly) interface between the designer and the computer. The designer can specify the quality threshold of solutions, situation trigger actions and other parameters.

Their role is to help the designer in a (boring) Q&A preference estimation procedure (e.g. "Statement: A is the most important to me and B the least important of all the objectives" the agent transforms into a series of questions and answers suitable for the preference module).

5.1.1 Application within the system

An agent has been implemented in the system that helps the designer in the preference estimation procedure. It allows the designer to specify the complete preference order on the command line or in a file e.g.:

$$y_9 \approx y_{10} \gg y_1 \approx y_2 \approx y_6 \succ y_3 \approx y_4 \succ y_5 \gg y_7 \approx y_{11} \succ y_8$$

instead of answering the following standard sequence of questions:

$$\begin{aligned}
& y_9 \approx y_{10} \\
& y_1 \approx y_2 \approx y_6 \\
& y_3 \approx y_4 \\
& y_7 \approx y_{11} \\
& y_1 \succ y_2,\ y_1 \succ y_5,\ y_1 \gg y_7,\ y_1 \gg y_8,\ y_1 \ll y_9 \\
& y_3 \succ y_5,\ y_3 \gg y_7,\ y_3 \gg y_8 \\
& y_5 \gg y_7 \\
& y_5 \gg y_8 \\
& y_7 \succ y_8
\end{aligned}$$

In the initial stage, the designer will probably prefer the second method (pair–wise comparisons), but as the process goes on, specifying the complete order is easier, especially if the changes are incremental (as in the case of the incremental agent described in section 5.4.1). Since the preference method transforms preferences into a total order, these two methods are equivalent, providing that the initial complete order specified is not circular (also checked by the agent).

5.2 Search agents

The role of search agents is to look for "interesting" solutions. Here the notion of being "interesting" is defined by the designer, or for instance a good solution with a large Euclidean distance from the majority of the population. It looks for "novelty" solutions which might be overlooked and ignored otherwise.

Among the search agents the following classes of agents have been investigated and developed:

JumpOut agent: agent that searches exclusively outside of variable boundaries;

Quality monitoring agent: agent that monitors the quality of solutions;

Constraint agent: agent that tries to find out what solutions can be obtained by breaking one of the constraints;

Scenario agent: agent that solves the original problem minus one of the scenarios (dynamically, on–line specified constraints in a relatively rich mathematical language with logical operators (Cvetković and Parmee, 1999a));

Population monitoring agent: agent that monitors the convergence of the population.

These agents are described in subsequent sections. The following need to be considered when applying agents:

- Where to search?
- Variation of each variable parameter range;
- Constraint vs. objective space (i.e. shall we use penalty functions to transform constraints into objectives etc.).

5.2.1 JumpOut agent

This agent searches exclusively out of boundaries. It can initiate a new genetic algorithm that works in parallel with the main one, or it can initiate a quick hill–climber that starts from one of the solutions, changes a randomly chosen variable to be out of defined range and then performs hill–climbing. Alternatively, it could start modifying good solutions, not just any random population member. Parameters of the agents limit how far outside the domain the agent can go and for how many generations. How many individuals to create could also be specified. The method used could be hill climbing, simulated annealing (Laarhoven and Aarts, 1987), scatter search (Laguna, 2002), differential evolution (Storn and Price, 1995) and many others. If desired, this can be combined with tabu lists (Glover and Laguna, 1997) to remember already explored regions.

This area has also been investigated within the Plymouth Engineering Design Centre (PEDC) through the work of Beck and Parmee (1999).

JumpOut agents are useful for questioning the initial constraints and domain of the problem and for attempting to expand the problem domain. During conceptual design, some limits are set rather ad hoc. This agent assesses whether it is possible to obtain better solutions by ignoring some of the initial limitations.

5.2.2 Quality monitoring agent

If the solution fitness is at least 90% of the best solution, this agent notifies the designer about it. The quality threshold (the percentage of the best solution, 90% in this example) is a configurable parameter. Also, the user can specify when to be notified:

- Immediately, so that the search can be lead in that direction, or
- Afterwards, for off–line analysis.

5.2.3 Constraint agent

This agent tries to break some of the constraints, and if a good solution is obtained, it notifies the designer. Constraints can have different levels of 'softness' assigned (from 0 to 1, or from 'absolutely unchangeable' to 'you can do whatever you want with it'). For each solution information regarding the number of constraints broken (if using penalty functions for resolving constraints) is stored.

Another constraint agent monitors the best solutions and for each of them, tries to further optimise the solution, ignoring (i.e. breaking) one of the constraints. If the obtained solution is significantly better, present it to the designer and let him decide if that constraint is necessary. The designer can mark some of the constraints as non–questionable (as before).

5.2.4 Scenario agent

The scenario agent is similar to a constraint agent, except that it deals with scenarios i.e. set of constraints connected with logical operators AND, OR and NOT (Cvetković and Parmee, 1999a; Cvetković, 2000). Each agent solves the original problem minus one of the scenarios. For m scenarios, that means $m+1$ parallel GAs (one with all scenarios). This could be very costly since parallel search processes are required. However, the increased use of parallel and more and more powerful computers makes such distributed strategies increasingly feasible. One example of scenarios is given in section 6 below.

5.2.5 Population monitoring agent

If the GA search is too concentrated (i.e. too converged in variable space) in one part of the search space, this agent "jumps" far away from the converged point in space (but still within the feasible domain $\mathcal{D}$) and starts a new search there. Bookkeeping about already explored regions is needed in order to avoid visiting the same region many times.

Three levels of spontaneous behaviour are available:

- Machine–based agent automatically decides to try jumping out of regions, breaking constraints etc.;
- The designer only decides on the action taken;
- Interactively: agent suggests and the designer declines or accepts.

5.3 Information agents

This class of agents is more intelligent then the previously described classes and should be able to make autonomous decision concerning (but not limited to):

- "spawning" an agent to search in a given direction;
- "killing" an agent that is not very successful;
- negotiation between agents (unless they need to consult the designer);
- recognition of the novelty of a solution (eventually consulting the database of existing solutions) and the turning designer's attention towards it;
- when to consult the designer.

One example of an information agent is the incremental agent described below in section 5.4.1.

5.4 Closing design loop

In a conceptual design context the agents can be applied in the manner presented in Figure 5.

[Figure 5 about here.]

Common to all optimisation processes, there is a standard search path: Preferences → Search Engine → Output. The new component here is a process that picks up solutions from the search engine and presents them to a consortium of agents that look at it in terms of different interests or points of view (say agent A monitors objective 1, agent B monitors objective 2, ..., agent D monitors variable 1 which, ideally should be between 0 and 5 or between 15 and 20 etc.). This information is presented to the designer together with some suggestions (e.g. "can we change this preference?" or "this solution path is no good for some constraints") and preferences and some mathematical model details are changed (with the designer's approval). This connects agents nicely with the scenario concept. The agents are employed to monitor constraints and scenarios and to analyse those that are never completely fulfilled. The unfulfilled constraints and scenarios usually give an indication about the changes needed to improve the design. Ideally, these changes should be suggested by the agents.

The next section describes an agent that tries to fulfil scenarios through changing preferences and variable ranges. Every time it finds an unfulfilled scenario, it suggests the changes to the designer and if approved, continues search in the modified setting.

5.4.1 Incremental agent

The incremental agent developed in this section will close the design loop as presented in Figure 6. This is a more detailed view of the general IEDS (Interactive Engineering Design System, described in section 2) presented in Figure 1. For the sake of simplicity, in Figure 6 it is assumed that agent and scenario values are incorporated into the fitness value as some form of penalty.

[Figure 6 about here.]

The incremental agent works in the following way:

1. Use the original designer's preferences (both for objectives and for scenarios) and run optimisation process;
2. If some of the scenarios are not fulfilled, suggest increasing importance of those scenarios that are not fulfilled and repeat the search process;
3. If some scenarios are still not fulfilled although they are classified as the most important, suggest changing variable ranges (of those variables mentioned in scenarios) and repeat the search with this new setting;
4. If some scenarios are still not fulfilled, give up and report the results to the designer.

6 Example of agent use

The following example illustrates the use of agents and the automated preference adjustment through the use of incremental agents described in section 5.4.1. The example also illustrates the interactiveness of conceptual design process using the IEDS.

Example 1 Suppose that for BAE System's airframe design problem the following set of scenarios is given:

```
# constraints for F-111 Aardvark Bomber
S1: y11 >= 9.74  & y11<=19.20   # Wing span
S2: x4 <=61.07 & x4 >=48.77     # Wing plan area
S3: y9 >=4707                   # Ferry range
S4: y10 >= 45360                # Take off mass
S5: x3 >= 0.75                  # max cruise speed 919km/h
S6: y1 <= 951                   # take-off run
```

The process goes as follows:

1. The original set of objective preferences i.e.

   $$y_1 \approx y_9 \approx y_{10} \approx y_{11}$$

   and the original set of scenario preferences i.e.

   $$S_1 \approx S_2 \approx S_3 \approx S_4 \approx S_5 \approx S_6$$

   give the solution where $y_{10} = 30936$, so scenario S_4 (that requires that $y_{10} \geq 45360$) is not fulfilled and is noted by the agent.

2. At that stage the agent suggest increasing importance of scenario S_4 i.e.:

   $$S_4 \succ S_1 \approx S_2 \approx S_3 \approx S_5 \approx S_6$$

   but the obtained solution still does not satisfy scenario S_4.

3. At that stage the agent suggests another increase of the importance of the S_4 scenario i.e.:

   $$S_4 \gg S_1 \approx S_2 \approx S_3 \approx S_5 \approx S_6$$

   This again gives a solution where scenario S_4 is not fulfilled and this is noted by the agent.

4. Since the importance of the scenario S_4 cannot be further increased, the agent suggests modifying the variable bounds. It suggests increasing the range of all variables by 10% and starting again.

5. With all equal scenario preferences there is no difference, so S_4 importance increase is suggested again.

6. For a preference setting

$$S_4 \succ S_1 \approx S_2 \approx S_3 \approx S_5 \approx S_6$$

   the solution finally satisfies S_4, but violates scenarios S_1, S_2 and S_6.

7. Agent suggests

$$S_4 \succ S_1 \approx S_2 \approx S_6 \succ S_3 \approx S_5$$

   which again gives a solution where S_1, S_2 and S_6 are satisfied but S_4 is not so the next suggestion is

$$S_4 \gg S_1 \approx S_2 \approx S_6 \succ S_3 \approx S_5$$

   This gives a solution where scenarios S_1, S_2 and S_3 are not fulfilled: $x_3 = 0.92$, $x_4 = 86$, $y_1 = 759$, $y_9 = 290$, $y_{10} = 46982$ and $y_{11} = 23.6$. At this point the agent might call the designer for further assistance.

8. From the analysis so far, the designer can immediately see that he can (using the given airframe model) either have S_4 fulfilled, or S_1 and S_2 (wing span and wing plan area are connected to take-off mass). At this point the designer decides to use the following preferences (the last set of preferences above)

$$S_4 \gg S_1 \approx S_2 \approx S_6 \succ S_3 \approx S_5$$

   and to just increase the range of variable x_4 from $[20, 80]$ to $[20, 120]$, keeping all other variables to their original ranges. This gives a solution $x_3 = 0.87$, $x_4 = 120$, $y_1 = 951$, $y_9 = 7846$, $y_{10} = 49924$ and $y_{11} = 26.8$, that violates scenarios S_1 and S_2 but is probably the best compromise in these circumstances.

9. If the designer further decides to minimise take–off mass instead of maximising, the result is $x_3 = 0.86$, $x_4 = 120$, $y_1 = 878$, $y_9 = 9829$, $y_{10} = 45360$, and $y_{11} = 26.8$.

10. If the designer, out of curiosity, further increase the range of x_4 to $[20, 140]$, the solution is $x_3 = 0.87$, $x_4 = 140$, $y_1 = 951$, $y_9 = 8517$, $y_{10} = 49452$ and $y_{11} = 14.5$, where only scenario S_2 is not fulfilled and the obtained aeroplane has very large wing plan area but small wing span (probably some delta–shaped wings).

    Etc.

As it can be seen from this example, IEDS provides a powerful system for interactive analysis and change of parameters in the run and gives an almost immediate feedback about the objectives, constraints and preferences.

Note that in this particular example the agent did not find an acceptable solution, but by trying different methods it has enabled the designer to find the right one (as much as concept of "rightness" is unambiguous in multi–objective framework).

6.1 Design agent cooperation

Consider a system with several agents, each with a task to optimise a single objective. The question is how to make them collaborate. Each agent is aware of the quality of its own solution. If the quality of one's solution is inferior to the quality of solution of some other agents and their solutions are conflicting, that agent compromises and accepts a worse solution from its point of view, for the benefit of other agents. In a case where they cannot decide (e.g. both agents think that they have quality solutions), the designer is asked to decide. Once the designer resolves a conflict, the agents need to remember the decision and try to learn from it so that the next time a similar situation happens, they can resolve the conflict among themselves without the designer's intervention.

Some form of voting system, where the importance of each agent also plays a certain role in resolving conflicts, can be useful. However, that can lead to problems, since according to Arrow's impossibility theorem (Arrow, 1951), it is not possible to construct a group preference relation satisfying the following basic principles: complete domain, positive association of social and individual ordering, independence of irrelevant alternatives, individual's sovereignty, and nondictatorship.

If an agent is successful, it is made more important than the others (so that good solutions usually count as more important in the negotiation process). For each "best solution", increase the value of the solution. If an agent is less successful, reduce the agent's importance or the quality of its solutions. Limits to both maximal and minimal possible importance of an agent are needed in order to keep diversity of solutions generated. More details are given in (Cvetković, 2000). A similar learning method is used by Campbell (2000).

7 Conclusion and discussion

A frequently asked question is "do we need agents?". Talking about agents, Jennings and Wooldridge (1995) say the following:

> "Although agent based technology clearly has an important role to play in the development of leading edge compound applications, it should not be regarded as a panacea. The majority of applications which currently use agents could be solved using non–agent techniques (in most cases not as well, but in some cases better!). . . . As with all system designs, the ultimate choice depends upon a large number of technical and non–technical factors . . .
>
> Whilst this new system paradigm offers many exciting opportunities, it has a down side which invariably places a limit on the types of application to which agents can be applied. The first major problem is that the overall system is unpredictable and non–deterministic: which agents will interact with which other in which way to achieve what cannot be predicted in advance. Even worse, there is no guarantee that dependencies between the agents can be managed effectively, since the agents are autonomous and free to make their own decisions . . . The second main disadvantage is that the behaviour and properties of the overall system cannot be fixed at design time. While a specification of the behaviour of an individual agent can be given, a corresponding specification of the system in its entirety cannot, since global behaviour necessarily emerges at run time."

In this paper some techniques and methods used in the interactive design system for conceptual design of products have been described. The use of agents is described and some examples of their use presented.

As this research illustrates, the agents do not need complex architectures: simple agents such as the "jump out agent" are very useful in the optimisation process. Their use enables the designer to concentrate on the higher–level aspects of the design process and frees him/her from having to worry about the behaviour of the optimisation module. Filtering and information agents are useful for turning the designer's attention to some interesting and/or unusual (but promising) solutions. They are also useful for questioning all (variable) limits and constraints.

We do not claim that these agents are very intelligent or sophisticated. However, they should be considered in terms of the environment in which they are used i.e. in an interactive design system. In this context however, their usefulness has been illustrated to some extent. In the previous example, the agent did not really solve the problem, but during its work it has collected enough information that the designer was able to see immediately the way to an acceptable solution. The designer is able to solve the problem without using agents, but not without extensive and tedious 'trial and error' runs. The agents described in this article are designed for a conceptual design environment where design goals and constrains are still rather vague and in that environment any help to the designer is important.

Due to the complex problem domain, it is very hard to judge the role of the agents in the system i.e. if they are limiting factor or a factor of improvement in design. The primary role in using agents is pragmatic: they should make the design task easier to the designer.

For the same reason it is very hard to compare the developed system with agent systems described in literature.

Future work will include the development of new classes of agents and their tighter integration in design processes.

References

Arrow, K. J. (1951). *Social Choice and Individual Values*, John Wiley & Sons. 2nd edition 1963, published by Yale University Press.

Beck, M. A. and Parmee, I. C. (1999). Design exploration: Extending the bounds of the search space, *Proceedings of the 1999 Congress on Evolutionary Computation – CEC99*, IEEE, Washington D.C., USA, pp. 519–526.

Berker, I. (1995). *Conflicts and negotiations in single function agent based design systems*, Master's thesis, Worcester Polytechnic Institute, USA. http://www.cs.wpi.edu/Research/aidg/SiFA/ilan.html.

Bonham, C. R. and Parmee, I. C. (1998). Cluster oriented genetic algorithms (COGAs) for the decomposition of multi-dimensional engineering design spaces, *in* B. H. V. Topping (ed.), *Advances in Computational Structures Technology*, Civil-Comp Press, pp. 87–95.

Brenner, W., Zarnekow, R. and Wittig, H. (1998). *Intelligent Software Agents: Foundation and Applications*, Springer-Verlag Berlin Heidelberg.

Brown, D. C., Dunskus, B. V., Grecu, D. L. and Berker, I. (1995). SINE: Support for single function agents, *Applications of AI in Engineering AIENG'95*, Udine, Italy.

Campbell, M. I. (2000). *A-Design: An Electro-Mechanical Design Strategy Utilizing Adaptive Complex Systems and Multi-Objective Optimization*, PhD thesis, Mechanical Engineering Department, Carnegie Mellon, Pittsburgh, PA, USA.

Campbell, M. I., Cagan, J. and Kotovsky, K. (1999). A-design: An agent-based approach to conceptual design in a dynamic environment, *Research in Engineering Design* **11**(3): 172–192.

Cvetković, D. (2000). *Evolutionary Multi–Objective Decision Support Systems for Conceptual Design*, PhD thesis, School of Computing, University of Plymouth, Plymouth, UK.

Cvetković, D. and Parmee, I. C. (1999a). Genetic algorithm–based multi–objective optimisation and conceptual engineering design, *Proceedings of the 1999 Congress on Evolutionary Computation – CEC99*, IEEE, Washington D.C., USA, pp. 29–36.

Cvetković, D. and Parmee, I. C. (1999b). Use of preferences for GA–based multi–objective optimisation, *in* W. Banzhaf and J. D. et al. (eds), *GECCO–99: Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, Orlando, Florida, USA, pp. 1504–1509.

Cvetković, D. and Parmee, I. C. (2002). Preferences and their application in evolutionary multiobjective optimisation, *IEEE Transactions on Evolutionary Computation* **6**(1): 42–57.

Cvetković, D., Parmee, I. C. and Webb, E. (1998). Multi–objective optimisation and preliminary airframe design, *in* I. C. Parmee (ed.), *Adaptive Computing in Design and Manufacture: The Integration of Evolutionary and Adaptive Computing Technologies with Product/System Design and Realisation*, Springer–Verlag, pp. 255–267.

D'Ambrosio, J. G. and Birmingham, W. P. (1995). Preference–directed design, *The Journal of Artificial Intelligence in Engineering, Design, Analysis and Manufacturing (AIEDAM)* **9**: 219–230.

Fodor, J. and Roubens, M. (1994). *Fuzzy Preference Modelling and Multicriteria Decision Support*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V. (1994). *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, Massachusetts.

Glover, F. and Laguna, M. (1997). *Tabu Search*, Kluwer Academic Publishers, Boston.

Goel, A. K. (1997). Design, analogy and creativity, *IEEE Expert, Intelligent Systems & Their Applications* **12**(3): 62–70.

Greenwood, G. W., Hu, X. S. and D'Ambrosio, J. G. (1996). Fitness functions for multiple objective optimization problems: Combining preferences with Pareto ranking, *in* R. K. Belew and M. D. Vose (eds), *Foundations of Genetic Algorithms 4 (FOGA'96)*, Morgan Kaufmann, San Francisco, California, pp. 437–455.

Hayes-Roth, B. (1985). A blackboard architecture for control, *Artificial Intelligence* **26**: 251–321.

Huberman, B. A. and Clearwater, S. (1995). A multi–agent system for controlling building environments, *in* V. Lesser (ed.), *Proceedings of the First International Conference on Multi–Agent Systems, ICMAS'95*, AAAI Press/ MIT Press, pp. 171–176.

Jennings, N. R. and Wooldridge, M. J. (1995). Applying agent technology, *Journal of Applied Artificial Intelligence* **9**(4): 357–369. Special issue on Intelligent Agents and Multi-Agent Systems.

Laarhoven, P. J. M. v. and Aarts, E. H. L. (1987). *Simulated Annealing: Theory and Applications*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Labrou, Y., Finin, T. and Peng, Y. (1999). Agent communication languages: The current landscape, *IEEE Intelligent Systems* **14**(2): 45–52.

Laguna, M. (2002). Scatter search, *in* P. M. Pardalos and M. G. C. Resende (eds), *Handbook of Applied Optimization*, Oxford Academic Press, pp. 183–193.

Lander, S. E. (1997). Issues in multiagent design systems, *IEEE Expert* **12**(2): 18–26.

Nwana, H. S., Lee, L. and Jennings, N. R. (1996). Co-ordination in software agent systems, *BT Technical Journal* **14**(4): 79–89.

Pahl, G. and Beitz, W. (1996). *Engineering Design: A Systematic Approach*, 2 edn, Springer-Verlag, London.

Parmee, I. C. (1996). The maintenance of search diversity for effective design space decomposition using cluster oriented genetic algorithms (COGAs) and multi–agent strategies (GAANT), *in* I. C. Parmee (ed.), *Proceedings of Adaptive Computing in Engineering Design and Control*, PEDC, University of Plymouth, UK, University of Plymouth, Plymouth, UK, pp. 128–138.

Parmee, I. C. (1998). Exploring the design potential of evolutionary/adaptive search and other computational intelligence technologies, *in* I. C. Parmee (ed.), *Adaptive Computing in Design and Manufacture: The Integration of Evolutionary and Adaptive Computing Technologies with Product/System Design and Realisation. Proceedings of the 3rd Conference on Adaptive Computing in Design and Manufacture (ACDM'98)*, Springer–Verlag, pp. 27–42.

Parmee, I. C. (1999). Exploring the design potential of evolutionary search, exploration and optimisation, *in* P. J. Bentley (ed.), *Evolutionary Design by Computers*, Morgan Kaufmann, San Francisco, CA, pp. 119–143.

Parmee, I. C. and Bonham, C. R. (2000). Towards the support of innovative conceptual design through interactive designer / evolutionary computing strategies, *Artificial Intelligence in Engineering, Design, Analysis and Manufacturing (AIEDAM)* **14**(1): 3–16.

Parmee, I. C. and Purchase, G. (1997). Integrating computational intelligence technologies with design and manufacturing team practice, *Proceedings of Intelligent Design in Engineering Applications Symposium (IDEA'97)*, Aachen, Germany, pp. 95–99.

Parmee, I. C. and Watson, A. H. (1999). Preliminary airframe design using co–evolutionary multiobjective genetic algorithms, *in* W. Banzhaf and J. D. et al. (eds), *GECCO–99: Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, Orlando, Florida, USA, pp. 1657–1665.

Parmee, I. C., Cvetković, D., Bonham, C. R. and Packham, I. S. (2001). Introducing prototype interactive evolutionary systems for ill–defined multi–objective design environments, *Advances in Engineering Software* **32**(6): 429–441.

Parmee, I. C., Cvetković, D., Watson, A. H. and Bonham, C. R. (2000). Multi–objective satisfaction within an interactive evolutionary design environment, *Evolutionary Computation* **8**(2): 197–222.

Peace, G. S. (1993). *Taguchi Methods: A Hands–On Approach*, Addison–Wesley, Reading, MA.

Shen, W. and Norrie, D. H. (1999). Agent–based systems for intelligent manufacturing: A state-of-the-art survey, *Knowledge and Information System. An International Journal* **1**(2): 129–156. http://sern.cpsc.ucalgary.ca/CAG/publications/abm.htm.

Stenmark, D. (1999). Evaluation of intelligent software agents, Web page http://w3.informatik.gu.se/˜dixi/agent/agent.htm.

Storn, R. and Price, K. (1995). Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces, *Technical Report TR-95-012*, ICSI, University of Berkeley.

Sycara, K. (1991). Cooperative negotiation in concurrent engineering design, *in* D. Sriram, R. Logcher and S. Fukuda (eds), *Computer–Aided Cooperative Product Development*, Springer Verlag.

Sycara, K., Decker, K., Pannu, A., Williamson, M. and Zeng, D. (1996). Distributed intelligent agent, *IEEE Expert, Intelligent Systems & Their Applications* **11**(6): 36–46.

Watt, S. N. K. (1996). Artificial societies and psychological agents, *BT Technical Journal* **14**(4): 89–97.

Webb, E. (1997). MINICAPS – a simplified version of CAPS for use as a research tool, *Unclassified Report BAe-WOA-RP-GEN-11313*, British Aerospace, Warton, UK.

Wellman, M. P. (1995). The economic approach to artificial intelligence, *ACM Computing Surveys* **27**(3): 360–362.

Wellman, M. P. (1996). Market-oriented programming: Some early lessons, *in* S. Clearwater (ed.), *Market-Based Control: A Paradigm for Distributed Resource Allocation*, World Scientific.

Wellman, M. P. and Doyle, J. (1991). Preferential semantics for goals, *Proceedings of the 9th National Conference on Artificial Intelligence*, Vol. 2, The AAAI Press / The MIT Press, pp. 698–703.

Wellman, M. P. and Walsh, W. E. (2000). Distributed quiescence detection in multiagent negotiation, *Fourth International Conference on Multiagent Systems (ICMAS'2000)*, Boston, MA, USA, pp. 317–324.

Wooldridge, M. J. and Jennings, N. R. (1995). Intelligent agents: Theory and practice, *Knowledge Engineering Review* **10**(2): 115–152.

Ygge, F. and Akkermans, H. (1999). Decentralized markets versus central control: A comparative study, *Journal of Artificial Intelligence Research* **11**: 301–333.

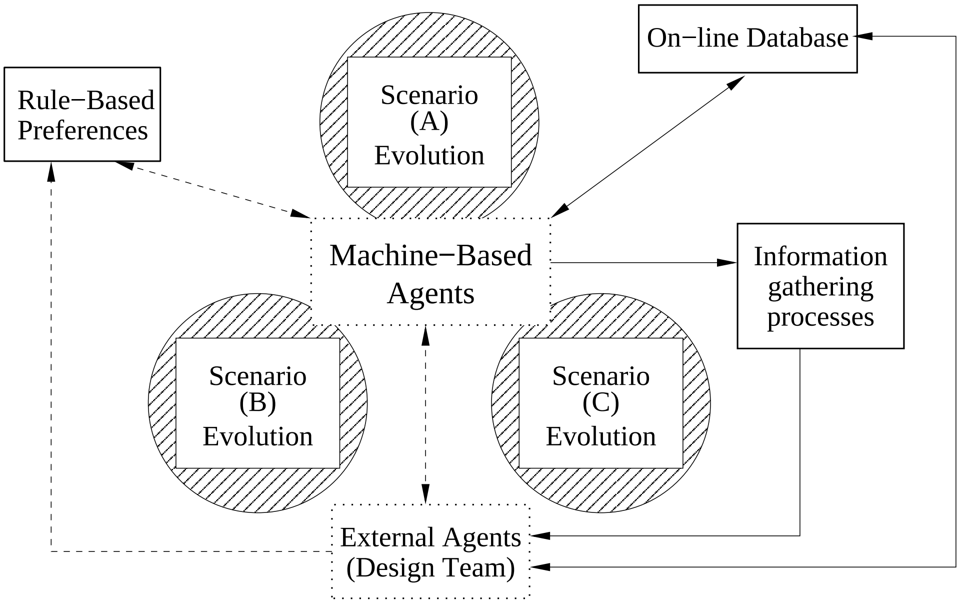
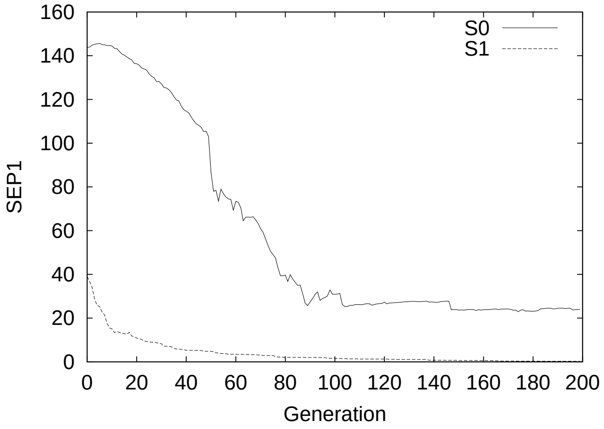
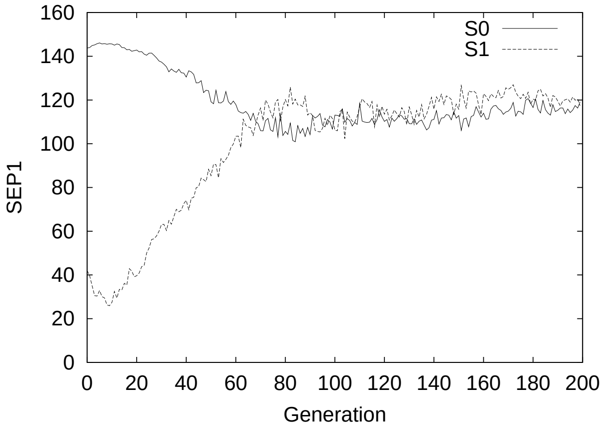


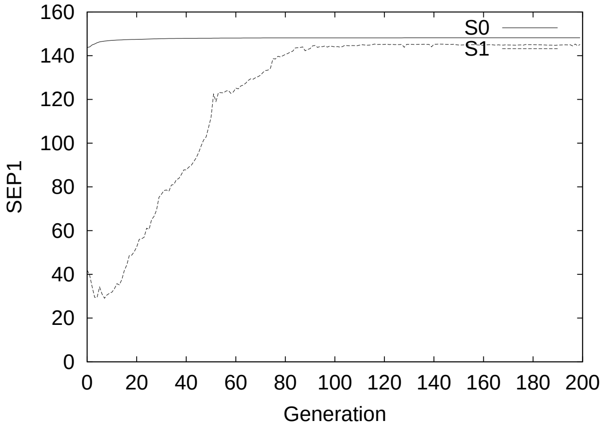
Figure 1: Schema of interactive evolutionary design system (IEDS).



(a) $y_3 \ll y_9$



(b) $y_3 \approx y_9$



(c) $y_3 \gg y_9$

Figure 2: Co–evolution with different preferences: The influence on optimising y_3 (SEP1, process S_0) and y_9 (FR, process S_1). Results for SEP1, average of 20 runs.

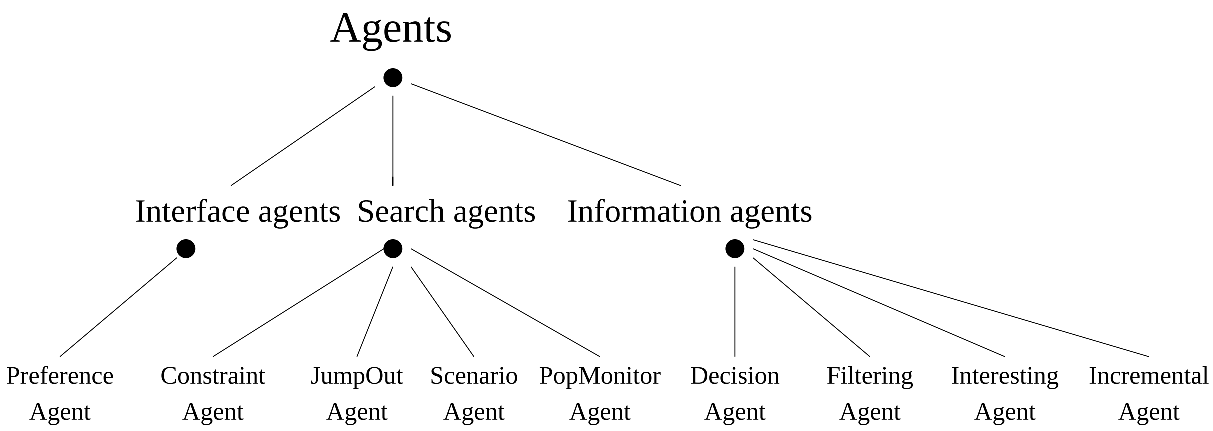
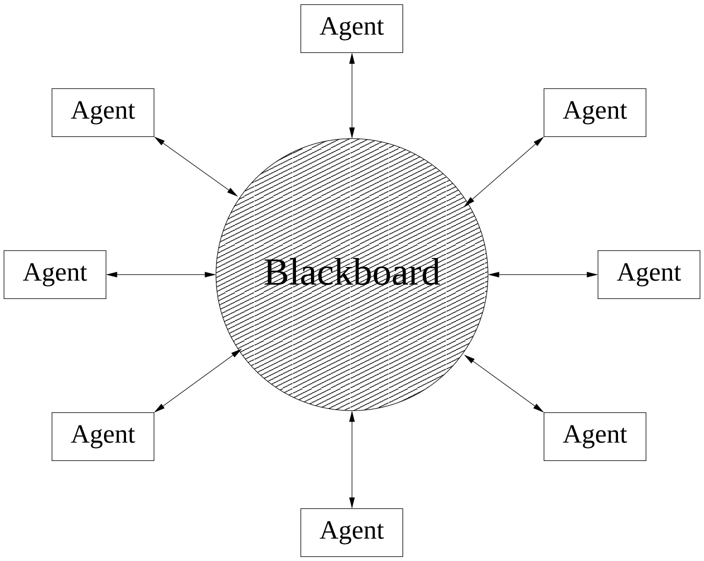
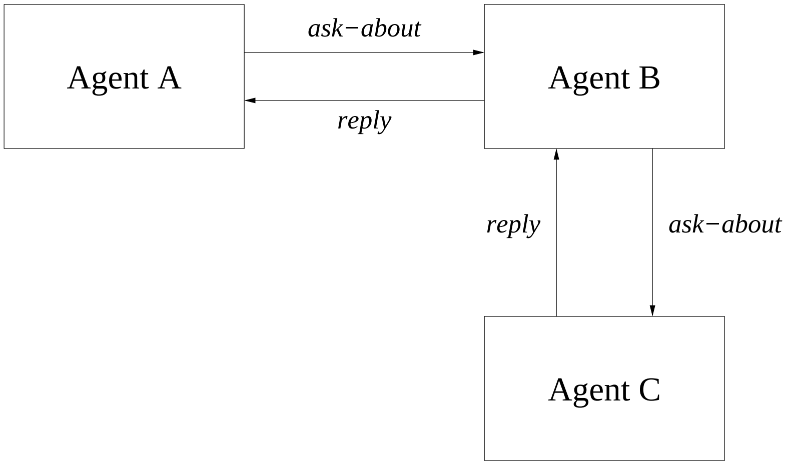


Figure 3: Agent classification and examples of agents within classes.



(a) Blackboard system



(b) Direct communication

Figure 4: Agents communication methods.

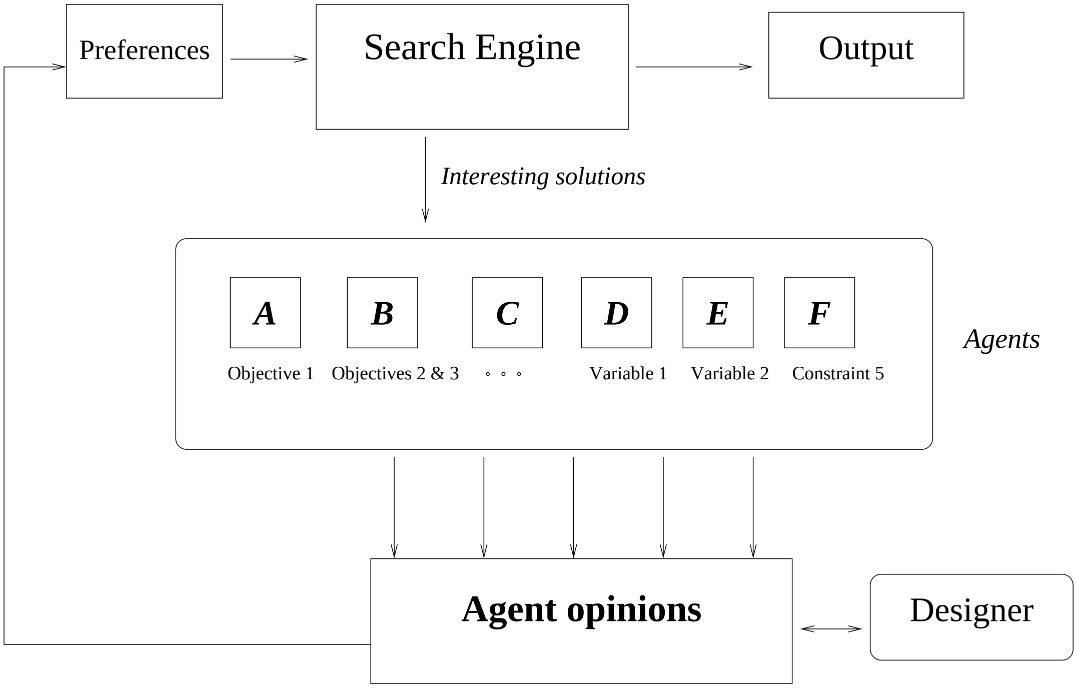


Figure 5: Agents in an optimisation context.

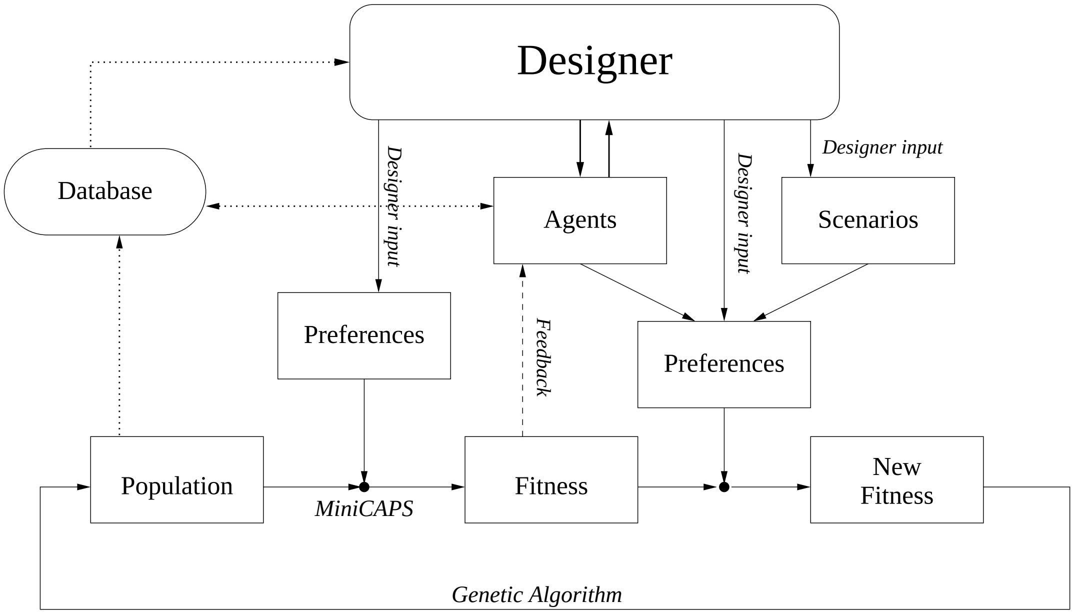


Figure 6: Closing design loop using agents. See text for details.