Swarm Visualiser - COS 301 Main Project
Functional Specification

Team: Dragon Brain
Members:
Matheu Botha u14284104
Renton McInytre u14312710
Emilio Singh u14006512
Gerard van Wyk u14101263

Contents

1 System Domain Model

In this section we will discuss and present a system-wide domain model and present the constituent systems, Modules, and their scopes

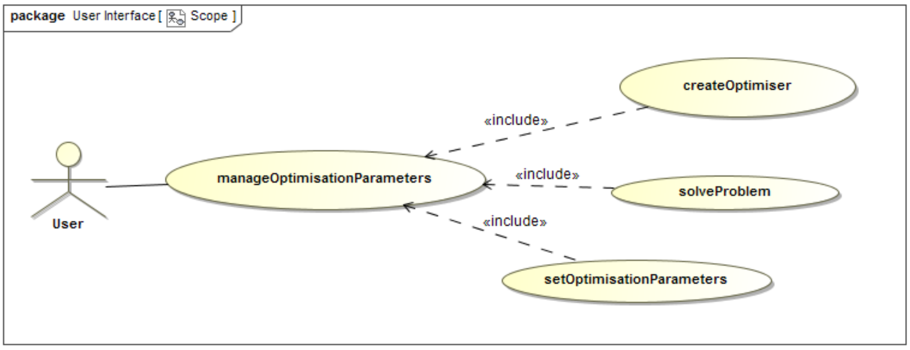
2 Modules

2.1 General Optimiser

The General Optimiser or OPT for short is the module that provides the problem solving capacities of the system. It makes use of swarm-based methods to traverse user defined search spaces and perform evaluations of particles within the search space against user defined objective functions.

2.1.1 Scope

The scope of the General Optimiser is presented below.



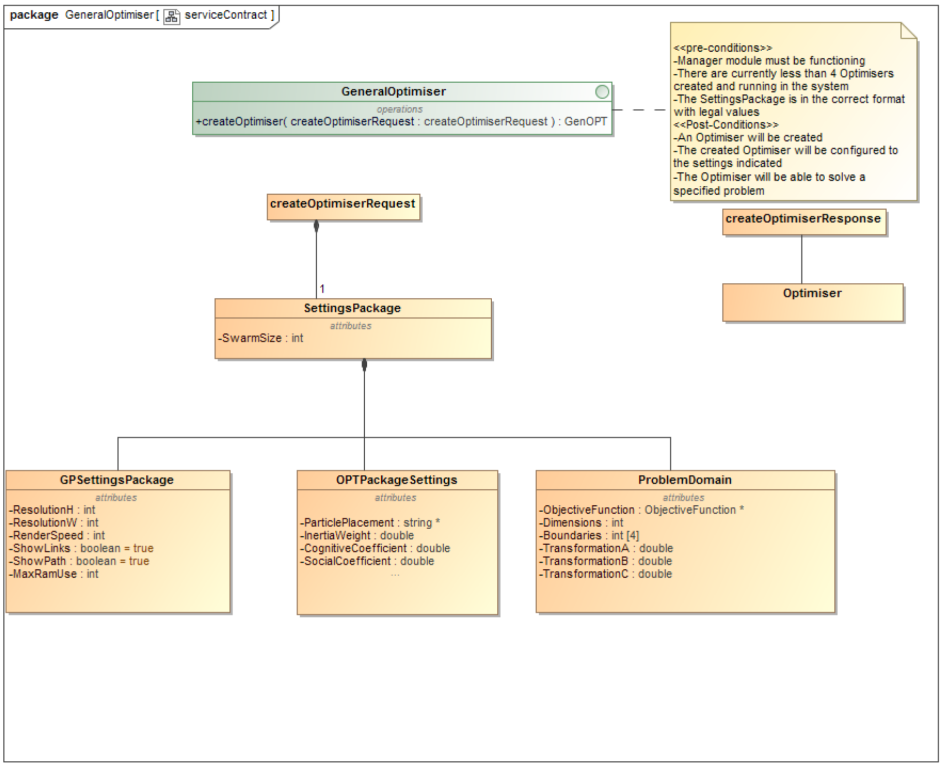
The User has 3 areas of general purpose use

- Creating Settings Packages which configure the Optimiser for runs
- Changing Optimiser parameters during a run
- Presenting problems for the Optimiser to solve

2.1.2 Service Contracts

We will now specify the service contracts that importantly define the capacities of the General Optimiser Module

2.1.3 Create Optimiser



The user will request for an Optimiser to be created. If there is space, that is less than 4 Optimisers already exist in the System, then the user will construct, indirectly, a SettingsPackage by entering their configuration parameters for the Optimiser. This Settings Package will be used by the Manager to construct and configure an Optimiser for use.

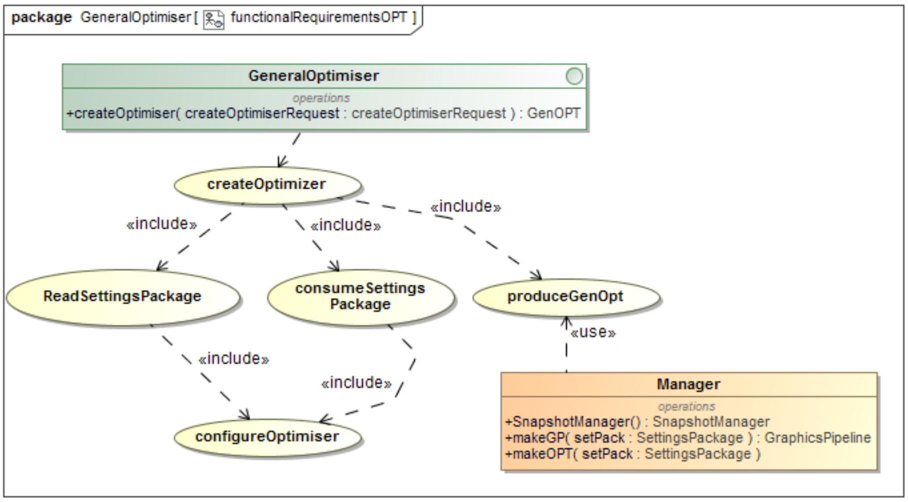
The exact conditions will be codified below:

1. There must be fewer than 4 Optimisers currently running in the system

2. The SettingsPackage received must contain specifications for all configuration parameters and must contain values for those parameters within legal domains.

3. The Manager Module is ready to receive/able to receive a new order/request from the User Interface.

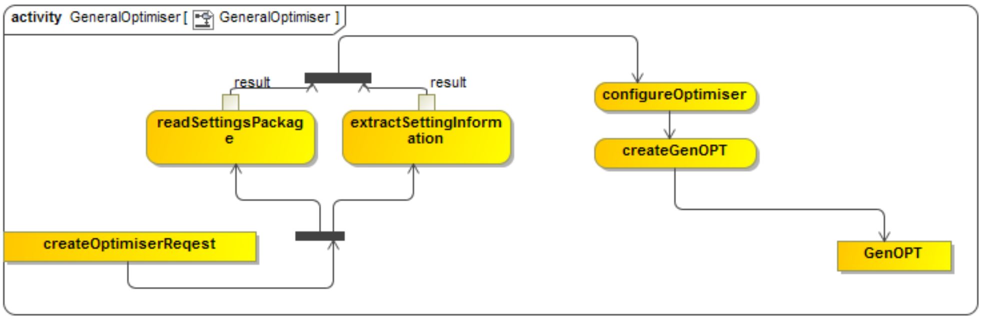
2.1.4 Functional Requirements



In terms of the functional requirements for the creation of an Optimiser object, a number of factors need to be considered in order for this to realised. Firstly, the reading and consumption of a Settings Package object relates to the process of extracting the necessary information from the settings package, the GenOPT package components, in order to provide the information during a constructor call. During the process, the Settings Package received is now invalidated and must be destroyed as it is at the end of its life-cycle. Once configuration has been completed, the GenOpt object will be produced and used by the Manager object.

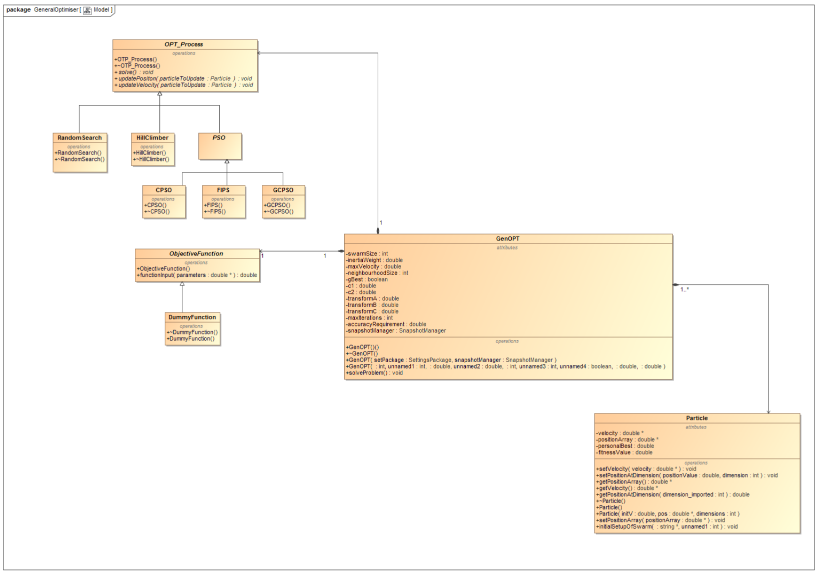
2.1.5 Process Design

The process specification defined here is for the process to create an Optimiser object.



2.1.6 Domain Model

Presented below is the domain model of the General Optimiser.

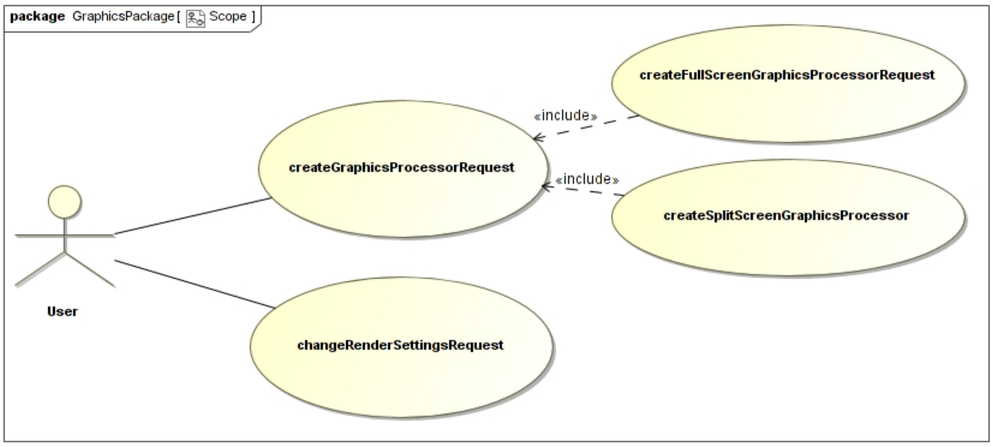


2.2 Graphics Processor

The Graphics Processor is responsible for visualising the data that is generated by the Optimiser.

2.2.1 Scope

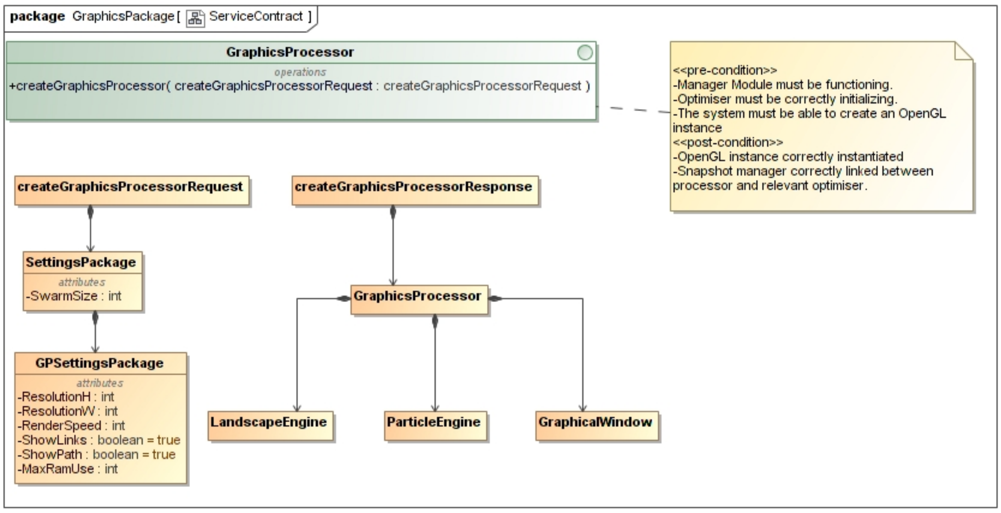
The scope for the Graphics Processor is presented below:



The User has 2 areas of general purpose use

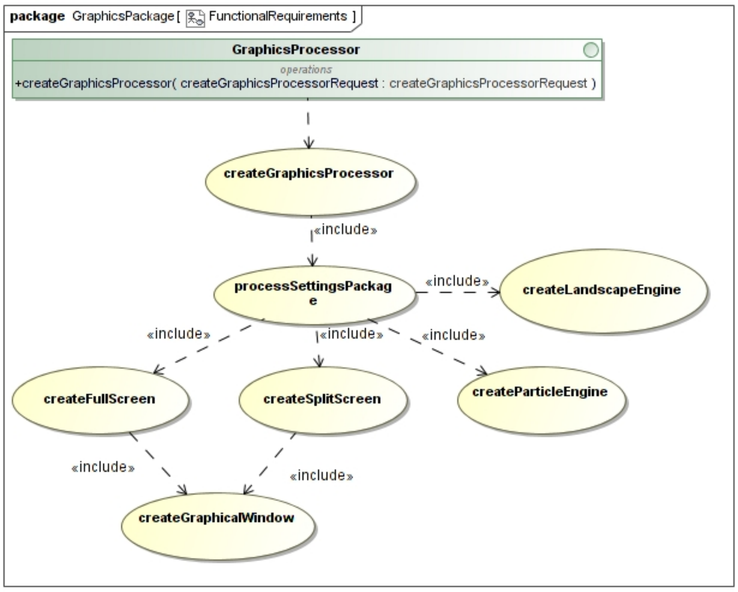
- Creating a Graphics processor to visualize a swarm which can be done in to ways:
  - Full screen mode for a single swarm visualiser.
  - Split screen mode for multiple swarms.
- Change the render settings

2.2.2 Service Contracts



The service contract for the graphics processor facilitates the creation of the processor. A graphics processor as a whole is a single entity which can then be associated with a optimiser. Once associated with that optimiser in the manager, the data from the optimiser will be visualised.

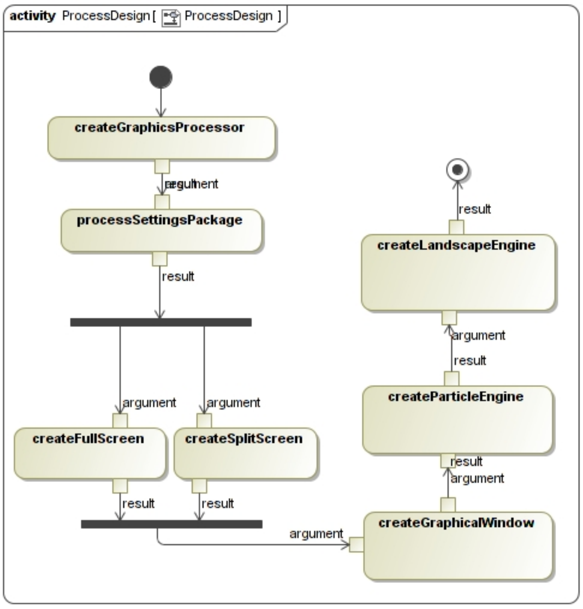
2.2.3 Functional Requirements



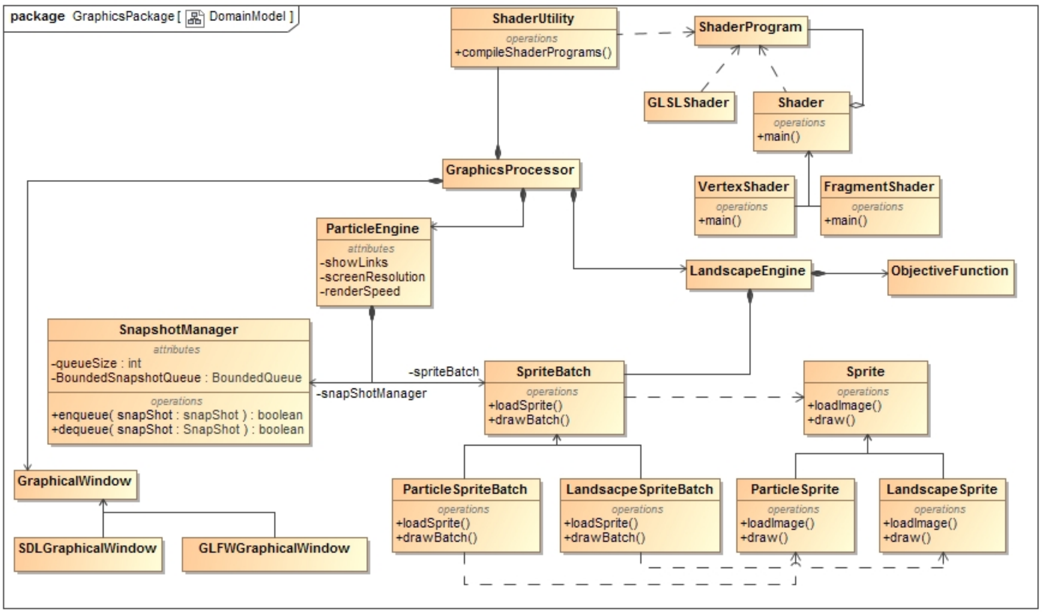
The functional requirements for creating a graphics processor are defined as follows: It will recieve a settings package. Determine how many optimizers will be running which will inform it whether or not it needs a split screen functionality. And then simply pass the render settings to the shaders.

2.2.4 Process Design

The following is the process design for creating a graphical processor:

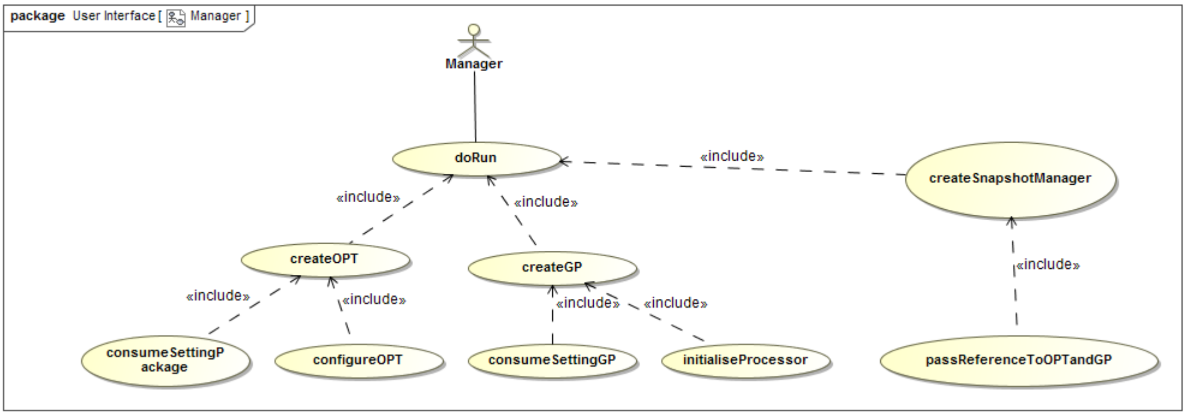


2.2.5 DomainModel



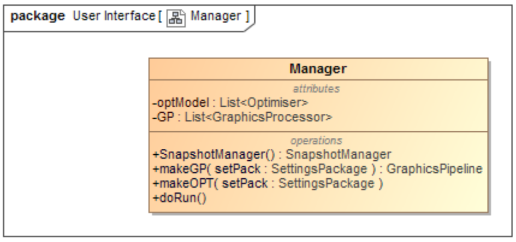
2.3 Manager

2.3.1 Scope



The scope of the Manager model can be defined in the context of a single doRun operation. A run is defined as the interaction between an optimiser and a graphics process mediated by the snapshot manager serving as the intermediate storage unit. The doRun operation will create the requisite components who will then in turn perform their operation once notified by their partners in the communication.

2.3.2 Domain Model

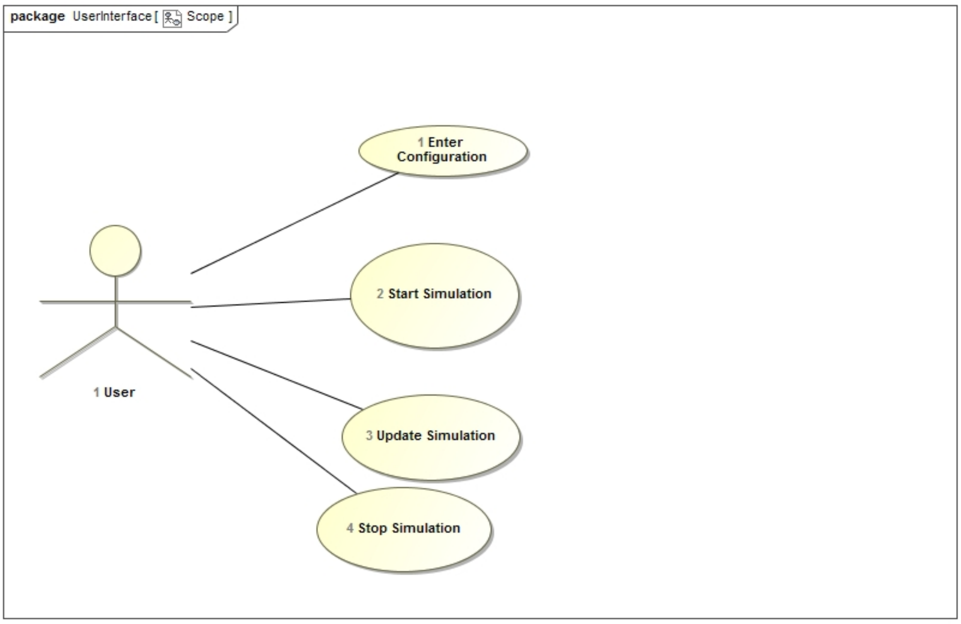


The Manager class encapsulates the core functioning integration unit of the system. The Manager class will contain references to specific Optimisers and Graphics Processors and Snapshot Managers. One set of the three components comprises the components needed to perform a run. Each of these components are tethered, or coupled, to each other for the duration of the run and cannot be uncoupled. This will end when the run is terminated with the components being destroyed and the resources allocated available for reuse after the fact.

2.4 User Interface

2.4.1 Scope

The scope of the User Interface is presented below.



The User has 4 main capabilities

- Changing the parameters of run environment
- Starting the simulation with the currently configured parameters.
- Stopping the current simulation.

2.4.2 Functional Requirements

In terms of the Functional Requirements of the User Interface, the focus is as follows:

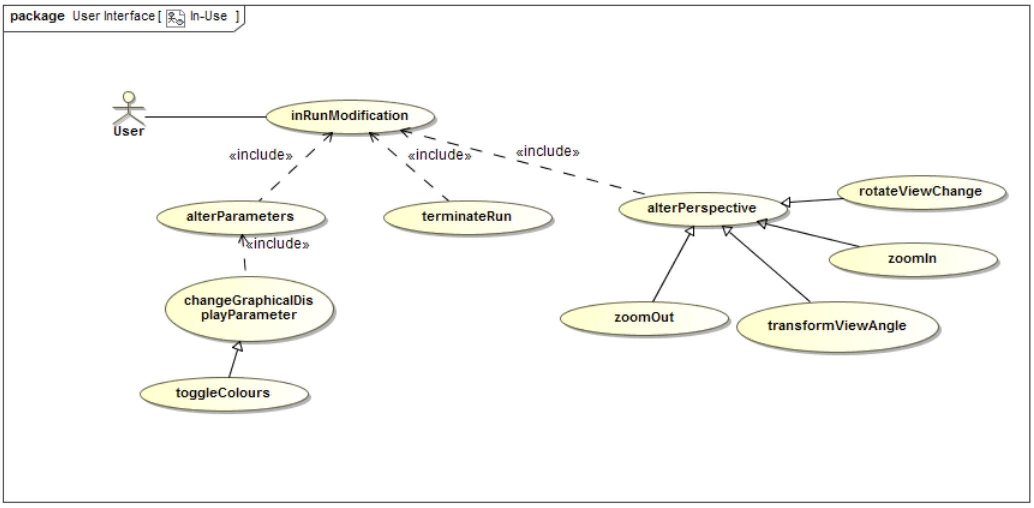
- The user must be able to configure parameters for the system via the GUI.
- The GUI should be capable of sending data to the SettingsPackage in order to generate a SettingsPackage.
- The system should generate a SettingsPackage and start the simulation upon user request.

3 System Use Cases

3.1 System Usage

3.1.1 In-Run

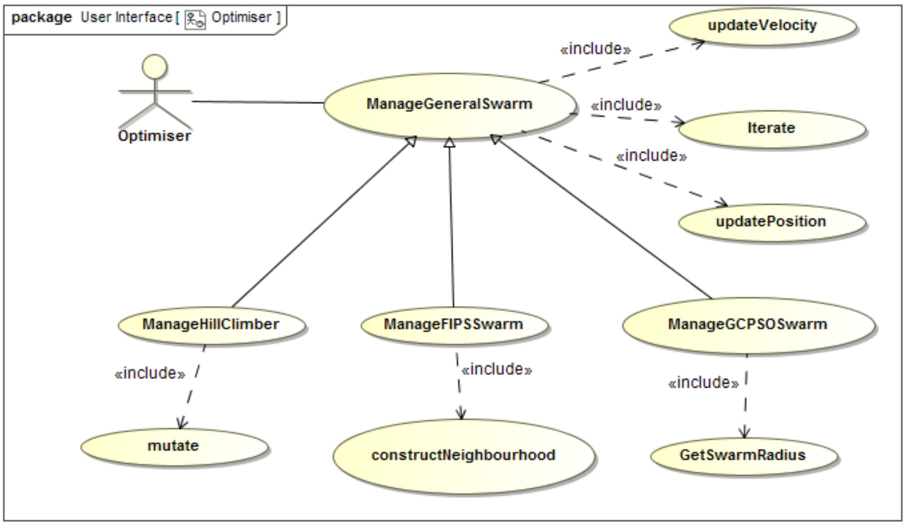
The following use case diagram highlights the user actions during the system operation



3.2 System Components

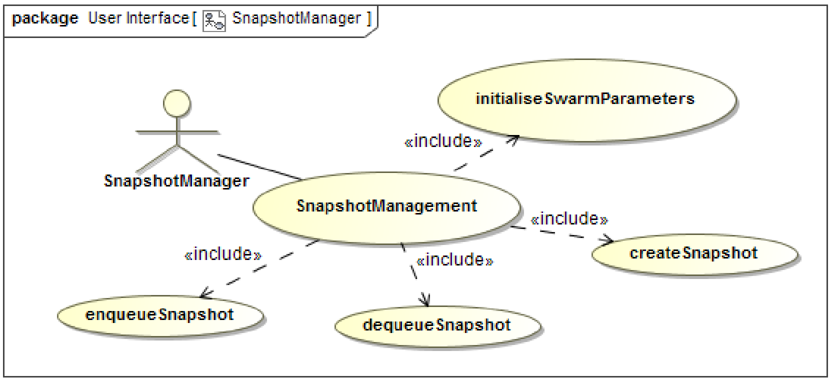
3.2.1 Optimiser

The use cases for the Optimiser component are represented below. Of particular interest is the generalisation hierarchy that exists from the ManageGeneralSwarm capacity. This hierarchy is extended to enable additional behavioural changes to the swarm at run time by polymorphism.



3.2.2 Snapshot Manager

The use cases for the Snapshot Manager component are represented below.



4 Algorithm Functional Specification

4.1 Algorithm Service Contracts

This project relies very heavily on its algorithms and their complexity is amongst the most difficult to implement and difficult to understand components of the project as a whole. Therefore, this section will deal with explaining how each of the algorithms and their components function as well codify the behaviour, functioning and operation of the algorithms in use in the system.

4.2 Algorithms

4.2.1 Operations and Limitations

To clarify, we will define some terms below

- Swarm- A collection of particles that is used to solve an optimisation problem
- Particle- A single worker unit of the Swarm that contains a position reference and solution variables to the problem being solved.
- Run - A run refers to a single span of time that starts when an optimiser is configured by the user and started. The run ends when the user decides to terminate the visualisation or the stopping conditions of the optimiser bring it to an end.
- Iteration - An iteration is a single unit of work for an optimiser. An iteration consists of updating all of the particles in the swarm in terms of their positions and velocities and determining if an ideal solution has been worked out yet.
- Optimiser - A Swarm based manager component that contains a swarm, a problem and will then make use of iterations in order to produce solutions to the given problem
- Velocity - A rate of change used by the particle to quantify how far per iteration it is able to move. This can change and is influenced by many factors like other particles or previous histories.
- Position - A set of parameters that indicate where in the problem domain the particle is in terms of mapping a given set of parameters to a given result in the problem space based on the function given.
- Stopping Conditions- These are specific conditions with regards to system parameters that when reached, are used to terminate an optimiser's functioning and stop it from continuing indefinitely. Examples include
  - Maximum Iterations Reached

– Sufficient solution accuracy reached
– Insufficient change of particle positions
– Approximate solution found
– Premature Halt signalled by operator

Specific Functioning Limitations
The project offers functionality in terms of user interaction in two areas

```
1) Configuring an Optimiser Run

1. The system functioning takes place in terms of the user using the GUI to config
2. Once configured, the user will then start the run which will begin the optimisa

1. The user can create and have up to 4 simultaneous runs at a given time dependin

2) Viewing and Changing the Visualisation

1. Once the user has started the visualisation process, they can perform the follo

1. Change viewing angle -This will happen by rotating the viewing plane or shiftin

2. Stop the run and terminate the optimiser in progress

There are some limitations to what the SwarmViz project can do.

1. It can only visualise 2D and 1D problems those being problems with 1 or 2 dimen
2. It cannot in real time change the function being used
3. Arbitrary functions cannot be inserted into the system during run time. New fun
```

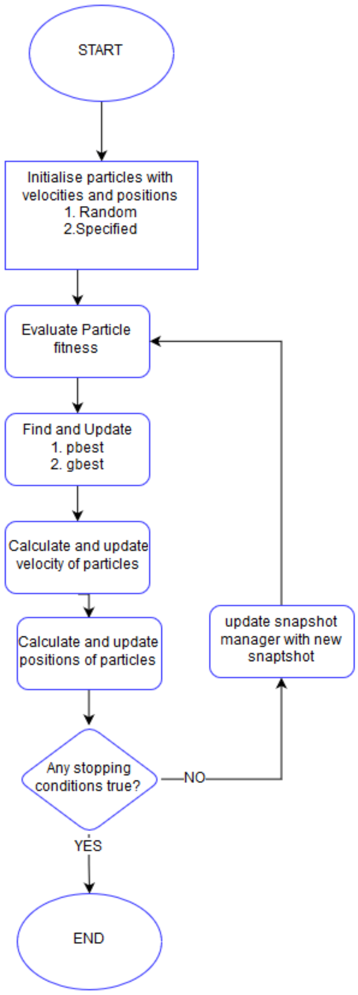
In general, the algorithm for an Optimiser is presented below.

1. Start of Run
2. Initialise Particles in Swarm with:
   (a) Positions
      - Random Positions within Domain
      - Specified Positions by user

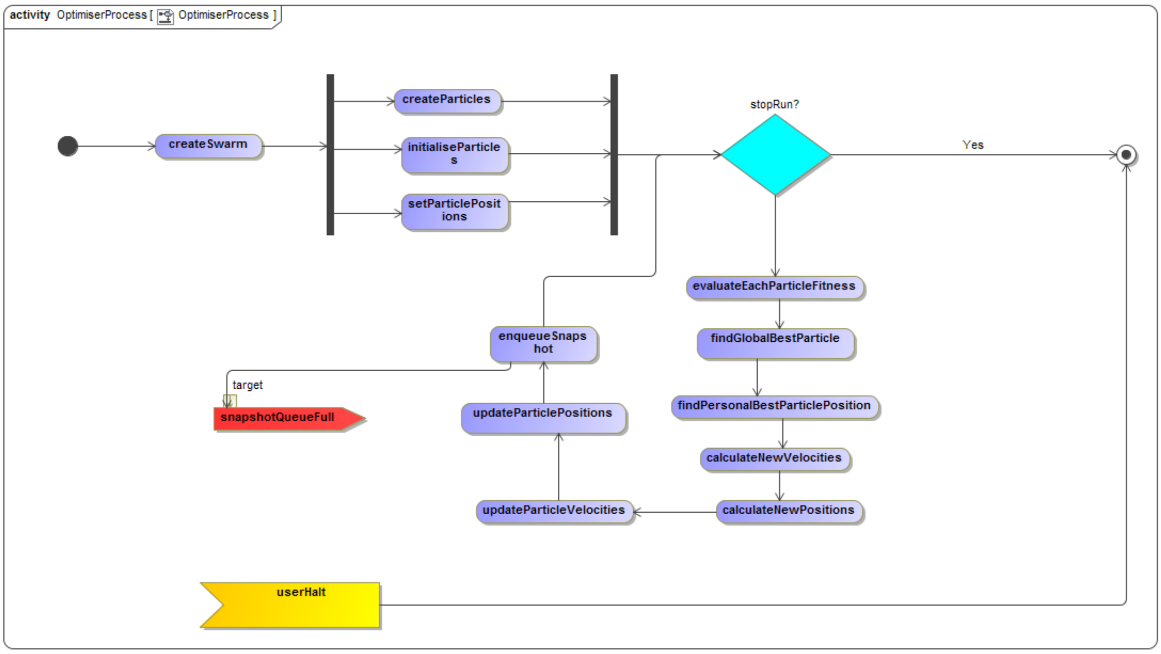
    (b) Velocities as per random distribution

3. Evaluate particle fitness of each particle in Swarm
    (a) Fitness=fitnessFunctionEvaluation(Particle[x])
4. Find and update best history and global best
5. Calculate and update each particle's new velocity
6. Update position of all particles by new velocity
7. Are any stopping conditions satisfied?
    (a) If so, HALT
    (b) Otherwise go back to 3
        i. Output graphical updates to Snapshot Manager
8. Run Terminated

This process is also represented in the following flowchart



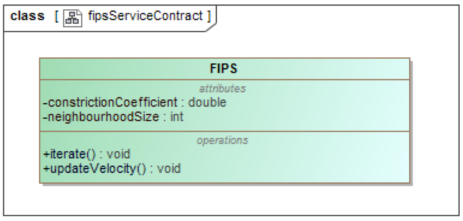
This process is also represented in the following activity diagram



Below are the algorithm specifics pertinent to each of the Optimiser algorithms currently implemented.

4.3 FIPS: Fully Informed Particle Swarm Optimisation

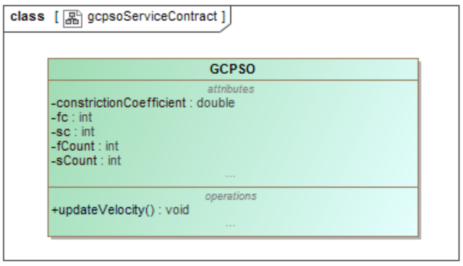
The point of the FIPS adaption of the standard PSO algorithm is to increase the reliability of the results. How this works, is it will increase the influence capacity of other particles to, based on a scalar k, influence others by increasing the influence capacity good nearby particles to influence the particle being considered. The effect of this is that, based on considering size of the scalar value, particles generally avoid worse positions. This comes at a greater cost to the performance of this especially since large capacity swarms could potentially have large neighbourhoods for potential consideration requiring increasingly more run time computation. One of the great benefits to this, is its capacity to be used for problems that involve clustering of data sets.



4.4 GCPSO: Guaranteed Convergence PSO

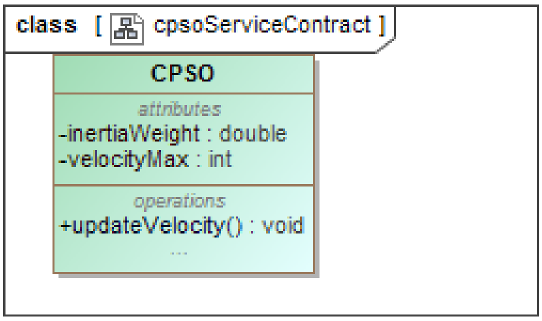
The reasoning behind the GCPSO is to provide an additional capacity for the swarm to prevent itself from converging on a poor position, a minima when there are potentially better positions within range of the swarm. To do this, it makes use of an additional search particle that exists contiguous to the swarm but not as a part of the swarm.

Once the swarm has potentially converged, the additional search particle, called Beta for short, will then attempt to within the maximum potential radius of the swarm, find a better position. If it is able to do so, the convergence is halted and the swarm will continue to iterate. If not, it will more reliably guarantee that the position that the swarm converged to is the best one within its capacity to reach.



4.5 CPSO: Conical Particle Swarm Optimisation

The CPSO is the least variant of the traditional PSO algorithms. The only significant changes are a modified update velocity function which makes use of velocity clamping and inertia weights to aid in convergence actions.



4.6 Hill-Climber Approach

The Hill-Climber Approach is a non-PSO approach. It makes of a swarm but none of the particles have any capacity to influence each other. Rather,each particle only has one capacity, namely, to only go in directions that result in a better position. In this way, each particle only moves to better positions but each particle is isolated from the next so finding the best position is difficult and largely determined by the maximum bounding of the domain.