1 High Level Design

1.1 Layers

The Clusterer is organized in three (very) separate layers, each layer has some clear responsability and communication between layers is strictly limited (i.e. there is exactly one point of communication in each layer that can communicate either with the next layer up or the previous one down). The three layers are:

1. **Main Application** : This layer contains all the application specific classes, this includes concrete instances of the abstract Distributable Jobs (which encapsulates a non atomic work) as well as all data specific classes (including Data description and handling). It is important to note that only code written for this layer would eventually have to be changed if one were to shift the problem domain to something completely different. In this application you have three distinct Distributable Job instances:

   RepresentativesJob : Create a set of representatives (call it R) with the following properties:

   - $\forall x, y \in R : x \neq y \Rightarrow distance(x, y) > cutoff$
   - $\forall x \in Structures : \exists r \in R : distance(x, r) \leq cutoff$

   ClustersJob : Create a partition on $Structures$, say $P \in Structures^N$ with the following properties:

   - $\forall S \in P : !\exists r \in R : r \in S$
   - $\forall r \in R : !\exists S \in P : r \in S$
   - $\forall S \in P, s \in S : \exists r, r' \in R : r \in S \wedge r \neq r' \Rightarrow distance(s, r) \leq distance(s, r')$

   CentersJob : From each cluster, assign the center to it as the closest structure to the geometric mean of the cluster. If we assume that this center is $c \in C$, where C is a particular cluster ($C \in P$), then we have:

   - $\forall s \in C, c' \in Centers : c \neq c' \Rightarrow distance(s, c) \leq distance(s, c')$

   In layer 1 you can also find the definition of a Structre (principal chain of atoms in a given protein) and all classes encompasing the handling of a Structre Data Base.

2. **Work Distribution Management** : All Distributable Jobs will eventually separate themselves in a number a Job Units (an atomic work unit). It is the responsability of this layer to handle the assignment of these units to a specific processing client (just Client from now on) and later inform the results thus obtained. From this layer there is no relevance as to where Clients come from or are located (and can be therefore implemented using a variety of multiprocessing paradigms such as processes in a single core, threads of some sort, BOINC clients, MPI, etc.). Also, the type of problem to be solved is irrelevant from this layer's point of view. This layer would be subsequently modified if one were to modify the distribution policies of Job Units. At this point, you have the abstract definition of a *DistributableJob* and the *Distributer* itself, which will work as the JobUnit scheduler of the system.

3. **Client Management** : Responsabilities of the layer include handling client registration and status throughout the application's runtime. The layer does not care about specific problem details or the Job Units themselvs, it will only function as a bridge between single Clients and the Job Units sent from layer 2 classes. Code in this layer can thus be modified if one wishes to change the implementation of how specific Clients are handled or where they exist. Since in this specific application Clients happen to be based on the volunteer computing model of BOINC, you can find interfaces that communicate with a BOINC server at this stage.

Outside the layers you can find alternate libraries implementing standard containers (used thourghout the application) or even application specific ones such as GROMACS.

Within this layer view, it could be argued that layer 3 alone constitutes a framework for dealing with BOINC clients, layers 2 and 3 taken together form a framework for generic problem solving using BOINC and all three layers taken as one single piece of software create the clustering application.