Chapter 7

CiClops - Collaborative Laboratory

> *"I abhor averages. I like the individual case. A man may have six meals one day and none the next, making an average of three meals per day, but that is not a good way to live." — Louis D. Brandeis*

CiClops (Computational Intelligence Collaborative Laboratory Of Pantological Software), still in its early stages of development, was initially designed to address the scalability limitations of the CILib simulator discussed in the previous chapter, by storing simulation results in a structured database and distributing simulation workloads over a cluster of workstations. Further, CiClops is intended to facilitate empirical studies by maintaining a repository of past simulation data and providing statistical analysis tools. The following high level goals have been identified for CiClops:

- **Scalability:** The CiClops framework should support an arbitrary number of samples per experiment and enable those experiments to be clustered over multiple workstations.
- **Simulation repository:** CI simulations can be very computationally intensive, sometimes requiring days to complete an experiment, even scaled across a cluster of machines. Complete simulation results should be stored in a shared repository, so that existing simulation data can be used as a basis for future comparisons without the need to perform expensive re-computations. Further, the simulation data should keep track of its dependencies on code and data sets, so that if any dependencies change then the results can be recalculated to ensure their correctness.

- **Statistical analysis tools:** The majority of researchers (90% in one study [16]) apply inappropriate parametric tests without first considering whether the assumptions on which those tests are based are satisfied [65]. Further, it has been empirically shown that these assumptions typically do not hold [81, 20]. Thus, CiClops should implement and provide decision support for sound statistical hypothesis testing, so that researchers without the necessary statistical background can reliably perform statistical testing without making errors. It would also be convenient if built-in tools could be used for visualising data in various ways.

- **Ease of use:** CiClops should provide an intuitive GUI, which facilitates experimentation with different parameters and algorithmic configurations.

- **Security:** A granular permission system is required to ensure that, while simulation results and configurations should be sharable, they can also be kept private whenever necessary. For example, it may be desirable to keep results private while working on a competitive publication. In addition, a full audit trail should be maintained to discourage misbehaviour and ensure the integrity of results in circumstances where permissions are permissive. Further, since the services provided by CiClops may have a salable value, only authorised users should be granted any access at all.

- **Revenue stream:** As discussed in Section 4.5, means of turning CiClops into a revenue generating resource should be investigated.

The following section gives a general overview of the CiClops architecture and Section 7.2 reviews its underlying data model. The software component responsible for executing units of work on each node of a cluster is discussed in Section 7.3. Next, the CiClops client interface is covered in Section 7.4. Finally, the current status of CiClops is discussed in Section 7.5

7.1 Architectural Overview

As shown in Figure 7.1, CiClops is implemented using the J2EE framework (refer to Section 5.3) and consists of three essential components:

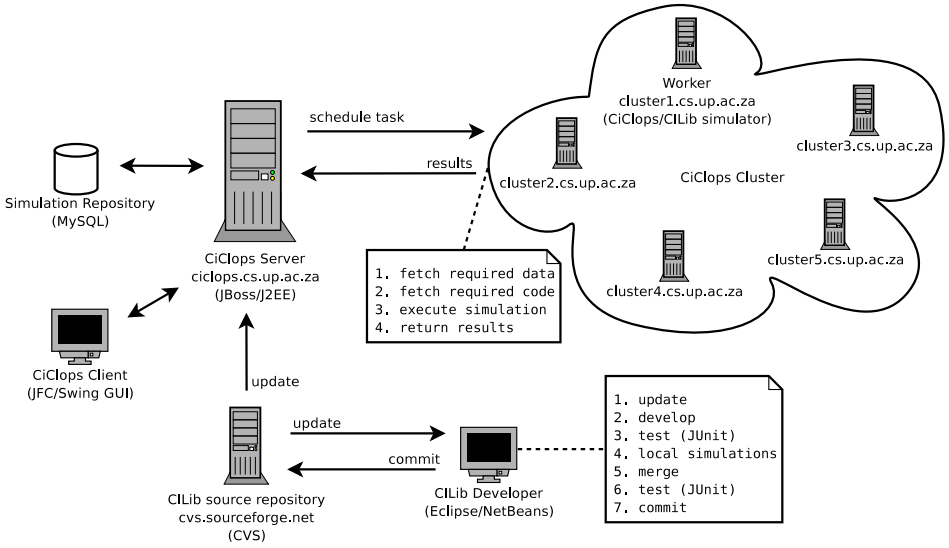

Figure 7.1: CiClops Overview

- **The CILib code base:** CILib forms the most important component, since it is used to conduct the actual simulations. The only change to CILib is the addition of a different simulator, which executes only a single sample at a time, sending the results to the CiClops server instead of writing them to a local file. Note, CiClops periodically (or at the express demand of a user) updates its version of CILib according the version stored in the CVS repository at SourceForge, so care should be taken by developers not to break it, which is why testing is emphasised in the diagram. The CVS code must be kept in a pristine state. Developers must ensure that they update their local version of the code, merge any conflicts with the CVS repository and run local test simulations before committing any changes. If sufficient unit tests are provided to perform proper regression testing, then few problems should be experienced with this approach. Alternatively, CiClops will need to implement different namespaces for code used by different developers, which would inhibit collaboration by spawning multiple versions of the code base.

- **A cluster of workstations:** Each cluster node, or worker, consists of a light weight stub which executes tasks, taking the form of CILib simulations, on behalf of the CiClops server. Workers always execute simulations using the latest available version of the CILib classes and any data sets by means of remote class loading

and efficient local caching of data sets.

- **A central server and data store:** The CiClops server is implemented as a J2EE application and deployed on the open source JBoss application server. The back-end data store is a MySQL relational database, although the J2EE persistence framework makes this largely irrelevant to the application, affecting only the deployment descriptor, which is generated automatically using XDoclet (refer to Section 5.4). The server is responsible for configuring experiments, scheduling tasks on the cluster, archiving simulation results and performing statistical analysis on the results. The load balancing services provided by the J2EE container (refer to Section 5.3.2) means that CiClops can also be scaled up to multiple servers if and when the load of many workers becomes too high for one server to handle.

Finally, some kind of user interface is required to interact with the system. Presently, this is provided in the form of a rich JFC/Swing based GUI client (refer to Section 7.4), with a view to providing a web based front end in the future. Fortunately, this should not be difficult to accomplish, since all the CiClops application logic is executed on the server, lying within the application tier of the J2EE framework.

7.2 Data Model

The data model, or persistence tier, of CiClops is implemented exclusively using CMP entity beans (refer to Section 5.3.1). Figure 7.2 illustrates the object relational mapping employed by CiClops using private attributes, however, it should noted that those private fields do not physically exist and were provided for the sole purpose of making the diagram more readable, since they do at least exist conceptually.

The central concept in the data model is that of a simulation, which is characterised by its name, a description, an XML configuration for CILib and the number of times the experiment represented by this configuration should be repeated, or simply the number of samples. For each sample, a simulation stores the results for each measurement, which are serialised using a CILib domain (refer to Section 6.2.1) and then compressed to save on database space. The domain string is also stored with each measurement so that CiClops is able to deserialise it again, using the CILib domain classes via a *Proxy* (refer to Section 3.2.5) that makes use of the Java reflection API.

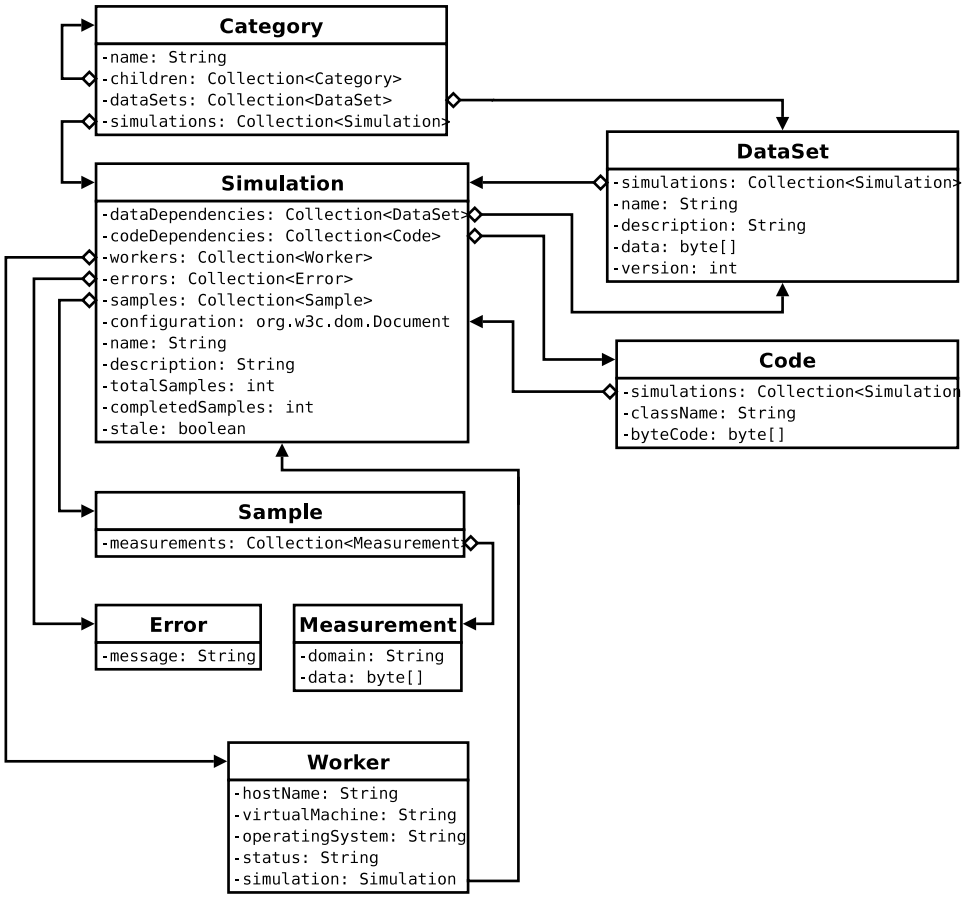

Figure 7.2: CiClops Data Model

Further, a simulation keeps track of its dependencies on particular CILib classes and data sets, or conversely, code and data set entities keep track of the simulations which are dependent on them. Whenever, a class or data set becomes modified they can *Iterate* (refer to Section 3.3.2) over their respective collections of simulations marking each simulation as stale and as a consequence a candidate for rescheduling whenever the cluster is idle. Fortunately, constraints on the data model like these can be isolated in the persistence tier and as such no error in application logic can ever cause a class or data set to become modified without their dependent simulations being marked as stale, particularly considering the transaction isolation provided by the container (refer to Section 5.3.4).

A simulation is scheduled over multiple workers and any errors, in terms of exceptions thrown, experienced by a worker are stored for that simulation to be later examined by

the user. Finally, simulations and data sets are organised into a hierarchical name space, which is imposed by named categories.

7.3 Workers

The data model in the previous section implies that the smallest unit of work that can be scheduled to a worker is a single sample. Experiments should always be sampled at least 30 times [106], meaning that even a single experiment should be able to saturate a cluster of 30 workstations. Currently, the CIRG@UP has fewer than 30 dedicated machines at its disposal and the default number of samples is set to 100 to provide for more robust statistical analysis that may be accomplished using larger samples. Further, it is expected that many different experiments will be configured simultaneously, possibly even by multiple users, enabling CiClops to saturate even hundreds of cluster workstations with this simple scheduling policy. Further parallelism can only be achieved by implementing much more complex scheduling rules, which would require cluster aware algorithms in CILib and incur significantly higher network communication overheads. Responsibilities of workers include:

- **Remote class loading:** Most of the worker logic is implemented in the CiClops simulator, which is actually component of CILib. In fact, the worker part of CiClops consists of little more than a remote class loader, which overrides the standard Java class loader, and a *Proxy* (refer to Section 3.2.5), which is used to fire up the simulator using the reflection API and pass it the XML configuration for a simulation. Thus, code that runs on the cluster is stored and executed from a central location, where it can be upgraded to add new features at any time, without ever modifying the configuration of a workstation.

- **Fetching and caching data sets:** Data sets used in simulations can be very large, and in order to save network bandwidth it makes sense to cache as many as possible data sets on the cluster workstations. Each worker checks the version of any locally cached data set against the server before every simulation and updates its local copy if the versions do not match. The CiClops simulator exposes data sets using the same `net.sourceforge.cilib.Problem.DataSet` interface as the

standard CILib simulator does (refer to Section 6.2.2), so clients do not need to treat remotely loaded data sets any differently.

- **Serialisation and compression of results:** Measurements are serialised using the CILib domain classes (refer to Section 6.2.1) and compressed using the standard Java `java.util.zip.GZipOutputSteam` output stream *Decorator* (refer to Section 3.2.3), providing a relatively good trade off between compression ratio and speed, before being sent back to the CiClops server for storage. Compressing the results on the workstation means that compression load is also distributed across the cluster and further network resources are spared.

7.4 Client

The CiClops client, which as far a possible conforms to the MVC architectural pattern mentioned in Section 5.3.3, currently only supports the configuration of simulations, exporting of their results for external processing and monitoring of the cluster progress.

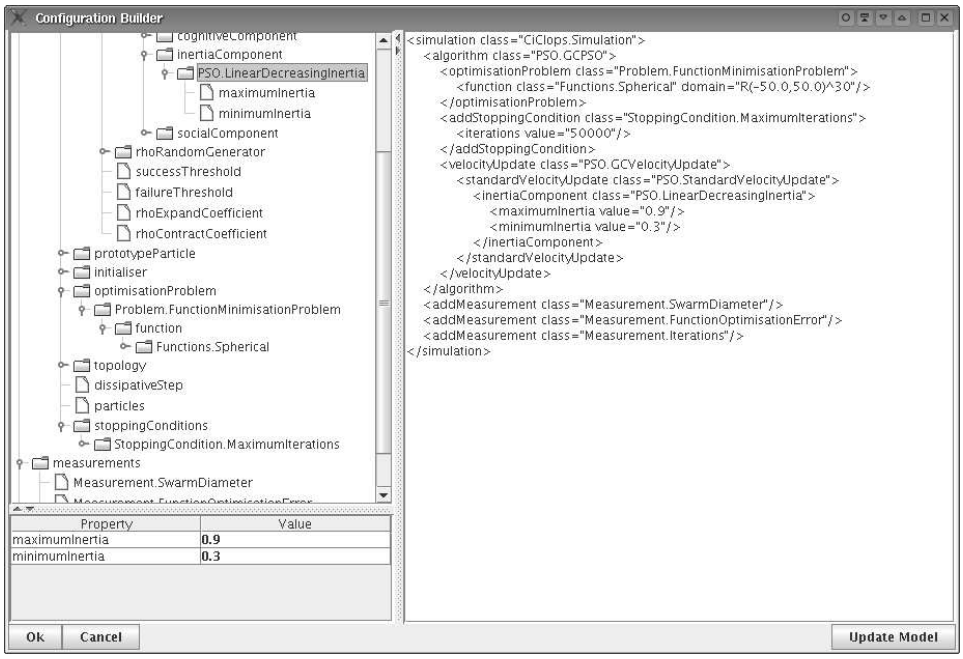

Figure 7.3: Configuring a CILib simulation using CiClops

Figure 7.3 is a screen-shot of the CiClops client being used to build an XML con-

figuration for a CILib simulation. The user may choose to edit the XML configuration directly, however, the hierarchical view of classes and the property editor promote discoverability of CILib features, which the user would otherwise have had to consult the CILib API documentation to learn about. Since the textual, hierarchical and property views all make use of the same model, a combination of these mechanisms can be used simultaneously to edit the configuration. The XML document is validated by the CiClops server against a dynamic schema (refer to Section 5.1.2) which reflects the classes stored in the database.

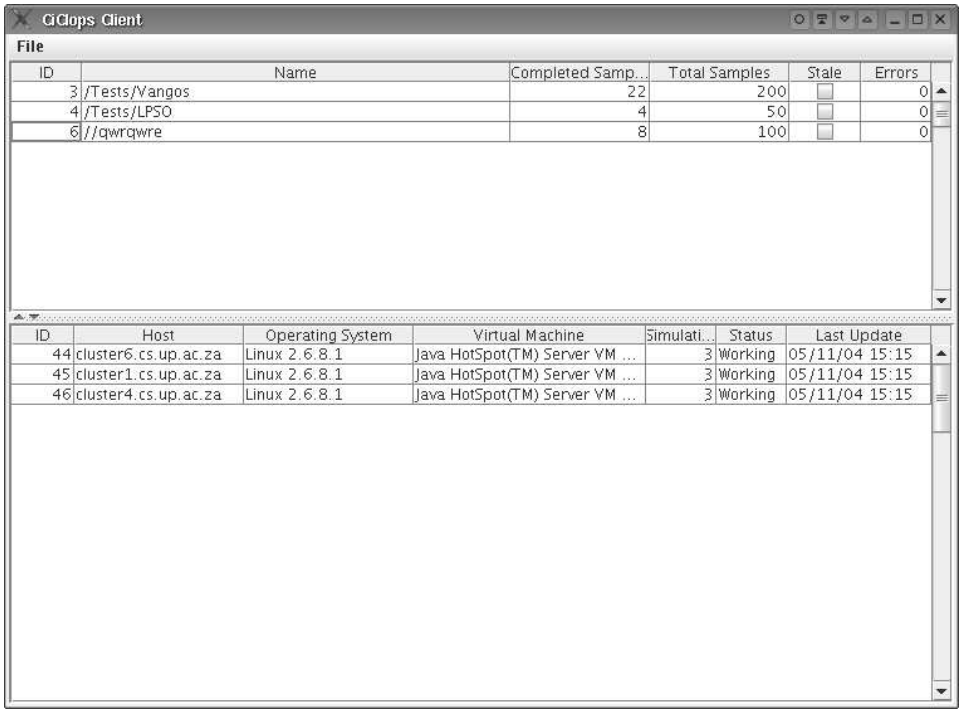

Figure 7.4: CiClops monitoring CILib simulations

Figure 7.4 is another screen-shot taken of the CiClops cluster monitoring view. The figure shows three test simulations (indicated in the top pane) being executed on a small cluster of workstations (indicated in the bottom pane).

7.5 Status

As stated at the beginning of the chapter, CiClops is at an early stage of its development. Custodianship of the source code has recently been handed over to the CIRG@UP and the group as a whole will be continuing its development.

Many of the design goals have already been met, including solving the CILib scalability issue, maintenance of a simulation data repository and the provision of an easy to use mechanism for configuring simulations, by means of a hierarchical GUI builder.

The J2EE declarative security model using XDoclet tags (refer to Sections 5.3.4 and 5.4) presents some challenges. For example, the fact that the security permissions do not appear anywhere, except in the deployment descriptor, means that there is no way for a GUI client to query the security model in order to determine whether or not to present a specific option to a user, without resorting to custom security code. The use of code annotations, which can be queried using the reflection API as provided in the recent Java 1.5 release, for declaring security permissions may provide a solution to this problem, but still needs to be investigated.

Further, statistical analysis methods still need to be adequately investigated. Instead of implementing all the required functionality in-house, it may be better to draw on other software such as the tools available from the Java numerics project[1].

Finally, the CIRG@UP still needs to decide how best to market CiClops to the broader research community, while maximising collaborative and profit opportunities.

[1] http://math.nist.gov/javanumerics/