Packaging simulation results with CostGlue *

Matevz Pustišek[#], Dragan Savić[#], Francesco Potortì°

[#] University of Ljubljana (SI), email : matevz.pustisek@fe.uni-lj.si, dragan.savic@ltfe.org

° ISTI-CNR, Pisa (IT), email : Potorti@isti.cnr.it

Abstract: Researchers performing simulations in the field of computer telecommunications are often faced with the time-consuming task of converting huge quantities of data to and from different formats. We examine some of the requirements of the telecommunications simulation community and propose an architecture for a general purpose archiver and converter for big quantities of simulation data to be released as free software.

Introduction

We describe the motivation, design issues and the approach to the implementation of a low-overhead software package that can import simulation or measurement results into a common data structure, launch post-processing applications on the imported data and store data or export it into various output formats.

Modern simulation packages and statistical tools use different, even proprietary formats for the results. This can be a major obstacle to the efficient exchange of scientific data. Moreover, if we consider issues like storage and data management, documentation and meta description, analysis and display or data filtering, a need for a common format or a tool to handle simulation results becomes apparent. This topic has been addressed [1] in the framework of the European COST 285 Action "Modelling and Simulation Tools for Research in Emerging Multi-service Telecommunications"[1], a forum where researchers from all around Europe periodically meet to address issues related to simulation of communications systems. The point made was that apparently no general purpose tools exist for exchanging big quantities of simulation data coming from different sources in different formats. Not only the need of a common format for exchanging data was highlighted, but also the need of feeding this data to different tools for postprocessing them, each requiring a different input format.

To better understand the scope of different requirements we define a reference model that encompasses data creation, flows and processing in the analysis of the telecommunications systems by simulation. The main functional parts composing such a model (simulators, data collectors, graphing tools, statistical tools) are covered by the many existing tools that are used by the research community; we focus on the input data in form of simulation, like ns-2 [2] traces, or measurements, like Tcpdump [3] traces. Raw data can be post-processed (e.g. calculating average packet delay) and the results stored separately from the raw data or complementing it, so that further analysis is possible based on both raw and preprocessed data. Finally, the results can be exported in various more or less widespread output formats, like ASCII or XML [4].

* This work was funded by the European Commission under the COST 285 Action, by the CNR/MIUR program "Legge 449/97" (project IS-Manet) and by the Ministry of Higher Education, Science and Technology of the Republic of Slovenia (program P2-0246).

[1] COST stands for COoperation in the field of Scientific and Technical research, see <http://www.cost285.itu.edu.tr/> for information on COST 285.

Data storage is based on the HDF5 [5] data format, which was selected after an analysis of the available options. HDF5 has been successfully applied in several scientific projects; it enables efficient data storage and lookup. Among the features most relevant to our purpose, it provides support for extremely large quantities of data, for meta descriptors and for embedded compression. A set of programming libraries is available in C and Python, which simplifies software development based on HDF5.

The proposed CostGlue software architecture is modular, to make it possible to include the future development contributions from other research communities. It is composed of three building blocks. The core is a Python application called CoreGlue, which provides HDF5 database connection, a simple command line, an HTML based interface and basic functions for import, filter and export. It also controls and executes other parts of the application. Specific functions, such as conversion from specific data format and calculations, are implemented as sets of self-descriptive loadable modules. A graphical user interface is envisaged via the use of a web browser, allowing user friendly and efficient remote access to the application.

CostGlue will be released as free software. The next sections describe into deeper detail the proposed architecture of CostGlue and particularly of its core, CoreGlue.

1. A model for the simulation process

In order to obtain a general and systematic overview of data creation, flow and processing, we define a reference model for the simulation process, which is depicted in Figure 1. The model provides a layered decomposition of main functions that are usually encountered when using simulation as a research method. There are three layers in the model.

The *source layer* provides raw simulation output, describing a simulation run into the smallest detail. Usually one or more records for every event during the simulation run are created at the source layer. Structure and volume of the data at this point is entirely dependent on the simulator. Most frequently it is in the form of large tabular traces, in ASCII or binary format. An optional source recoding sublayer handles the raw source data: its main purpose is to convert between different formats (e.g. from ASCII to binary or vice versa), data compression (e.g. Gzip [6], Bzip2 [7]) or removing private information, e.g. real IP addresses, from simulation traces. The source layer supports both raw simulation data and real measurements data. In fact, during our discussions, we found out that nearly the same model can be applied to the analysis of real network traffic traces. In this case the raw data is not a result of a simulation, but for example, data traces captured in a network link. Apart from the different tool (traffic capture tool, like Tcpdump or Ethereal [8] instead of a simulator) that generates raw data, all the functions of the upper layers remain the same in both cases.

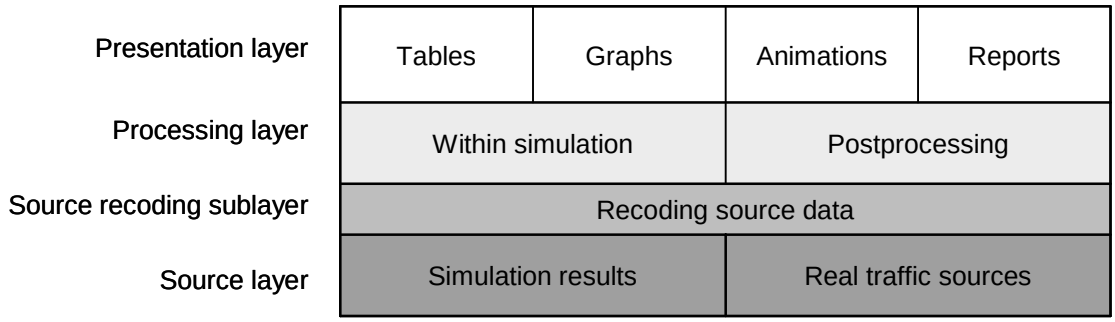

Figure 1: Reference model of a simulation process

The *processing layer* is responsible for simulation data analysis. At this level cumulative results can be derived from raw data (e.g. mean packet delay is calculated) or statistical confidence of the results determined (and consequently additional simulation runs conducted). An important characteristic of the data processing layer is that the amount of data received from the source layer is usually much larger that the amount of results of post-processing.

The *presentation layer* is the final stage where the results are organized in a form useful for communicating the most important findings with other interested practitioners. In case of simple tabular printouts of the results, this layer is void or only does trivial modification of the data (e.g. changes in number formats, column spacing). But frequently data is shown in 2- or 3-dimensional graphs (e.g. being part of scientific reports or research papers, web pages) or even presented in animated form (e.g. nam - Network AniMator [9]). At this layer the predominant requirement is flexibility of presentation and a possibility to create new or modified presentation objects from new or changed simulation results without reformatting.

We can map the functionality of particular tools used in simulation to the layers of our model. Usually, a single tool provides more than one functional layer or even all of them. In the most favorable situation, it would encompass all the functions needed and implement them adequately to meet all the researcher's needs. In practice this occurs very rarely and there is usually a set of complementing tools applied to cover the required scope of functions within the model. The selection of tools is made on arbitrary conditions, including their capabilities and performance, researchers' past experience with a particular tool or availability of tools. In case of simulated results, raw data can be generated by discrete event simulators (e.g. ns-2, Opnet [10]). Raw data can be captured in real networks, with sniffers, such as Tcpdump or Etheral. Source recoding can be done with small dedicated tools (e.g. Gzip, Tcpdpriv [11]) or different proprietary shell scripts. Data analysis can be performed with generic tools for mathematical computation (e.g. Octave [12], Matlab [13], Mathematica [14], Excel [15]), special statistical tools (SPSS [16], R [17]) or proprietary and dedicated programs or scripts. Often, the simulation package provides the functionality for data processing and analysis and it is up to the researcher to decide whether this is adequate or if an additional more powerful and flexible tool should be used. Other than in dedicated graphing or animation tools, presentation ability can be provided in generic mathematical tools and sometimes even simulators.

2. Formats for exchanging data

CostGlue acts as a central repository for data generated by various different simulation programs and as a converter both from several output formats and to several input formats. Therefore, it is important to know what programs and formats are generally used by telecommunications systems practitioners. To this end, we used the information that we got from the COST 285 participants, representatives from more than ten European nations, about the kind of tools they use for their simulation work. From this sample, we learned that no single simulation tool has a dominant position, but there is a great variety of used tools.

Among those that use a predefined format output, the most used appears to be the network simulator ns-2. Among the generic tools for mathematical computation that are used to run simulations, Matlab appears to be used by many. A large part of the simulators is composed by standard scripting or programming languages and generally by ad hoc simulators. On the side of tools used for postprocessing or graphing, there is an even greater variety. These observations, while limited in scope, show that some sort of tabular ASCII format is of generally common use, and thus being able to read and write ASCII tabular data is certainly a requisite for our proposed archiver and converter. But the variety of tools used also calls for a general way of reading and writing many formats: that's why we consider the modular architecture of CostGlue a necessary feature for the tool to be useful at all.

Another interesting point is that simulation data and measurement data have a lot in common, and a tool useful for one can be useful for the other. However, measurements are often output in particular formats, and an input converter is very frequently needed. An interesting feature that can be made part of CoreGlue is the ability to give a similar treatment to data coming either from measurement or from simulation of the same environment, and archive them in the same format. This is the reason why the first prototype of the CostGlue will include the ability read ns-2 data and Tcpdump data, store them into a common format and write in both formats. This capability would make it easy to use the many tools available that are able to analyze and graph data obtained by both ns-2 and Tcpdump.

All the above discussion leads to a scenario where a simulation tool is run several times, each time producing tabular data, that is, data that can be conveniently stored into a two-dimensional structure

having relatively few columns and a possibly huge number of rows. What about data that cannot be naturally converted to a two-dimensional format? In this case, the inner structure of the archived data needs to be different. One of the challenges of the proposed tool is to being efficient in the most common case of collections of tabular data, but still be useful in the case of non-tabular data.

3. The database

A common file format solves several problems regarding the exchange of the simulation data. Therefore we made a thorough analysis of different data formats and their corresponding libraries for data manipulation. Among many we have focused on the following set of data formats: HDF4, HDF5, netCDF, ODB, FITS and OpenDX [18]. Beside these, we also considered using plain text formats, XML and SQL databases. The results of the analysis makes it clear that the HDF5 file format is the most suitable for this task since it meets all the requirements of data organization e.g., separation of raw data and metadata and different requirements of contemporary computer system architectures, such as managing big quantities of data, offering a general data model, supporting complex data structures, portability among different computer platforms, parallel data access and processing, diversity of physical file storage media, etc.

3.1 The database structure

Summarizing what we said above, most simulation data in the computer communications area are collections of tables of numeric data: each simulation run generates a table of data having few columns and a possible huge number of rows. Each table is associated with certain parameters that are specific for a simulation run that generated it, and is uniquely identified by the values of those parameters. We are interested in defining a database structure that is able to efficiently accommodate this type of data. To meet these requirements we have chosen to build a database in HDF5.

HDF5 is a data format and an associated software library consisting of two primary objects: dataset and group. A dataset represents a multidimensional array of data elements, which can hold different types of data. The data stored in datasets can be either homogenous (only one data type used within single dataset – simple dataset) or compound (different number of data types within one dataset – compound dataset). Since tabular data collected from certain simulators often contains data with different types (e.g. integer, float, char), we use compound datasets to accommodate the nature of simulation outputs. An HDF5 group is a structure containing zero or more HDF5 objects. By using two primary HDF5 objects, data can be organized hierarchically by means of a tree structure where an arbitrary number of HDF5 objects are derived from the main “root” group. Groups and datasets have a logical counterpart in directories and files in a hierarchical file system and, similarly to a file, one can refer to an object in an HDF5 file by its full path name.

To meet the requirements of effective data storage, especially those that are critical to management, understanding and reuse of scientific data, each HDF5 object may have associated metadata stored in the HDF5 file – referred as archive – in a simple attributes form. Attributes usually represent a small dataset connected to a certain group or a dataset. Their purpose is to describe the nature and/or the intended usage of the object they are attached to.

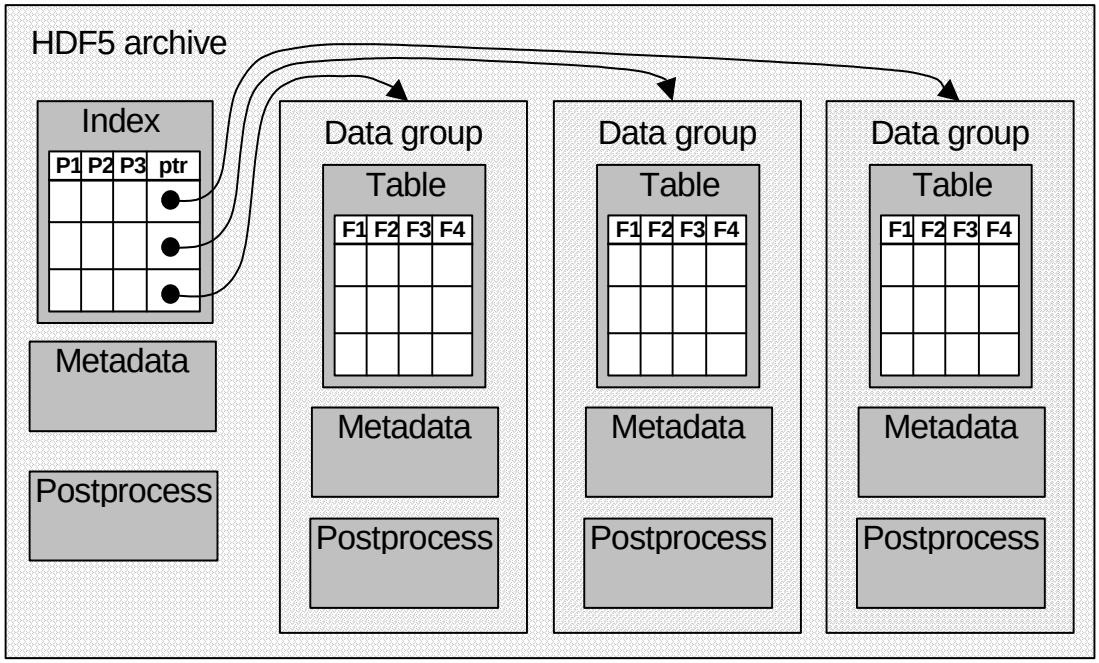

Figure 2: The logical structure of the database

In the design of the database structure our goal was a flexible representation of the stored simulation data by using one multidimensional array, where a user can easily extract a desirable portion of the simulation data. Even though HDF5 supports multidimensional arrays, it is not efficient to store huge amount of data in just one array. Furthermore, due to the HDF5 primary aspect of use, which is to have data organized hierarchically, storing everything inside a single multidimensional array means losing on the side of flexibility. We then introduced an indexing table which maps the logical view of a multidimensional matrix into an HDF5 hierarchical structure as detailed in the following.

Figure 3 presents a detailed overview of the proposed database structure where the *indexing table* represents the logical part and all other groups and datasets represent actual data where the raw simulation data, metadata and postprocessing data is stored. The whole database is treated as one archive containing a “root” group from which all other groups and datasets form a two-level tree.

An immediate extension to having tables of numeric data is having tables of fixed-length data, which can be flags, numbers or strings. The PyTables library [19] allows efficient manipulation of 2-dimensional HDF5 compound datasets from Python referred to as *tables* from now on. Each table, together with metadata and postprocess data, is attached to a *data group*, which will usually holds the data produced during a single simulation run. Data groups are indexed by vectors of *parameters*. An *index* is a 2-dimensional array, referred above as the indexing table, where the parameters relative to each data group are stored: each column corresponds to a different parameter, and each row contains the values of the parameter relative to a data group. So an index is used as the data structure for accessing a data group using an array of parameter values relative to that data group as the key. To each data group a table is attached, where each row is filled with the values of the *fields*, each field corresponding to a column of the table.

The overall structure is then a collection of 2-dimensional tables indexed by arrays of P parameters. Overall, this can be logically seen as a matrix with P+2 dimensions, where the first P dimensions are sparse and the last 2 dimensions are dense. We define the first P indices as parameters that identify a data group. As for the last two indices, the first index represents the field in the data group's table, and the second index the row number of the data group's table.

Let us describe how the results of an example real-world simulation can be stored in the described structure. We are simulating the behavior of packet switches in ns-2; each run is characterized by

several parameters, such as architecture type, buffer size, number of I/O ports and traffic load. Each simulation run differs in at least one of these parameters. When storing the results of one simulation run in the archive, we populate a new row in the indexing table: values in each row uniquely identify a simulation run. Each row contains the full path to a data group, containing the table with the results of the simulation run, metadata and processing data (see Figure 3). Metadata stores information about the type of scripts used to generate that simulation run, type of network topology, traffic patterns, etc. Postprocessing data include packet loss probability, maximum, minimum and average packet delay. Metadata and processing data are also associated with the whole archive, and contain information relative to the whole set of simulation runs.

The CoreGlue manages the index and the database structure, including the tables. Modules are responsible for metadata and postprocess contents, both for the single data groups and for the whole archive. The CoreGlue and the modules together constitute the whole application, which is named CostGlue.

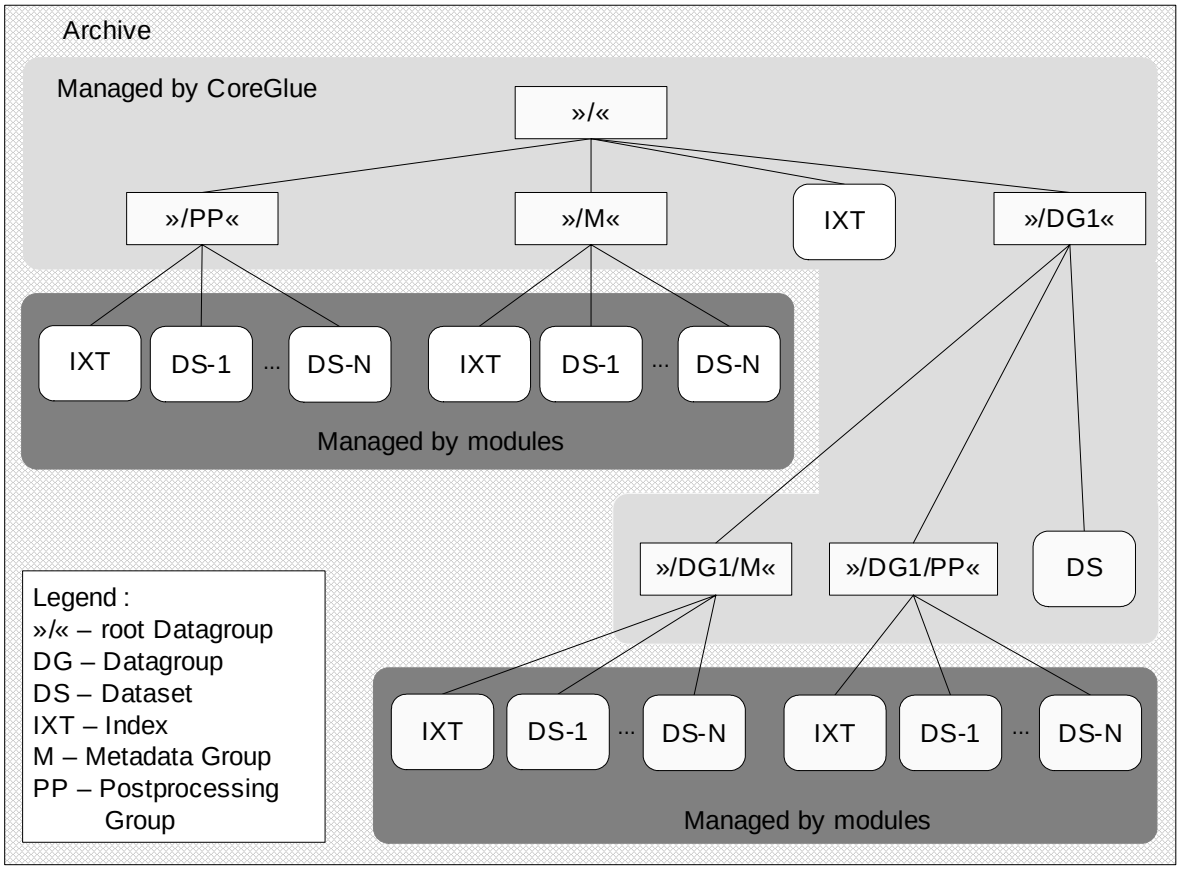

Figure 3: HDF5 database structure

4. CostGlue architecture

The CostGlue is being written in Python [20]. This language was chosen because of anecdotal evidence of its efficiency both in memory usage and processing power and for its programmability ease, due to automatic garbage collection and many native functions and types. Portability among operating systems is excellent and library availability for many tasks, especially mathematical ones, is rich. With respect to its main competitor, Java, Python has a generally smaller memory footprint and, being the implementation completely free, does not suffer being controlled by a single private entity.

The architecture of the CostGlue application can be seen in Figure 4. The CoreGlue connects other parts of the application by performing the following tasks: it reads module descriptions, executes modules, passes parameters, reads module execution results, handles the HDF5 database, provides a

CLI (Command Line Interface), provides an HTML interface, generates XML for additional user interface and takes care of exporting data in various formats (CSV, TXT …).

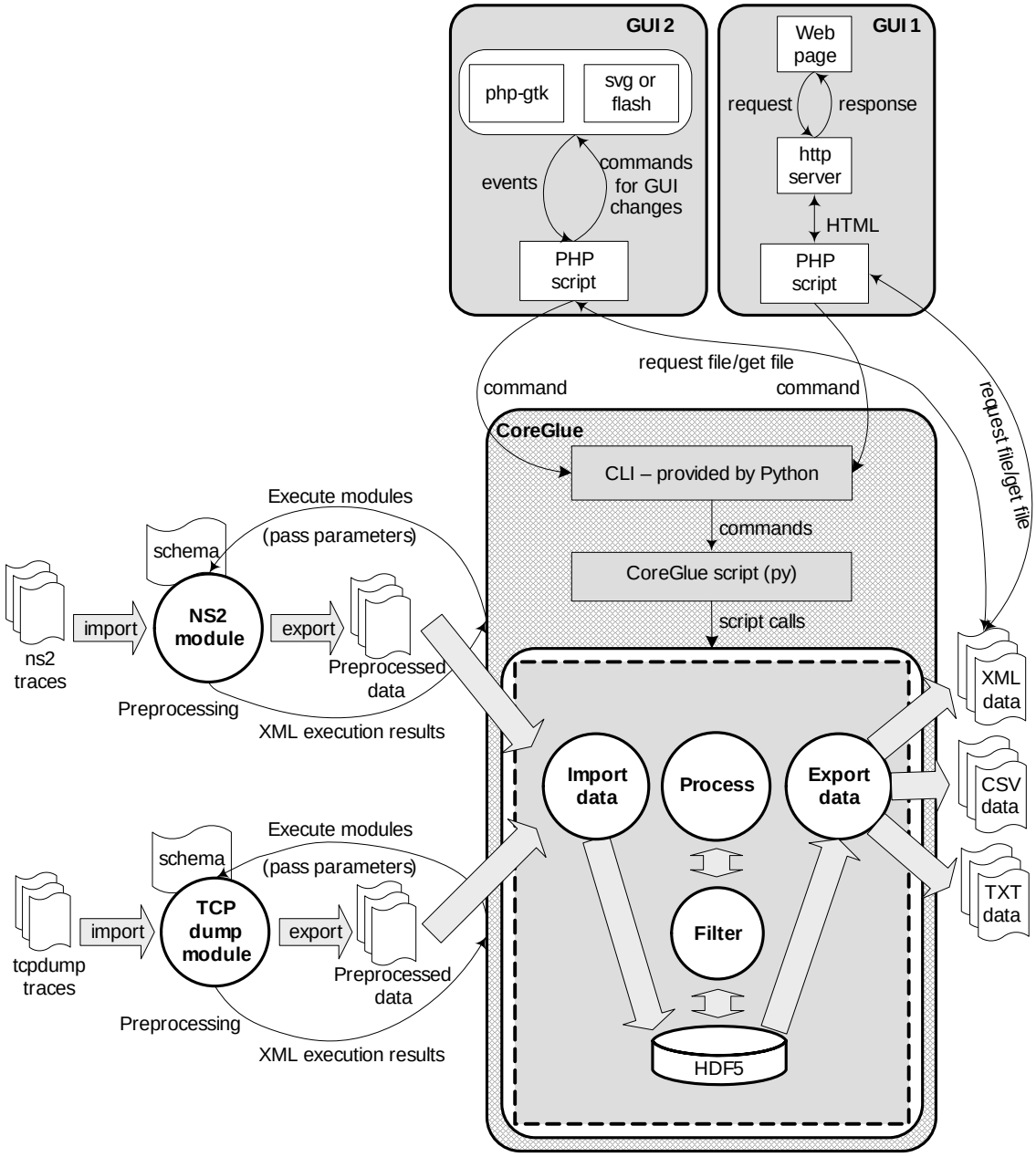

Figure 4: CostGlue software architecture

4.1 Overview of the module API

Python, like many modern languages, supports dynamic module loading. We exploit this possibility by providing a module library with a restricted pool of modules, and leaving the possibility open to other parties to write other modules. Modules are Python self-describing programs that reside in a fixed place. The CoreGlue can look for available modules and query them one by one in order to know their capabilities; this can be used for example for building a menu for a GUI (graphical user interface). Modules are able to describe the parameters they need and the type of output they produce. The needed parameters can then be provided to the module by the CoreGlue, with or without input from the user; in case input is required, CoreGlue can check that the parameters provided are consistent with the constraints defined by the modules. A GUI can also use this information to present the user with choices for the input parameters.

Modules interact with the CoreGlue through a well-defined API which contains a minimum of classes and methods for interacting with the database. The methods allow a module to manipulate the data group index in order to add or remove data groups, and to manipulate the data groups, in order to add or remove data, metadata and postprocess data to it.

The API also includes methods for accessing data with *selectors* written with the index notation that is widely used in matrix computations programs such as Matlab or Octave. In this notation, each of the P+2 indices can be "1", indicating the smallest index; "end", indicating the highest one; "n:m", indicating all the indices between n and m included; ":", indicating the whole range from smallest to highest, "n:s:m", indicating the range from n to m in steps of s; an array like "[1 5 6 8]" indicating the selected indices. Using the Matlab index notation one can take complex orthogonal slices of the multidimensional matrix composed of all the data in the database; in fact, the database structure can be seen as a sparse matrix with P+2 dimensions, and being able to take a slice of this matrix could prove a powerful feature of CostGlue.

4.2 The command line and the HTML GUI

The CoreGlue always needs to load a module to do anything useful. When invoked by a command line, its first argument is the module name, followed by the parameters needed by the module. When invoked with the `--html` option, the CoreGlue acts as an HTTP server, providing a graphical user interface. Through this interface, the user can look at the list of available modules and, for each of them, look at a description, at the input they require and at the output they provide. The CoreGlue provides an interface for the input of module parameters, complete with checking, thanks to the information that it reads from the modules themselves. In the simplest case, this is analogous to calling the module on the CoreGlue command line, but more convenient for interactive use. A module can also provide a graphical interface or graphical output by itself.

Conclusions

The purpose of the conversion and storage tool whose design we described in this paper is to facilitate the exchange and management of simulation data among researchers, and to ease the task of using different simulation, measurement, data processing and visualization tools. If this design will develop into a usable package, then it will also be useful outside of the academic community and will hopefully be enhanced by other simulation practitioners thanks to its modular structure. Once its usefulness will be proved, some extensions will need to be provided, such as being able to accommodate non-tabular data.

We believe that software developed as part of research activity should be released with a free software license, because research results should be made available for use by anyone, for any purpose and be freely modifiable, in order to further knowledge and usefulness [21]. The choice of license will be among the MIT X license, the GNU LGPL and the GNU GPL licenses, which we think the best serve the purpose of free research software [22].

References

[1] Arnold Bragg, "Observation and Thoughts on the Possible Approaches for Addressing the Tasks Specified in the COST 285 Work-Plan", COST 285 TD/285/03/15, http://www.cost285.itu.edu.tr/tempodoc/TD_285_03_15_Arnold%20Bragg.pdf

[2] "The Network Simulator - ns-2", http://www.isi.edu/nsnam/ns/

[3] "Tcpdump public repository", http://www.tcpdump.org

[4] "Extensible Markup Language (XML)", http://www.w3.org/XML

[5] "HDF5 Home Page", http://hdf.ncsa.uiuc.edu/HDF5/

[6] "The Gzip home page", http://www.gzip.org/

[7] "The Bzip2 home page", http://www.bzip.org/

[8] "Ethereal: A network protocol analyzer", http://www.ethereal.com/

[9] "Nam: Network Animator", http://www.isi.edu/nsnam/nam/

[10] "Opnet modeler", http://www.opnet.com/products/brochures/Modeler.pdf

[11] "Tcpdpriv, A tool for anonymising tcp traces", http://www.openbsd.org/3.5_packages/vax/tcpdpriv-1.1.10.tgz-long.html and http://fly.isti.cnr.it/software/tcpdpriv/

[12] "Octave home page", http://www.octave.org/
[13] "The MathWorks - MATLAB and Simulink for Technical Computing", http://www.mathworks.com/
[14] "Mathematica - Wolfram Research, Inc.", http://www.wolfram.com/
[15] "Excel 2003 Home Page", http://office.microsoft.com/en-us/FX010858001033.aspx
[16] "SPSS home page", http://www.spss.com/
[17] "The R Project for Statistical Computing", http://www.r-project.org/
[18] "HDF5 Wins 2002 R&D 100 Award", http://hdf.ncsa.uiuc.edu/HDF5/RD100-2002/All_About_HDF5.pdf
[19] "PyTables – Welcome Page", http://pytables.sourceforge.net/html/WelcomePage.html
[20] "Python Programming Language", http://www.python.org/
[21] Francesco Potortì, "Free software and research", short paper, *proceedings of the International Conference on Open Source Systems*, Genova (IT), pp. 270-271, July 2005.
[22] David Wheeler, "Make Your Open Source Software GPL-Compatible. Or Else.", web essay at http://www.dwheeler.com/essays/gpl-compatible.html, revised February 2005.