# CostGlue: Simulation Data Exchange in Telecommunications

**Dragan Savić[1], Francesco Potortì[2], Francesco Furfari[2],**

**Janez Bešter[1], Matevž Pustišek[1], and Sašo Tomažič[1]**

[1]University of Ljubljana, Faculty of Electrical Engineering,
1000 Ljubljana, Tržaška 25, Slovenia
[2]ISTI-CNR, Pisa, Italy

dragan.savic@fe.uni-lj.si (Dragan Savić)

*The final, definitive version of this paper has been published in SIMULATION, Volume 84, Issue 4 of April, 2008 by Sage Publications Ltd, All rights reserved. \copyright The Society for Modeling and Simulation International (Simulation Councils Inc), 2002. It is available at: http://online.sagepub.com/*

Abstract

Exchanging simulation data among simulation practitioners is to a great extent hindered by the use of different kinds of data formats in simulation software packages. The purpose of the CostGlue project is to facilitate the exchange of simulation data in the field of telecommunications. We propose a common data interchange format and a data exchange model for raw simulation data, metadata and post-processing data. Based on this model, we additionally propose a framework, CostGlue, designed for packaging simulation output data into the common interchange format, launching post-processing plugins and exporting data into input formats for various third party tools. As a proof of concept we have implemented the framework as a software package and released it as free software.

## 1. Introduction

Due to the complexity, size and heterogeneity of telecommunication networks, simulation represents an indispensable, universal and cost efficient approach to the development of new networks and consequently to the development of new services and applications. A number of simulators are currently in use for this purpose. Information about the behavior and operation of a modeled system is gathered from the simulation data, i.e., the output of individual simulation programs. A common simulation data exchange format is required to facilitate the sharing of this information. At present, simulation data exchange is to a great extent hindered by many different data formats used in simulation software packages. This topic has been addressed within the framework of the European COST 285 Action "Modeling and

Simulation Tools for Research in Emerging Multi-service Telecommunications" [1], a forum where European researchers periodically meet to address issues related to simulation of communications systems. COST 285 observed that no general solution exists for exchanging big quantities of simulation data from different sources and in different formats. The need was expressed both for a common format for exchanging data, and for transforming this data for use with different data analysis tools, each requiring a different input format.

Conversion between different tools for simulation data analysis generally requires a conversion program for each pair of formats [2]. This means that a total of $N(N-1)/2$ different conversion programs are needed, when $N$ different formats are used. A common interchange format would simplify data exchange: the number of conversions required to share the data reduces from $N(N-1)/2$ to $N$, as only one conversion tool is needed for each format.

The process of delivering valuable information about the simulated environment involves several activities such as modeling and simulation, analysis and presentation of simulation results, which are generally performed by several different independent applications. We define a reference model of the process involving data collection, simulation and result analysis, in order to provide a general and systematic overview of the simulation procedure.

Following the reference model, this paper describes the project CostGlue [3], which is intended to facilitate simulation data exchange. The project includes the CostGlue *data exchange model*, the *metadata XML description*, the *CostGlue framework*, and a *prototype implementation*. We propose a data exchange model as an abstract model that describes how data is represented and used. The model considers three types of data: raw simulation data, metadata and post-processing data. Special attention is given to the latter two, whose structure is based on the Open Archival Information System (OAIS) reference model [4], modified with a more detailed description of simulation data collection related to telecommunications. All three types of data are stored in an archive based on the common data interchange format. When dealing with a large number of archives, it is possible to aggregate them in catalogues to be published in repositories for scientific communities. For this reason, we use the taxonomy from the Council for the Central Laboratory of the Research Councils (CCLRC) Scientific Metadata Model (CSMD) [5]. The CostGlue prototype, which implements this CostGlue framework, is able to package simulation output data into the common format, launch post-processing plugins and export data into various formats. The framework consists of a core, a database, an API (Application Programming Interface), and an arbitrary number

of specialized plugins. The core is the sole responsible for accessing the database. Specific functions, such as import and export of data and different mathematical calculations, are provided by self-describing plugins, which are loaded on demand. The plugins access the database by interacting with the core through a well-defined API. The prototype CostGlue software package should be viewed as a proof of concept of the feasibility of the proposed data exchange model and the framework. Since measurements and simulation data have very similar format, they can be stored into a common database, which allows easy comparison, a valuable feature in the process of modeling and simulation [6].

The paper is organized as follows. In Section 2 we provide a brief overview of the related work. The simulation process model is described in Section 3. It is followed by a short discussion of the simulation tools and data formats in Section 4. The CostGlue data exchange model and the metadata XML description are presented in Section 5 and the accompanying CostGlue framework with available plugins is presented in Section 6. Some concluding remarks are given in Section 7.

## 2. Related work

To the best of our knowledge, no general solution for simulation data exchange exists in the field of telecommunications. However, there are some relevant examples that partially address the idea of simulation data exchange among researchers. Starting with those most related to the field of telecommunications, several online catalogues with measurement data for different types of telecommunication networks are available. One of them is the Community Resource for Archiving Wireless Data At Dartmouth (CRAWDAD) [7], a wireless network data resource for the research community. It hosts several data sets in the form of trace files (snmp, tcpdump, syslog, etc.) in different data formats. The data sets are accompanied by several analysis tools. In addition, the CRAWDAD archive contains metadata with the description of CRAWDAD data, tools, related papers and their authors. Despite the efforts of the CRAWDAD community, the diversity of different file formats and tools for each format still hinders data exchange among the researchers.

Multilayer Network Description (MND) [8] aims to assist multilayer modeling by improving the data interchange between various tools. This is accomplished by defining a generic data structure for the multilayer environment which is also present in communication networks. It uses eXtensible Markup Language (XML) to increase flexibility and to improve the visibility of relevant data. The drawback of the MND is that it represents only one of the many possible descriptions of a simulated environment, therefore not enabling general

applicability in all of the simulated environments relevant to the domain of telecommunications.

A software tool called DataSpork [9] developed at the University of Illinois comes from a different research area. It is distributed as a part of the Material Computing Center software archive. DataSpork was designed in order to perform statistical analyses common to most simulation methods, as well as to allow for an extension to other data types and analysis tasks. Although it is a standalone tool, its development is a part of an overall effort to improve storage and exchange of simulation data in the area of Materials Computation. It enables the creation of new data readers, which means that the data can be imported from any kind of data source as long as the necessary extensions are developed. This tool has no capability of data export, multidimensional data layout abstraction, flexible data viewing or support for structured metadata.

The American Institute of Aeronautics and Astronautics (AIAA) Modeling and Simulation Technical Committee has proposed a standard [10] for the interchange of simulation modeling data of a vehicle or an aircraft between different simulation facilities. The purpose of the standard is to maximize the efficiency of data models exchange by providing a well-defined set of information, definitions, data tables and axis systems. The standard is implemented in XML. Unfortunately, the descriptions of models within the XML files are specific to the field of aeronautics and can not be used in the field of telecommunications.

The CCLRC e-Science Centre has developed a framework, called AgentX [11], which allows simple and automated exchange of information between components of a scientific workflow. The AgentX framework is being used in the eMinerals project [11], which focuses on studying environmental processes at the molecular level by using a range of atomistic simulation tools. From this project we borrowed CSMD, the taxonomy developed by CCLRC for aggregating the archives into catalogues.

## 3. Simulation process model

In order to obtain a general and systematic overview of creation, flow and processing of data, we define a reference simulation process model. The model provides a layered decomposition of the main functions encountered in both the simulation and measurement processes. This three layered model with one optional sub-layer is shown in Fig. 1.

The first layer - *source layer* - provides the raw simulation output, describing a simulation run to the smallest detail. Raw data is generated by a simulator: usually, one or more records are created for each event during the simulation run at the source layer. The structure and

format of the data at this point depends entirely on the simulator (e.g., ns-2, Opnet). Most frequently data is in the form of large tabular traces, in ASCII or binary format. An optional *source recoding sub-layer* handles the raw source data. Its main purpose is to convert between different formats (e.g. from ASCII to binary or vice versa), to compress data (e.g., Gzip, Bzip2) and to remove private information (e.g. header lines) from simulation traces. The source layer supports both raw simulation data and real measurements data. In fact, during our research, we found 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 simulation, but rather, for example, the data traces captured in a network link. Apart from the different tool that generates the raw data (traffic capture tools like Tcpdump or Ethereal instead of a simulator) all the functions of the upper layers remain the same.

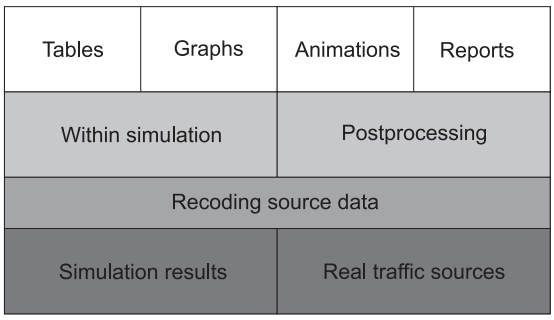


Figure 1: a reference model of a simulation process.

The *processing layer* is responsible for the analysis of simulation data. At this level, cumulative results can be derived from the raw data (e.g. the mean packet delay can be calculated). It is possible to determine the statistical confidence of the results and to conduct additional simulation runs if necessary. An important characteristic of the data processing layer is that the amount of data received from the source layer is usually much larger than the amount of results of post-processing.

The *presentation layer* is the final stage where the results are organized in a form useful for exchanging the most important findings with other simulation practitioners. In the case of simple tabular result printouts, this layer is empty or only slightly modifies the data (e.g. changes in number formats, column spacing). However, data is frequently shown in the form of 2- or 3-dimensional graphs (e.g. as a part of scientific reports or research papers, web pages, etc.) or even presented in animated form (e.g. ns-2 NAM - Network AniMator). At this layer the predominant requirement is the 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 complementary tools that covers the required scope of functions within the model. The selection of tools is based on arbitrary conditions, such as the capabilities and performance of the individual tools, the researcher's past experience with a particular tool, or the availability of tools. In the case of simulated results, raw data can be generated by discrete event simulators (e.g., ns-2, Opnet). Raw data can be captured in real networks, with sniffers, such as Tcpdump or Ethereal. Source recoding can be done with small dedicated tools (e.g., Gzip, Tcpdpriv) or different proprietary shell scripts. Data analysis can be performed with generic tools for mathematical computation (e.g., Octave, Matlab, Mathematica, Excel), special statistical tools (SPSS, R), 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 an additional more powerful and flexible tool should be used. Besides dedicated graphing or animation tools, presentation ability can be provided in generic mathematical tools and sometimes even in the simulators themselves.

## 4. Simulation tools and data formats

In order to provide an overview of tools and formats generally used by telecommunications systems practitioners, we addressed a specific questionnaire to the participants of the COST 285 project, representatives of more than ten European nations. The information we gathered can be summarized in the following observations:

- No single simulation tool has a dominant position. On the contrary, there is a great variety of simulation tools in use.
- Apart from tabular data, other types of elaborate structural formats are seldom used .
- Most of the time the data is used for statistics or graphing. Other uses such as data mining are rare.
- The most common method for evaluating the statistical accuracy of the simulation data is to use independent replications: runs with different simulation times are combined in order to assure stationary intervals of appropriate length. An alternative method using the *equivalent correlation length* obtained by a single simulation run is seldom used.

- A single simulation run produces anywhere from 1 MB to 2 GB of data and a simulation campaign requires from 1 to 100 runs. A measurement campaign requires from 1 to 5 runs, each generating from 100 MB to 50 GB of data.
- The required storage varies from up to 1 GB for short-term storage, to anywhere from 10 MB to 10 GB and more for long-term storage.
- The employed metadata include type, date, parameter values and their description, version tracking, configurations, simulation scripts, and location.
- The metadata is stored in different locations: coded into directory and file names, in separate files, in different storage location inside files (e.g., under the root directory, in shared directories), and in databases.
- Among the simulation tools which use a predefined output format, the most common appears to be the network simulator ns-2.
- Among the generic tools for mathematical computation and running simulations, Matlab appears to be used by most practitioners.
- A large part of the simulators is composed by standard scripting or programming languages and, in general, by ad hoc simulators.
- A great variety of tools is used for post-processing and/or graphing.

These observations, while limited in scope, show that some sort of ASCII format with tabular data is used relatively often.

The variety of tools and data formats used in simulations calls for a general way of reading from and writing to different formats. For this reason we suggested a modular architecture for the CostGlue framework.

## 5. CostGlue data exchange model

Most simulation data in telecommunications are basically sets of tables with numeric data. Each simulation run generates a table with a few columns and a large number of rows. Each table is associated with certain parameters specific for the simulation, and is uniquely identified by the values of these parameters. We are interested in defining a database structure that is able to efficiently accommodate this type of data.

In the next subsection a structural description of the data exchange model is presented together with the accompanying detailed description of the structure of metadata and post-processing data. This is followed by an overview of data manipulation.

5.1. Structural description of the data exchange model

The questionnaire presented in the previous section revealed that, most commonly, simulation data is organized into a hierarchical structure accessed via a set of parameters. This is usually done by hierarchically organizing the directories of a file system, with raw data usually located in the leaf directories. Each directory is named after the value of the parameter corresponding to its depth. Generally, this means that the number of tree levels depends on the number of different simulation parameters, and the number of directories at each tree level depends on the number of values of the parameters.

A more user-friendly method is to organize the simulation data in the form of a multidimensional array, where parameter values are used as indices into the array. By appropriately indexing the multidimensional array, it is possible to easily extract slices of the whole set of data. Hence the simulation parameters act as the primary interface to raw data, which simplifies raw data querying for users. This is especially useful when the simulation data is collected from multiple simulation runs, which is the common case.

Fig. 2 presents the multidimensional structure of the simulation data. The rows of the parameter table point to data groups and datasets, where the raw simulation data, metadata and post-processing data are stored.

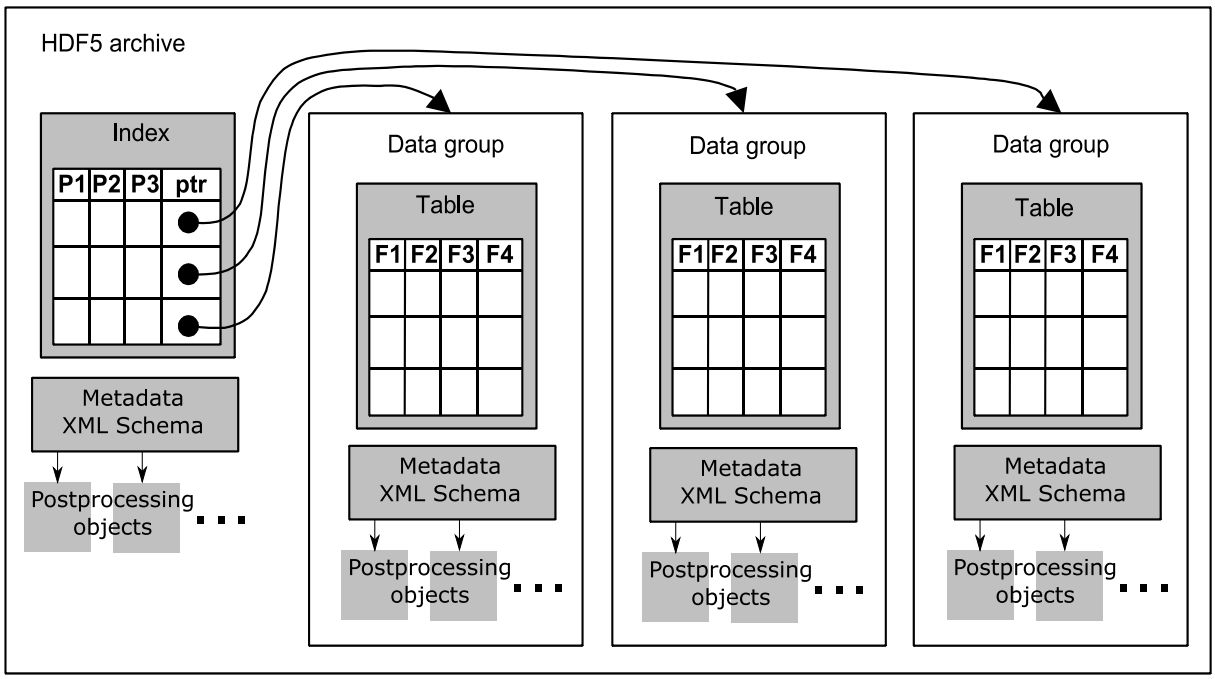


Figure 2: CostGlue multidimensional structure of the data organization.

Each table, together with *metadata* and *post-processing data*, is attached to a data group which usually holds the data produced during a single simulation run. Data groups are indexed with vectors of *parameters* representing an individual simulation run. The index into

the data groups is a 2-dimensional array, also referred to as the *parameter table*, where the parameters relative to each data group are stored. Each column in the parameter table corresponds to a different parameter, and each row contains the values of the parameter relative to a data group. Therefore a parameter table is used as the data structure for accessing a data group, while an array of parameter values relative to that data group is used as the key. A table with raw simulation data is attached to each data group.

The overall structure is a collection of 2-dimensional tables indexed by arrays of $P$ parameters, as shown in Fig. 2. This can be logically seen as a matrix with $P$+2 dimensions, where the first $P$ dimensions are sparse and the last 2 dimensions ($P$+1 and $P$+2) are dense. The first $P$ indices are defined as parameters identifying a data group. As for the last two indices, the first one represents the record (row) number in the data group table, and the second one is the field (column) number of the data group table.

### 5.2. Metadata and post-processing data

In recent years increasing attention has been devoted to metadata in all application domains. The XML Schema [12] provides a means for defining the structure, contents and semantics of an XML document and is therefore widely used to collect metadata, that is, data about data. In order to insert metadata we have defined a metadata XML Schema whose document instances can be saved together with the simulation data, as shown in Fig. 2. Metadata can be associated to every data group; metadata referring to the archive as a whole is saved together with the parameter table, while metadata for a single simulation run is saved in the related data group. Metadata can also refer to any kind of additional data, labeled as *post-processing objects* in Fig.2. Such cases are, for example, the statistics on the raw simulation data, charts, images, and any other type of data produced from or relevant to the raw simulation data.

The metadata XML Schema is derived from the Information Model defined in the OAIS (Open Archival Information System) reference model [4] and uses parts of the Scientific Data Model (CSMD) [5]. The OAIS reference model is a technical recommendation enabling permanent or indefinite long-term preservation of digital information. The objective of the CSMD model is to enhance interoperability of scientific information systems among research organizations. The adoption of a common XML schema from the CSMD model could facilitate further aggregation of telecommunication archives in catalogues to be published in repositories for the scientific communities [7, 13].

Three main elements are present in the metadata XML Schema: one for the metadata relative to the root (*Study*), one for the metadata relative to a data group (*DataGroup*), and one that combines them together (*Archive*), as shown in Fig. 3.

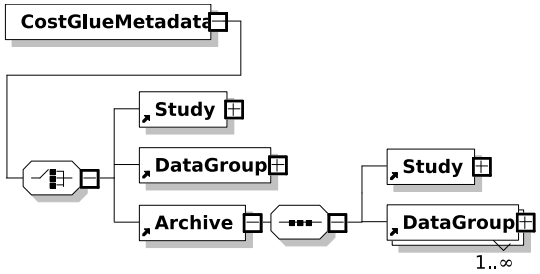


Figure 3: Main element of the CostGlue metadata XML Schema.

The metadata relative to the root is stored in the root of the archive together with the parameter table while the metadata relative to each data group is stored in the data group together with the data table.

More details on the CostGlue XML Schema with additional pictures are available at [14]. Most of the elements in the schema are optional. The choice is motivated by the need not to impose an excessive burden on the experimenter. However, the drawback of this choice is the possibility to have vastly incomplete metadata. On the other hand, it is possible to envision that a certain plugin can certify variable degrees of completeness of the metadata. This way the repositories can accept only the archives that comply with a certain degree of metadata completeness.

The metadata is separated from the rest of the data allowing for easily describing the complete archive. Metadata requires little data storage and can also be made part of the repositories, because metadata includes pointers to the archive location. Data group elements can be extracted separately for efficiency and flexibility.

The metadata Schema does not include a way to serialize the archive data. This choice was made because we see the possibility of exporting the whole archive (simulation data together with metadata) in the XML format as a feature at a different level. Specifically, the HDF5 file format used in the implementation of the software prototype includes the XML schema [15] and the tool [16] to convert a whole HDF5 binary file into the XML file.

### 5.3. Data manipulation

The manipulation part of the data exchange model includes updating and querying the data contained in the database. Updating the data involves importing different simulation outputs

which often have different file formats. After conversion, the raw simulation data is stored into tables, with the user having to specify data types of the table fields.

When working with raw data in post-processing stage the idea is to take advantage of the logical multidimensional data layout abstraction, where the query output is a slice of data spanning one or more simulation runs. Since the whole database can be seen as a sparse matrix with P+2 dimensions the slices of data can be extracted either using *selectors* written in *index notation* or assigning different conditions to arbitrary fields of the table. Both types of queries produce two-dimensional arrays, which can be used for further computations or plotting.

## 6. CostGlue framework

Besides the data exchange model, we have also designed a corresponding CostGlue framework for simulation data exchange. The architecture of the framework, shown in Fig. 4, consists of a core, database, API and several specialized plugins.

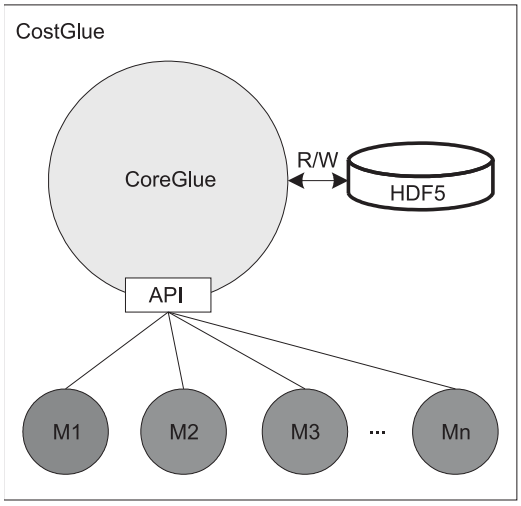


Figure 4:The CostGlue framework.

The core (see Fig. 5) is responsible for reading from and writing to the database, as well as for the dynamic loading of the plugins devoted to different specialized tasks. The core contains three different queues: command, result and report queue. They are used for exchanging different types of messages with the plugins.

For the database we have chosen the HDF5 (Hierarchical Data Format) [17] as the format to store the described data structure in Chapter 5. HDF5 consists of two primary objects - dataset and data 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 homogeneous (only one data type used within a single dataset - simple datasets) or compound (different data types

within one dataset - compound datasets). Since tabular data collected from simulators often contains data in different forms (e.g. integer, float, char), we used compound datasets for our framework. A data group is a structure containing zero or more objects hierarchically organized by means of a tree-like structure, where an arbitrary number of objects are derived from the main "root" data group. Data groups and datasets have a logical counterpart in directories and files in a hierarchical file system and, similarly to a file system, one can refer to an object in the simulation archive by its full path name.

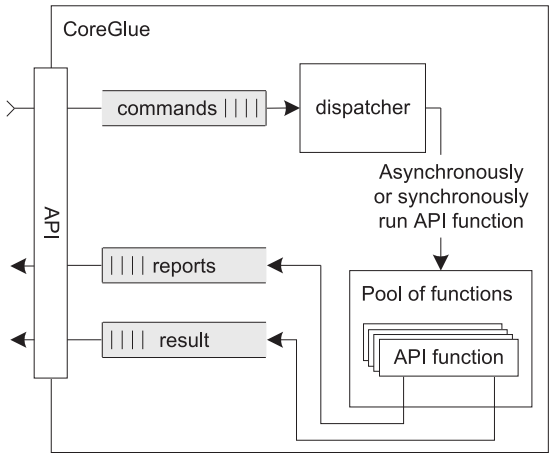


Figure 5: CoreGlue: the core of the CostGlue framework.

The data stored in the database, which is represented as a P+2 dimensional matrix, can be queried by using selectors written in index notation. In this notation, each of the P+2 indices can be "1", indicating the smallest index; "end", indicating the highest index; "n:m", indicating all the indices between n and m included; ":", indicating the whole range from the smallest to the highest index, "n:s:m", indicating the range from n to m in steps of s; or an array like "[1 5 6 8]", indicating the selected indices.

When using the framework for simulation data exchange, the first step is to call the core, together with a plugin name and its optional arguments. A given plugin can work either alone, e.g. by performing a batch job, or it can depend on other plugins, e.g. when a command line interface calls a specialized plugin to perform a task. Plugins also differ in the way they are built. The simple ones wait for the results after sending the command to the core, and are therefore unable to perform any additional tasks. On the other hand, complex plugins run asynchronously in relation with the core, so they can periodically check the result queue for results and, in the meantime, inform the user about the current work progress by reading report messages from the report queue. Plugins are run as separate threads inside the core. Examples of tasks a plugin can perform are:

- data import/export to/from the database,

- statistical computations over data stored in the database,
- data extraction from the database with complex filters,
- transformations of the data contained in the database,
- graphical output created from the data in the database,
- a generic Graphical User Interface (GUI) for exploring the data in the database, doing simple import/export or computations and running other available plugins,
- a plugin-specific GUI, etc.

### 6.1. Overview of the API for plugins

The core of CostGlue framework can look for available plugins and query them one by one to get to know their capabilities. This can be used, for instance, to build a menu for a GUI. Plugins are able to describe the parameters they need and the type of output they produce. The core provides the required parameters to a plugin with or without input from the user. In the case where user input is required, the core checks the provided parameters for consistency. A GUI can also use this information and present the choices to the user. The plugins communicate with the core through a well-defined API which contains all the necessary classes and methods for interacting with the database. The methods allow a plugin 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 post-processed data. Below are a few examples of commands provided by the API:

- Archive open (filename, flags) - opens an archive on disk: arguments are the same as in the C stdio library, returns an archive object.
- Group open (paramvals, exists) - paramvals is a list of parameter values; if exists is true, returns the existing data group with the given parameters, else return a newly created group with the given parameters.
- FieldList add (fieldlist) - adds fields (that is, columns) to a table by taking a list of field names, types (e.g. Int8, Float32) and sizes of the fields.

### 6.2. Implementation of the CoreGlue framework

As we are interested in storing simulation data into a common database, we made a thorough analysis of the different possibilities of data storage. We focused mainly on the scientific data formats, as they are most commonly used in the scientific community. After the

analysis we narrowed our research down to a specific set of scientific data formats. These were: HDF4, HDF5, netCDF, ODB, FITS and OpenDX. In addition to the scientific data formats, we also considered using plain text formats, relational and object oriented databases. In the end we decided to use the HDF5 [17] data format. The reason for such a decision was that HDF5 meets all the requirements of data organization, (i.e. separation of raw data and metadata) and different requirements of contemporary computer system architectures. These are: managing huge 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, and sustained development and maintenance. As a confirmation that we have made a good decision, the netCDF format itself has recently decided to evolve towards using HDF5 as its underlying storage format.

Another important issue was the representation of the data. The HDF5 is a binary file format, which means that we need a compatible application capable of reading the file format to view the data. This is a disadvantage, since plain text file formats are much easier to be read. However, this problem is easily solved by using a specialized HDF5 tool [16] provided by the HDF5 development group, which converts binary files into XML text files.

Converting large binary files into XML text file format has some disadvantages. Data stored in XML format requires much more space than the same data stored in HDF5 binary format. Thus, the conversion results in a high data overhead. Data lookup also slows down, since XML parsers are not very efficient when dealing with large amounts of data, compared with the data read performance of the HDF5 library. Nevertheless, this option is available and the choice is left to the users.

For easy and efficient data manipulation we chose PyTables, a Python library built on top of the HDF5 library uses NumPy as a support package, allowing for sophisticated scientific computations.

The extraction of raw simulation data for post-processing is supported by using the Matlab-like index notation. By this it is possible to take complex orthogonal slices of the multidimensional matrix composed of all the data in the database.

The core of the software framework is written in Python. This means that the plugins for specialized tasks also have to be written in Python. To enable plugins to be written in other scripting or programming languages, a wrapper should be built around the core. In addition, we use Psyco, a Python extension module, which speeds up the execution of Python code significantly.

In order to give a flavor of the performance of the prototype application we imported and exported an ns-2 trace, which is a mixture of numbers and strings in ASCII tabular form. In this test both the read and write speeds were around 1 MB/s. For a reference, Matlab on the same machine reads and writes a big matrix of numbers (no strings) with a speed of around 3 MB/s. Matlab has similar speeds both when reading from ASCII and writing to binary or the other way around.

### 6.3. Examples of plugins

Plugins are meant to be used for specialized tasks such as the import, export, computation, and visualization of simulation data. We have developed a number of plugins, which are briefly described below.

***A plugin for ns-2 and tcpdump*** – this plugin is used for importing ns-2 simulation data and tcpdump measurements into a common table format in order to enable the comparison of these two types of data. This proved to be a valuable feature, especially in the process of modeling and simulation [6]. Two data formats are available: a *full* one, that preserves all the input data, and a *minimum* one, that stores only those data that can be safely converted from ns-2 to tcpdump and vice versa. Both formats can be converted into an input file for NAM, the ns-2 network animator, for a movie-like view of the network behavior at the packet level.

***The command line and the HTML GUI plugins*** – the interactive command line plugin is used for debugging the core and other plugins. Additionally, it can be used to import and export tabular data, whereby the user is required to specify options and arguments. A specialized plugin can automatically recognize the type of trace and automatically provide suitable naming, and possibly, filtering. The basic feature of the HTTP GUI is that when the core can act as an HTTP server, providing a graphical user interface accessible with any web browser. Through this interface, the user can look at the list of available plugins together with their description, the input they require and the output they provide.

***The plugins for antenna measurements and simulations*** – two different import plugins were designed with the help of the researchers from SSR (Signals, Systems and Radio) group at the University of Madrid. The first plugin deals with antenna measurements made in a special antenna chamber. The measurements need to be converted in order to be stored into HDF5 database in the form of tables. A proprietary visualization plugin from SSR is used for visualizing the antenna's directivity diagrams. The second specialized plugin was developed in order to import data from the GRASP8 antenna simulator. The whole process (import, export and visualization) is quite similar to the one described above.

A very useful and practical feature is the ability to compare the measurement results with the simulated ones. The visualization plugin supports this activity as it allows plotting diagrams from both types of data.

### 6.4. Examples of a real-world simulation

Let us now describe how the data of a real-world simulation can be stored in the described structure. In the first example, taken from [18], we simulate the behavior of packet switches using the ns-2 simulator. Each simulation run is characterized by several parameters. We are interested in the following simulation parameters: architecture type (one of OQ – Output Queuing, VOQ – Virtual Output Queuing, and FPCF - Forward Planning Conflict-Free), buffer size, number of I/O ports, traffic load, transport protocol (TCP and UDP), and flow size distribution (one of heavy-tailed Pareto or constant flow size). Each simulation run differs in at least one of these parameters.

Referring to data exchange model when storing the results of one simulation run in the archive, we populate a new row in the parameter table: values in each row uniquely identify a simulation run. Each row includes the full path to a data group, containing the table with the results of the simulation run, metadata and post-processing data, as shown in Fig. 2. Metadata stores information about the type of scripts used to generate that simulation run, the type of network topology, traffic patterns, etc.

| line | arc. type | buffer | ports | load | protocol | flow dist. |
|---|---|---|---|---|---|---|
| 1 | voq | 60 | 4 | 30 | udp | pareto |
| 2 | voq | 60 | 4 | 40 | udp | pareto |
| : | : | : | : | : | : | : |
| : | : | : | : | : | : | : |
| : | : | : | : | : | : | : |
| 202 | oq | 100 | 4 | 90 | tcp | const |
| 203 | oq | 200 | 4 | 90 | tcp | const |
| 204 | oq | 400 | 4 | 90 | tcp | const |

*Legend:*
**line** – row number of the parameter table
**arch. type** – architecture type of the packet switch (voq – virtual output queuing, fpcf – forward planning conflict free, oq – output queuing)
**buffer** – buffer size of the packet switch queues
**ports** – number of input/output ports of the packet switch
**load** – traffic load of the packet switch
**protocol** – transport protocol (udp – user datagram protocol, tcp - transmission control protocol)
**flow dist.** – distribution of flow sizes (pareto or constant distribution)

Figure 6: Parameter table with few sets of different parameters.

By using the CostGlue multidimensional data access via framework's API, it is possible to obtain the raw data for computational tasks by just specifying the parameter values of interest. Using the simulation parameters as a primary interface to raw data comes in handy especially when we are dealing with more than one observation. In Fig. 6 we can see that 204

simulations are driven with different values of parameters. The obtained slices are further processed and the results are fed to the visualization plugin for plotting graphs, which are depicted in Fig. 7.

Post-processing data in this particular case includes packet loss probability (PLP) for three different observations (see Fig. 7). In the first observation we analyze the PLP versus the type of the transport protocols for different packet switch architectures. The load of the packet switch ranges from 30 to 100%. All other simulation parameters remain constant. The second observation analyzes PLP versus flow size distribution for different types of transport protocols where the load varies from 10 to 100%. The third observation analyzes the influence of buffer size (from 10 to 400 packets) on PLP for different transport protocols and packet switch architectures.

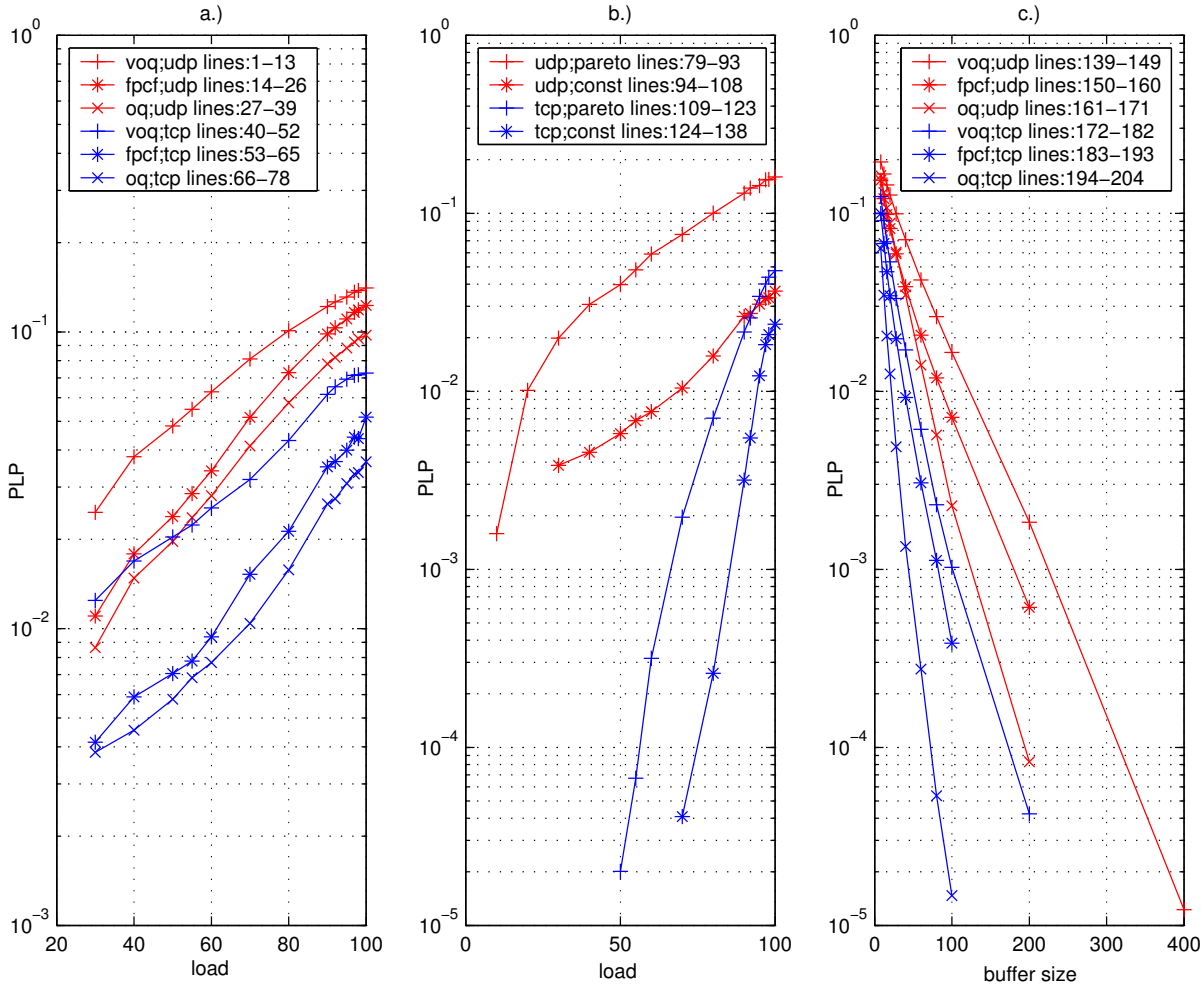


Figure 7: Post-processing results for packet switches.

In the second example we compare ns-2 simulation and measurements for two different network scenarios. Both simulation results and measurement data are saved into the common CostGlue format. Specifically, we compare the packet flow of the ns-2 sending node with the

packet flow of a real sending host where we used a packet sniffer to intercept and log originating traffic (see Fig. 8).

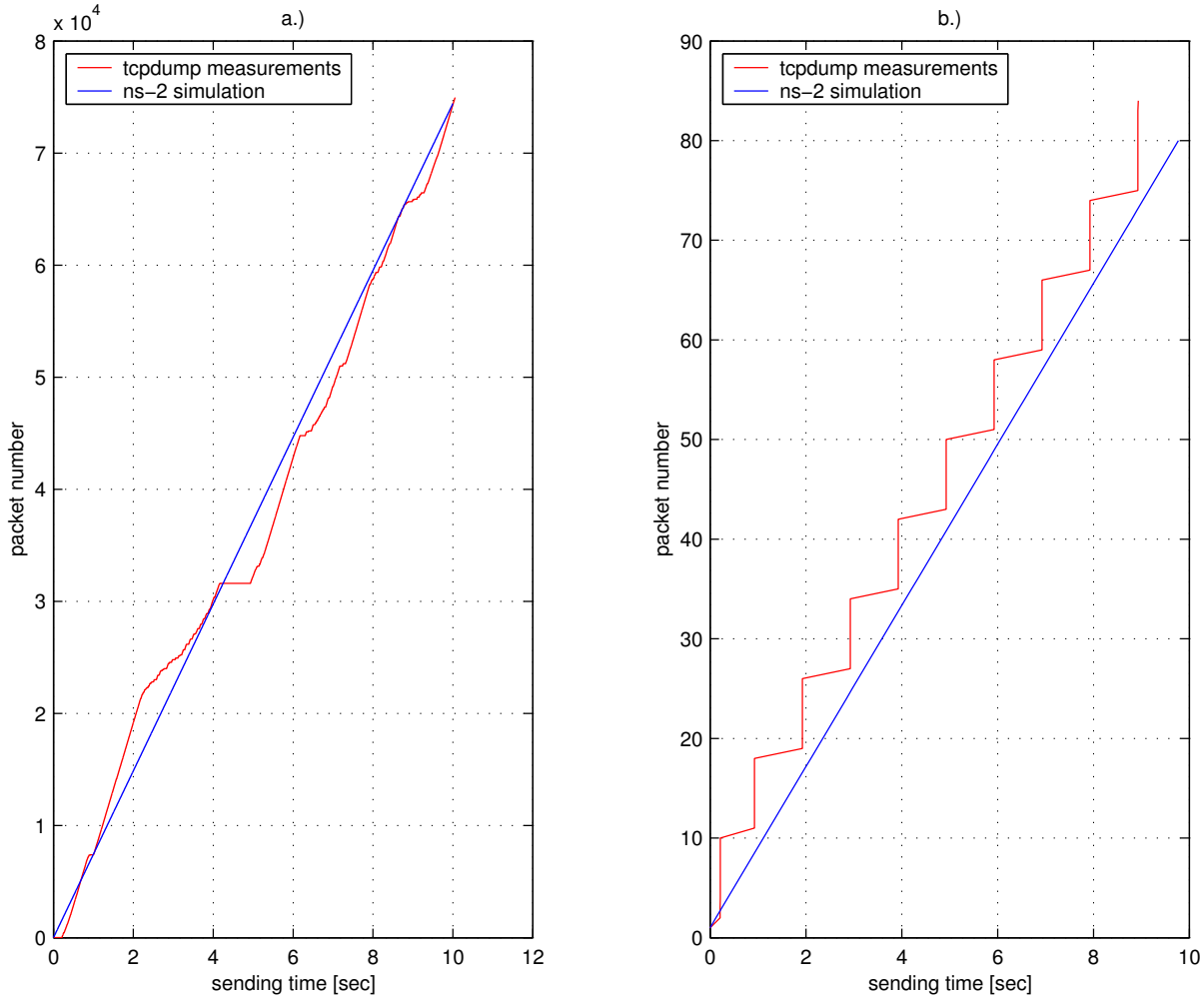


Figure 8: Comparison of simulation and measurements.

The network topology is very simple: it includes two connected nodes acting as FTP server and client respectively. In the first scenario we made a transfer of 100 Mb file from server to client side. The link bandwidth between nodes was set to 10 Mb/s in ns-2, whereas in the case of real data transfer we used a 100Mbit/s Ethernet link. In the second scenario we lowered the bandwidth link in ns-2 to 1 kb/s and used a 10 KB file for transfer. In the real network, we set a download bandwidth limit on the FTP server to 1 kb/s. The sending times for both scenarios are depicted in Fig. 5. Due to the bandwidth restriction set on the FTP server, in the second scenario we got a ramp curve instead of a near linear curve. This was something we did not anticipate. Being able to make this kind of comparisons helps us in the process of verifying models by directly validating them against experiments.

## 7. Conclusions

The purpose of the CostGlue project is to facilitate the exchange and management of simulation data among simulation practitioners and to ease the task of using different simulation, data processing and visualization tools, all of which have different input and output data formats. A prototype program implementing the CoreGlue framework, based on the CoreGlue data exchange model, has already been developed. We sincerely hope that due to its modular structure the prototype will be enhanced by other simulation practitioners, thanks to the use of a free software license [19]. One possible extension of our work includes the use of web services, where CostGlue would act as a black box capable of receiving commands via web service messages and generating results which would then be forwarded back to the user. This could result in an even more flexible exchange of simulation data.

## 8. Acknowledgments

We are grateful to the participants of the COST 285 Action for their helpful cooperation. This work was supported in part by the grant from 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 under grant no. P2-0246.

## 9. References

[1] BRAGG, A.: Observations and thoughts on the possible approaches for addressing the tasks specified in the COST 285 work-plan, COST 285 temporary document TD/285/03/15, CNUCE-CNR (IT), 2004.

[2] Gravitz, P. D., Sheehan, J., and McLean, T.: Common Activities in Data Interchange Format (DIF) Development, report, McLeod Institute of Simulation Sciences, http://www.ecst.csuchico.edu/~hla/LectureNotes/99S_177SIWPaper.pdf, 1999.

[3] CostGlue project web site: http://lt.fe.uni-lj.si/costglue/

[4] CCSDC 650.0-r-2: Reference Model for an Open Archival Information System, Blue Book, Issue 1, ISO 14721, 2003

[5] Sufi S., Mathews B., CCLRC scientific metadata model: Version 2, CCLRC technical report DL-TR-2004-001, September 2004.

[6] Balci O.: Verification, Validation, and Certification of Modeling and Simulation Applications, In Proceedings of the 2003 Winter Simulation Conference (New Orleans, LA), IEEE, Piscataway, NJ, pp. 150-158, 2003.

[7] Kotz D. and Henderson T.: CRAWDAD: A Community Resource for Archiving Wireless Data at Dartmouth, IEEE Pervasive Computing, vol. 4, 4, pp. 12-14, Oct. 2005.

[8] Rumley S., Gaumier C.: Multilayer Description of Large Scale Communication Networks, to be published in the proceedings of the COST 285 final symposium, March 2007.

[9] DataSpork analysis toolkit: http://www.mcc.uiuc.edu/dataspork/

[10] Jackson E., Hildreth B.: Flight dynamic model exchange using xml, AIAA Modeling and Simulation Technologies Conference and Exhibit, Monterey, California, Aug. 5-8, 2002

[11] Tyer RP *at al*: Metadata management and grid computing within the eMinerals project, Proc. UK All Hands Meeting 2007 (AHM2007), Nottingham, UK, Sept. 10-13, 2007

[12] XML schema: http://www.w3.org/XML/Schema

[13] DRIVER – Digital Repository Infrastructure Vision for European Research: http://www.driver-repository.eu/

[14] Furfari F., Potortì F., and Savić D.: The CostGlue XML schema, Tech. Rep. cnr.isti/2008-TR-03, CNR-ISTI, via Moruzzi, 1, January 2008.

[15] HDF5 XML Schema: http://www.hdfgroup.org/DTDs/HDF5-File.xsd.txt

[16] H5dump, a tool for converting binary HDF5 file into a XML file: http://www.hdfgroup.org/HDF5/doc/Tools.html

[17] Folk M., McGrath R., Yeager N.: HDF: an update and future directions, In International Geoscience and Remote Sensing Symposium (IGARSS'99), IEEE, Ed., vol. 1, pp. 273–275, 1999

[18] Pustišek M., Savić D., Humar I., Bešter J.: Transport Protocol Dependent Communications in Different Packet Switch Architectures, IEEE Electrotechnical Conference MELECON 2006, pp 704- 708, May 2006

[19] Potortì, F.: Free software and research, in proceedings of the International Conference on Open Source Systems (OSS), M. Scotto and G. Succi, Eds., ECIG Edizioni Culturali Internazionali Genova, pp. 270–271, July 2005.