

# Functional Specifications for Archiving Distributed Data

Kenny Q. Zhu  
*Princeton University*

Daniel S. Dantas  
*Princeton University*

Kathleen Fisher  
*AT&T Labs Research*

Limin Jia  
*University of Pennsylvania*

Yitzhak Mandelbaum  
*AT&T Labs Research*

Vivek Pai  
*Princeton University*

David Walker  
*Princeton University*

## 1 Introduction

Modern data analysis tasks increasingly involve drawing information from a wide variety of distributed data sources, processing that data locally and analyzing the results. Consequently, computational biologists, cosmologists, financial analysts, systems administrators, distributed systems engineers and others are continually building new ad hoc systems for gathering data from distributed sources and archiving it locally along with provenance meta-data. To make the process robust, auxiliary subsystems must monitor the data acquisition process to detect and catalogue errors.

As an example, consider the data manipulated by CoMon [9], a system designed to monitor the health, performance and security of PlanetLab [10]. Every five minutes, CoMon attempts to contact each of 842 PlanetLab nodes across 416 sites worldwide. When all is well, which it never is, each node responds with an ASCII data file in mail-header format containing information including the kernel version, the uptime, the memory usage, and the ID of the user with the greatest CPU utilization. CoMon archives this data in compressed form and processes the information for display to PlanetLab users.

Similar problems appear in the natural and social sciences, including biology, physics and economics. For example, systems such as BioPixie [3], Grifn [2] and Golem [11], built by computational biologists at Princeton, routinely obtain data from a number of sources scattered across the net. Often, the data is archived and later analyzed or mined for information about gene structure and regulation. Figure 1 summarizes selected distributed ad hoc data sources.

This paper presents a declarative specification language, called PADS/D, that allows users to describe a collection of distributed data sources they wish to gather and archive. More specifically, the PADS/D programmer will specify: **where** the data is located; **when** to get the data; **how** to obtain it; **what preprocessing** the system

Name/Use	Properties
CoMon [9] <i>PlanetLab host monitoring</i>	Multiple data sets Archiving every 5 minutes From evolving set of 800+ nodes
CoBlitz [8] <i>File transfer system monitoring</i>	Multiple data sets Archiving every 3 minutes From evolving set of 800+ nodes
CoralCDN [4] <i>Log files from CDN monitoring</i>	Single Format Periodic archiving From evolving set of 250+ hosts
AT&T Arrakis <i>Website host monitoring</i>	Execute programs remotely to collect data Varied fetch frequencies
AT&T Regulus <i>Network monitoring</i>	Diverse data sources Archiving for future analysis Per minute, hour, and day fetches
AT&T Altair <i>Billing auditing</i>	Thousands of data sources Archiving and error analysis
GO DB [1] <i>Gene function info.</i>	Multiple Formats Uploads daily, weekly, monthly
BioGrid [12] <i>Curated gene and protein data</i>	XML and Tab-separated Formats multiple data sets $\leq$ 50MB each Monthly data releases
NCBI [6] <i>Biotechnology info.</i>	Links to multiple bioinformatics datasets

Figure 1: Example distributed ad hoc data sources.

should do when the data arrives; and **what format** the data source arrives in.

The PADS/D system compiles these high-level specifications into a collection of tools for archiving the data and maintaining provenance metadata. Our current tool suite includes an archiver, a printer, a performance monitor, an database loader, an accumulator, an alerter, a selector and an RSS feed generator. The system can generate all of these tools from PADS/D descriptions and declarative tool configuration specifications.

In the remainder of this short paper, we describe our

```

let sites =
  [
    "http://pl1.csl.utoronto.ca:3121";
    "http://plab1-c703.uibk.ac.at:3121";
    "http://pl1.csl.utoronto.ca:3121"
  ]
feed simple_comon =
  base { |
    sources = all sites;
    schedule = every 5 min, starting now,
              timeout 60.0 sec;
    format = Comon_format.Source;
  |}

```

Figure 2: `simple_comon.fml`: Simple CoMon feed.

```

feed comon_1 =
  base { |
    sources = any sites;
    schedule = every 1 min,
              lasting 2 hours;
    format = Comon_format.Source;
  |}

```

Figure 3: `sites.fml`: Code fragment for data from one of many sites.

language and system using Princeton’s CoMon [9] system as an example. Readers interested in further examples, a semantics for the language and comprehensive related work should refer to our technical report [13].

## 2 PADS/D: Introduction by Example

The PADS/D language allows users to describe *feeds*, which is the term we use for streams of data and meta-data. To introduce the features of the language, we present a series of examples drawn from the CoMon monitoring system.

The PADS/D specification in Figure 2 describes a simple CoMon statistics feed `simple_comon` using the `base` feed constructor. This constructor contains three fields. The `sources` field indicates that data for the feed comes from all of the locations listed in `sites`. The `schedule` field specifies that relevant data is available from each source every five minutes, starting immediately. When trying to fetch such data, the system may occasionally fail, either because a remote machine is down or because of network problems. To manage such errors, the schedule specifies that the system should try to collect the data from each source for 60 seconds. If the data does not arrive within that window, the system should give up. Finally, the `format` field indicates that the fetched data conforms to the PADS/ML [5] description

```

(* Ocaml helper values and functions *)
let config_location =
  ["http://summer.cs.princeton.edu/status/ \
  tabulator.cgi?table=slices/ \
  table_princeton_comon&format=nameonly"]
let makeURL (Nodelist.Data x) =
  "http://" ^ x ^ ":3121"
let old_locs = ref []
let current list_opt =
  match list_opt with
  Some l -> old_locs := l; l
  | None -> !old_locs

(* Feed of nodes to query *)
feed nodes =
  base { |
    sources = all config_location;
    schedule = every 2 min;
    format = Nodelist.Source;
  |}

(* Dependent CoMon feed of nodes feed *)
feed comon =
  foreach nodelist in nodes
  create
  base { |
    sources = all
      (List.map makeURL
       (List.filter Nodelist.is_node
        (current nodelist)));
    schedule = once, timeout 60.0 sec;
    format = Comon_format.Source;
  |}

```

Figure 4: `comon.fml`: Uses feed of node locations to drive data collection.

named `Source` defined in the file `comon_format`.

With replicated distributed systems, many sites may contain all the desired information. For efficiency, monitors for such systems typically ask for data from all sites in parallel but stop polling as soon as one site returns the required information. The `comon_1` feed defined in Figure 3 specifies this behavior by using the `any` constructor in the `sources` field. The schedule for `comon_1` indicates the system should fetch data every minute for two hours, using the `lasting` clause to indicate the duration of the feed. It omits the `starting` and `timeout` specifications, causing the system to use default settings for the start time and the timeout window.

The `simple_comon` example hard-codes the set of locations from which to gather performance data. In reality, the CoMon system has an Internet-addressable configuration file that contains a list of hosts to be queried, one per non-comment line. This list is periodically updated to reflect the set of active nodes in PlanetLab.

Figure 4 specifies a version of the `comon` feed that

```

ptype nodeitem =
  Comment of '#' * pstring_SE(peor)
| Data of pstring_SE(peor)

let is_node item =
  match item with
    Data _ -> true
  | _ -> false

ptype source =
  nodeitem precord plist (No_sep, No_term)

```

Figure 5: `nodelist.pml`: PADS/ML description for CoMon configuration files.

depends upon this configuration information. To do so, the description includes an auxiliary feed `nodes` that describes the configuration information: it is available from the `config_location`, it should be fetched every two minutes, and its format is described by the PADS/ML description `source` given in the file `nodelist.pml`, which appears in Figure 5. This PADS/ML description specifies that `source` is a list of new-line terminated records, each containing a `nodeitem`. In turn, a `nodeitem` is either a '#' character followed by a comment string, which should be tagged with the `Comment` constructor, or a host name, which should be tagged as `Data`. The description also defines a helper function `is_node`, which returns true if the data item in question is a host name rather than a comment. Given this specification, the `nodes` feed logically yields a list of host names and comments every two minutes. In fact, because of the possibility of errors, the feed actually delivers a *list option* every two minutes: Some if the list is populated with data, `None` if the data was unavailable at the given time-slice.

Using the `nodes` specification, we define the `comon` feed as a *dependent* feed: each `nodelist` in the `nodes` feed defines a collection of sources for the `comon` feed. The `comon` **sources** specification processes the `nodelist` to manage errors and strip out comment fields. The code that handles this processing illustrates that the PADS/D domain-specific language is embedded in OCAML. We use OCAML terms where necessary to specify simple transformations. In particular, the `current` function checks if the `nodelist` is `None`, signaling a fetching error, in which case it uses the most recently cached list of nodes instead. The **source** specification filters out comment fields, and then converts the host names to URLs with the required port using the auxiliary function `makeURL`. The **schedule** for this CoMon feed is `once` (with a timeout of sixty seconds) because we want to collect the data for each host in a given `hostlist` just once. The **foreach ... create** construct merges the resulting data from each machine into a single feed. As before, the format of

data fetched from each node matches the description `Comon_format.Source`.

With this specification, we expect to get data from all the active machines listed in the configuration file every two minutes. We further expect the system to notice changes in the configuration file within two minutes.

The previous examples all showcased feeds that contained a single type of data. PADS/D also provides a datatype mechanism that allows us to construct compound feeds containing data of different sorts. As an example where such a construct is useful, the CoMon system includes a number of administrative data sources. If `sites_mime` is a feed description of the profile information of all the nodes on CoMon, and `sites_keyscan_mime` is a feed of authentication information for each node, then the declaration

```

feed sites =
  Locale of sites_mime
| Keyscan of sites_keyscan_mime

```

creates a feed with elements drawn from each of the two feeds. The constructors `Locale` and `Keyscan` tag each item in the compound feed to indicate its source.

### 3 PADS/D: Tool Generation

The PADS/D system provides a suite of “off-the-shelf” tools to help users cope with standard data archiving and administration needs. After writing a PADS/D description, users can customize these tools by writing simple *configuration files*, such as shown in Figure 6. Each configuration file includes a feed declaration header and a sequence of tool specifications. The header specifies the path to the feed description file (`comon.fml`) and the name of the feed to be created (`comon`). Each tool specification starts with the keyword `tool` followed by the name of the tool (e.g., `feedaccum` and `rss`). The body of each tool specification lists name-value pairs, where values are OCAML expressions. Some attributes are optional, and the compiler fills in a default value for every omitted attribute. PADS/D compiles a configuration file into an OCAML program that creates and archives the specified feed, configures the specified tools, and applies them to the feed in parallel. In the following paragraphs, we describe the tools we have implemented.

**Archiver.** The archiver saves the data fetched by a feed in the local file system, organizing it according to the structure of the feed, with one directory per base feed. It places a catalog in each directory documenting the provenance of the data include its location of origin, its scheduled arrival time and the actual arrival time. The archiver will optionally compress files.

```

feed comon.fml/comon
tool feedaccum {
    minalert = true;
    maxalert = true;
    lesssig  = 3;
    moresig  = 3;
    slicesize = 10;
    slicefile = "slice.acc";
    totalfile = "total.acc";
}
... configs for other tools ...

```

Figure 6: `comon.tc`: Example tool configuration file.

**Printer.** The printer outputs the contents of a feed. If configured to print to a single file, the tool concatenates successive items with a specified separator. If configured to print to multiple files, it outputs the contents of each base feed into a separate file.

**Profiler.** The profiler monitors performance, reporting throughput, average network latency and average system latencies over a period of time. Users can specify in the configuration when to profile and for how long.

**Accumulator.** The accumulator maintains statistical profiles for feeds, including their error characteristics. For numeric data, the accumulator keeps aggregates such as averages, max/min values and standard deviations. For other data (*e.g.*, strings, URLs and IP addresses), it keeps the frequency of the top  $N$  most common values. For all data, it tracks error rates, the most common error values and their sources. The user can configure the accumulator to profile entire feeds at once, or incrementally. The latter option allows users to continuously monitor feeds and compare their current behavior with historical statistics. The accumulator can output either plain text or XML.

**Alerter.** The alerter allows users to register boolean functions which generate notifications when they evaluate to false on feed items. The tool appends these notifications to a file, which can be piped into other tools. The system provides a library of common alerters such as exceeding max/min thresholds or deviating from the norm (*i.e.*, trigger an alert when a selected value strays more than  $k$  standard deviations from its historical value). Users can supply their own conditions by giving OCAML predicates in the configuration file. These predicates can consider both feed values and meta-data about the feed when determining whether to trigger an alert.

**Database loader.** This tool allows users to load numerical data from a feed into a Round Robin Database (RRD) [7]. Users specify a function to transform feed items into numeric values and RRD parameters such as data source type and sampling rate. RRD indexes the data by arrival time. It periodically discards old data to make space for new. The tool supports time-indexed

queries and displays historical data as graphs.

**Selector.** The selector allows users to choose subcomponents of feed elements using path expressions. It returns a feed of the selected subcomponents, which may then be fed into other tools.

**RSS feed generator.** The RSS feed generator converts a PADS/D feed to an RSS feed. Users specify the title, link (source), description, update schedule and contents of the RSS feed. Content specifications are written in the path expression language.

## 4 Conclusions

The explosive growth of the Internet has made monitoring and managing data systems distributed across wide-area networks increasingly important. The possibility of partial failure and the need to synchronize makes such code tedious and difficult to write correctly, particularly for data experts whose skills are in domains other than networking. In this paper, we describe PADS/D, a specification language for archiving distributed ad hoc data sources. Our language allows users to specify where, when and how their data sources are to be acquired. From these specifications, we generate a variety of useful tools for archiving and processing the data and its provenance metadata. Overall, we hope the system will substantially improve the productivity of the modern data analyst.

**Acknowledgments.** This material is based upon work supported by the NSF under grants 0612147 and 0615062. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

## References

- [1] Gene Ontology Project.  
<http://www.geneontology.org/>.
- [2] M. CL, B. D, H. MA, H. C, and T. OG. Finding function: evaluation methods for functional genomic data. *BMC Genomics*, 7:187, 2006.
- [3] M. CL, R. D, W. A, H. M, C. C, T. CL, D. K, and T. OG. Discovery of biological networks from diverse functional genomic data. *Genome Biology*, 6(13), 2005.
- [4] M. J. Freedman, E. Freudenthal, and D. Mazieres. Democratizing content publication with Coral. In *NSDI*, 2004.

- [5] Y. Mandelbaum, K. Fisher, D. Walker, M. Fernandez, and A. Gleyzer. PADS/ML: A functional data description language. In *POPL*, 2007.
- [6] NCBI. National center for biotechnology information. <http://www.ncbi.nlm.nih.gov/>.
- [7] T. Oetiker. Round robin database tool. <http://oss.oetiker.ch/rrdtool/index.en.html>.
- [8] V. Pai and K. Park. CoBlitz: A Scalable Large-File Transfer Service over HTTP. <http://codeen.cs.princeton.edu/coblitz/>.
- [9] V. Pai and K. Park. CoMon: Monitoring infrastructure for PlanetLab. <http://comon.cs.princeton.edu/>.
- [10] PlanetLab. An open testbed for developing, deploying and accessing planetary-scale services, September 2002.
- [11] S. RSG, H. M, H. C, M. CL, and T. OG. GOLEM: An interactive graph-based gene ontology navigation and analysis tool. *BMC Bioinformatics*, 7:443, 2006.
- [12] C. Stark, B.-J. Breitkreutz, T. Regul, L. Boucher, A. Breitkreutz, and M. Tyers. BioGRID: A general repository for interaction datasets. *Nucl. Acids Res.*, 34:D535–539, 2006.
- [13] K. Q. Zhu, D. S. Dantas, K. Fisher, L. Jia, Y. Mandelbaum, V. Pai, and D. Walker. Language support for processing distributed ad hoc data. Technical Report TR-826-08, Princeton University, July 2008.