

INTELLIGENT ASSISTANCE IN SCIENTIFIC DATA PREPARATION

Steve Chien, R. Kirk Kandt, Joseph Roden,
Richard J. Doyle, and Scott Burleigh

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109-8099

Todd King and Steve Joy

Institute of Geophysics and Planetary Physics
University of California at Los Angeles
Los Angeles, CA 90024-1406

Abstract

Scientific data preparation is the process of extracting usable scientific data from raw instrument data. This task involves noise detection (and subsequent noise classification and flagging or removal), extracting data from compressed forms, and construction of derivative or aggregate data (e.g. spectral densities or running averages).

A software system called PIPE provides intelligent assistance to users developing scientific data preparation plans using a programming language called Master Plumber. PIPE provides this assistance capability by using a process description to create a dependency model of the scientific data preparation plan. This dependency model can then be used to verify syntactic and semantic constraints on processing steps to perform limited plan validation. PIPE also provides capabilities for using this model to assist in debugging faulty data preparation plans. In this case, the process model is used to focus the developer's attention upon those processing steps and data elements that were used in computing the faulty output values. Finally, the dependency model of a plan can be used to perform plan optimization and runtime estimation. These capabilities allow scientists to spend less time developing data preparation procedures and more time on scientific analysis tasks.

Introduction

Scientific data preparation is defined as the application of multiple transformations to collected data sets in order to produce data in an easily usable form. The questions a scientist asks dictate which data are to be collected as well as which transformations are to be applied. The need for simplified scientific data preparation has increased due to the volume of data now collected and the diverse uses for

any specific type of data. Automated scientific data processing systems can be used to simplify this process.

While general scientific data processing systems have existed for some time, the complexity of data types and transformations required in specific domains renders these systems of limited utility. As a result, many scientific teams develop their own software systems to accomplish the data preparation required in their specific domain. These systems suffer because they become too specific, and the effort spent developing such systems are only of value within the context of a particular domain and task. Because scientists desire to reuse their work, hybrid systems are appearing which provide useful analysis tools and definition of domain-specific data types and transformations. Plans are developed in these systems which specify which of the transformations to apply to a collection of data sets. By the nature of the processing steps required in many domains, these plans can become quite complex. We are now at a point where the complexity of these tools requires significant expert knowledge to use.

Master Plumber [King & Walker 1991] is a software tool developed by the UCLA Institute of Geophysics and Planetary Physics to create programs to prepare scientific data. While its primary area of application has been time-series magnetometer data, the tool is applicable to the general task of scientific data preparation.

Master Plumber is a dataflow system. Thus, in Master Plumber, data elements are represented by columns, which are streams of data being processed as they move through the system. Data processing steps are called fittings, and a plan to process a particular form of a dataset into another form is called a blueprint.

Thus, as shown in Figure 1, raw data might be read in using an intro_flatfile fitting, a running average computed using a runstat fitting, and the results written into an output file.

```
1.  intro_flatfile infile=foo
    columns=bx
```

- ```

2. runstat length=1287 shift=1
 columns=bx
3. write_flatfile outfile=bar
 columns=bx,rabx overwrite=YES

```

Figure 1: A Simple Blueprint

A major difficulty in constructing blueprints is tracking the many fitting and column interactions. While a typical blueprint might use 25 columns and 20 fittings, the more complex blueprints use hundreds of columns and 30 or more fittings. Because of the number of possible interactions, constructing and debugging scientific data preparation blueprints is a time-consuming task requiring expert knowledge.

Because of the complexity of the data preparation task, users sometimes make errors in blueprint construction. One type of construction error occurs when a user forgets to set up the data needed for a particular step. Unfortunately, this type of error can go unnoticed until far into the execution of the blueprint, wasting valuable time.

Another common situation is that the exact method of processing the data is dependent upon the character of the data. In this case the user will use some default methods for processing the data, examine the results, and modify the options. This tuning cycle continues until the data is in a satisfactory form.

The final aspect of blueprint development which complicates the development process is that new fittings are added to a system as new needs and requirements arise. In addition, new fittings also evolve with new options and characteristics being added. Any intelligent tool must be readily changed to remain useful in such a dynamic environment.

Currently there are approximately 65 fittings which are part of the standard Master Plumber system. These fittings perform a variety of transformations on the data flow, such as: introducing and writing data into several formats; displaying data on the screen; and actual numerical transformations. There are support libraries which allow for fittings to be written in either C or FORTRAN. A special fitting called PLISP takes programs written in a C-like language and performs the transformations on the data flow. This allows for new processing steps to be initially tested as PLISP programs and later be integrated as full-fledged fittings into the Master Plumber system.

Some scientists use data preparation systems indirectly with the help of software support personnel who write and debug the actual data preparation plans. The goal of PIPE is to make Master Plumber easy enough to use such that this type of support is not necessary. The combination of PIPE and Master Plumber will allow the blueprint developer to develop blueprints easier and faster, allowing them to spend more time on data analysis and less time on data preparation.

## Overview

To achieve these goals of assistance in the scientific data preparation process, PIPE [Chien et al. 1992] provides four capabilities:

1. constraint checking to detect invalid blueprints before execution;
2. diagnosis assistance of blueprints through dependency analysis;
3. optimization of blueprints through dependency analysis; and
4. runtime estimation, using models of fitting runtime performance.

The architecture of the PIPE system is shown in Figure 2. PIPE accepts a blueprint file and a set of descriptors for datafiles and uses a fittings knowledge base to construct a dependency graph representing the computations to be performed by each of the fittings in the blueprint. This blueprint parsing phase uses knowledge of fittings and their options to construct a dependency graph, which indicates for each fitting which columns are accessed and used to modify existing columns, create new columns, or remove existing columns. This dependency graph can then be used by the constraint checking module which determines if any of the constraints associated with the fittings have been violated.

In cases where blueprints must be debugged, PIPE can use the dependency graph to support isolation of the fault in the blueprint. Because the dependency graph tracks all of the operations upon the columns, when the user detects an error in one of the output columns, PIPE can present a list of fittings which modified the column in question. The user can then focus his attention upon these fittings, to determine where the error was introduced into the data, sometimes by plotting intermediate data. After isolating the first fitting at which the column is faulty, the user can query PIPE for information on the fitting to determine which columns were used to compute the changed column. This process continues until the fault is isolated to the data, fitting option settings, or fitting code itself.

PIPE also provides an optimization capability. Because PIPE constructs a full computation dependency graph, PIPE can determine the last fitting in which each column of data is used in the blueprint. Thus unneeded data can be removed from the dataflow, decreasing the execution time. Because many fittings operate on data by default, PIPE distinguishes between default processing and explicit processing. Default computation which does not result in a program output (e.g. plot, output file) can also be removed.

Finally, PIPE provides a runtime estimation capability. Using the dependency graph to determine which columns each fitting processes, and models of runtime for each fitting type, PIPE can provide an estimate of how long the

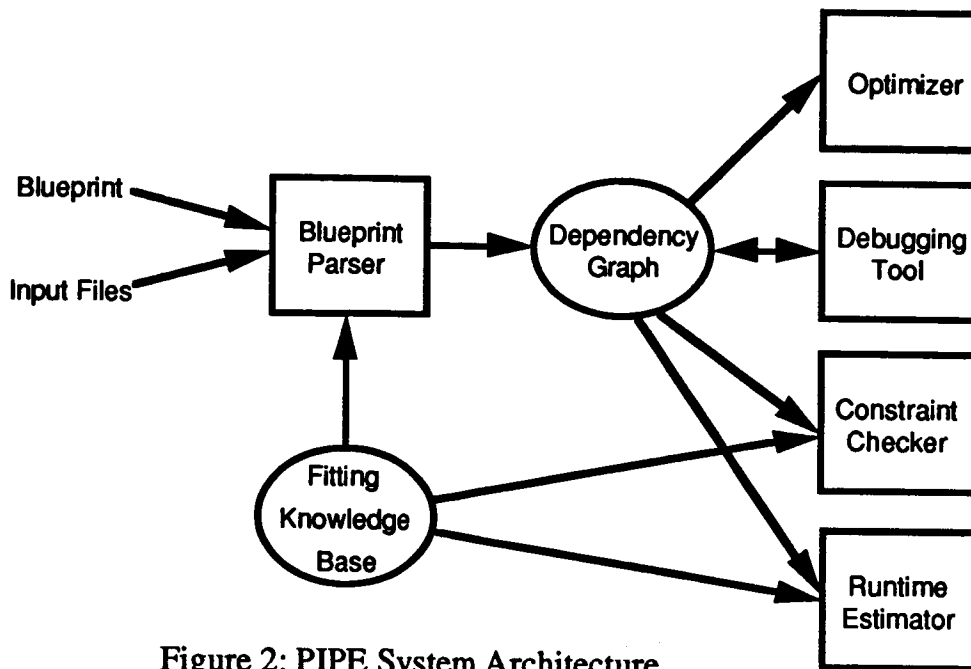


Figure 2: PIPE System Architecture

blueprint will take to run to completion for the specified datafiles.

### Blueprint Parsing

In order to provide assistance in blueprint development, PIPE constructs a dependency network representation of a blueprint. When a blueprint is read in by PIPE, it is processed from the first step onward. For each fitting, PIPE uses:

- methods stored in the fitting knowledge base,
- default values stored in the fittings knowledge base,
- fitting options,
- a list of existing columns in the flow, and possibly
- an input file

to determine:

- any new columns created by the fitting,
- any existing columns modified by the fitting,
- existing columns deleted by the fittings.

Additionally, for any new or modified columns, PIPE determines:

- the set of columns accessed in computing the value for the column.

Because columns may be processed by default or explicitly selected, the dependency network also makes note of this distinction. This facet of the processing is important in order to take appropriate action when optimizing the blueprint (see below).

### Constraint Checking

Constraint checking occurs while the blueprint file is being parsed (i.e., prior to execution). A description of the constraint checking algorithm follows.

#### During Parsing

```

for each fitting in the blueprint
 for each option specified
 check option type constraints
 check for required options

```

#### After Parsing

```

for each parsed fitting in blueprint
 for each option in fitting
 check option value constraints
 check inter-option constraints
 check dependency constraints
 check inter-fitting constraints

```

### Diagnosis Assistance

PIPE also provides a blueprint diagnosis facility. This capability supports two basic types of queries: column-centered queries and fitting-centered queries. The column-centered queries are of the form

"What fittings affected <column>  
before <fitting>?"

and default to the entire blueprint. This question can be easily answered using information from the dependency network. PIPE steps through the fittings in the blueprint and determines those fittings which create, modify, or delete <column>. This list of fittings is then displayed to the user in graphical form. The fitting centered queries are of the form

"What columns did <fitting>  
affect?", and

"What columns did <fitting> access  
in performing its processing to  
affect these columns?"

These types of queries can be answered by interpreting the dependency graph information on the designated fitting. The first query can be answered by determining the set of columns created, modified or deleted by the fitting. The second query can be answered by accessing dependency network information regarding which columns were accessed by the fitting in performing these operations.

### Blueprint Optimization

PIPE also provides a limited blueprint optimization capability. In this capability, PIPE examines the dependency graph of each column and determines the last fitting at which each column is accessed explicitly (i.e., not by default). PIPE then recommends removing this column immediately after this fitting. If this column is not processed in the remainder of the blueprint, this removal does not significantly alter the runtime of the blueprint. However, many of the fittings process all of the columns in the flow by default. Thus, when a column that is processed in the remainder of the blueprint is removed from the data flow a significant speedup can result. While commonly used blueprints are likely to have unused columns optimized by hand, automating this process relieves the user of the burden of determining the point at which a column can be removed. Additionally, by allowing PIPE to automatically determine the correct places to remove columns, PIPE reduces the chance that a user will inadvertently prematurely remove a column from the data flow, which would cause an error.

### Runtime Estimation

The final capability that PIPE provides is runtime estimation. PIPE estimates the runtime of a blueprint for a specific data set by applying the following algorithm:

```
for each fitting in the blueprint
 identify fitting runtime model
 compute runtime given dataset size
 add runtime to total runtime
```

compute new size of dataset

Tracking the size of a dataset in Master Plumber can be a difficult task. Original data set sizes are determined from input files. When data of different temporal granularity are introduced into an existing flow, or when decimation operations are performed, data set sizes will need to be recomputed. Sometimes a fitting can affect the size of the dataset in a manner that depends on the exact data processed. In these cases, the exact dataset size cannot be determined, so PIPE estimates the size of the dataset at the output of the fitting. These estimations are sufficient for giving the user reasonably accurate runtime estimates.

### An Example

We now illustrate each of the capabilities of PIPE using example blueprints. For an example of constraint checking, suppose a user has created a blueprint containing the following statement:

```
4. bin columns=bx delta=60.0 min_max
```

Because the option `min_max` requires that a value be specified, PIPE would indicate a constraint error such as:

- Fitting 4. bin option `min_max` required value not found; string type required.

As another example of the constraint checking, consider the following blueprint statement:

```
7. crossavg except=time avgname=xavg
```

Assuming the user removed the column named `time` earlier in the data flow, PIPE would issue a constraint error indicating:

- Fitting 7. crossavg option `except` undefined column `time`; a column with that name was deleted at fitting 4. drano.

An example of the diagnosis capability supported by PIPE is illustrated in the following scenario. Figure 3 shows a Master Plumber blueprint file. Suppose that the user examines the output of the blueprint and determines that column `o2` is producing results that are incorrect. The user tries to determine what may have affected column `o2` by querying PIPE:

```
Q: Which fittings created or
 modified column o2?
```

A: Fitting 10. drano created column o2.  
Fitting 12. plisp modified column o2.

The user determines that the o2 column was still incorrect before fitting 12. plisp, so the user wants to determine what columns were accessed by and were used in creating o2.

Q: Which columns were accessed by fitting 10. drano in order to create column o2?

A: Column raraby was accessed by fitting 10. drano in order to create column o2.

The user then continues backtracking through the blueprint to isolate the error:

Q: What fittings before fitting 10. drano modified column raraby?

A: Fitting 9. runstat created and modified column raraby.

By using PIPE in this way, the user can focus his attention directly upon the possibly faulty fittings instead of having to examine every fitting and column.

PIPE also uses the dependency graph to optimize blueprints. Because PIPE can determine which fittings modify which columns in the blueprint, PIPE can determine the last point at which each column is needed in the blueprint. In the example blueprint shown in Figure 3, PIPE makes the following recommendations for removal:

```
never introduce column rim
remove sens_x, sens_y, sens_z and bz
 after fitting 4
remove bx, by after fitting 8
remove rabx, raby after fitting 9
remove bxc, byc, bzc, and stime
 after fitting 12
```

PIPE also provides runtime estimation capabilities. For the optimization example shown above, PIPE estimates that the non-optimized blueprint will take 11:32 +/- 1:04 to run and the optimized blueprint will take 9:58 +/- 0:58 to run.

## Discussion

There are a number of interesting directions which remain open issues. First, PIPE currently assists the user by allowing the user to track the effects of processing steps. A more intelligent system would be able to analyze the data

and extract features which would inform the user as to what processing steps might be useful. For example, a system could examine the data to determine the length of gaps in data and use this information to determine whether gaps in the data need to be filled. A further analysis of the data (rates of change and Fourier analysis) might indicate what types of gap filling methods might be effective. This type of automation requires that the system possess a significantly deeper understanding of the data being processed.

Another aspect of the system is modelling the goals of the processing steps in order to make suggestions about ordering processing steps. Knowledge of the interactions between various processing steps, such as decimation of the data and computation of running averages, could be used to make suggestions on re-ordering of processing steps to improve accuracy or efficiency as needs dictate.

Distributed processing of the data is also an important issue. Because the Master Plumber system operates on data from the Planetary Data System, a distributed database, when a scientist decides to generate a specific data form, there are a number of combinations of processing and data transfer which are possible. Depending upon the data processing steps desired, it may be more efficient to process some data being accessed from a remote site before transferring it to the local site. Factors such as network transfer rates, available computer resources at each site, and current user loads at each site all affect this decision as well as the actual scientific data processing steps.

The current prototype version of PIPE was completed in July 1991. It is implemented in CommonLISP and LISView and runs on Sun workstations.

The C++ operational version of PIPE was completed in May of 1992. and is integrated with Master Plumber and MPTool and is in use by IGPP personnel at UCLA. This version of PIPE incorporated feedback upon the "look and feel" of the interface specified by IGPP personnel.

There are numerous related projects in providing intelligent assistance in scientific computing. The Kineticist's workbench project at MIT [Abelson et al. 1989] targets modelling and analysis of dynamic systems. The SINAPSE system [Kant et al. 1990] assists in construction of numerical models for data interpretation but is specific to seismic models represented as finite difference equations. The Reason system [Atwood et al. 1990] supports analysis of high energy physics data (and is a dataflow system). Finally, the Scientific Modeling Assistant project [Keller 1991] addresses support to facilitate development of scientific models.

## Summary

This paper has described a system to assist in the development of scientific data preparation programs and discussed issues in design for maintainability. This issue of maintainability was particularly important because the processing modules (fittings) are constantly evolving due to changing scientists' needs. In order to maximize

maintainability of the constraint knowledge base, information for each fitting is encapsulated in a fitting knowledge base file and as much as is practical, constraint information is represented in a general declarative fashion.

### Acknowledgements

This work was performed by the Jet Propulsion laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

### References

- [Abelson et al. 1989] H. Abelson, M. Eisenberg, M. Halfant, J. Katzenelson, E. Sacks, G. Sussman, J. Wisdom, and K. Yip, "Intelligence in Scientific Computing", *Comm. ACM*, 32(5):546-562, May 1989.
- [Atwood et al. 1990] W. Atwood, R. Blankenbecler, P. F. Kunz, B. Mours & A. Weir, "The Reason Project", Stanford Linear Accelerator Technical Report #SLAC-PUB-5242, April 1990.
- [Chien et al. 1992] S. Chien, R. K. Kandt, R. Doyle, J. Roden, T. King, and S. Joy, "PIPE: An Intelligent Scientific Data Preparation Assistant", *Proceedings of the International Space Year Conference on Earth and Space Science Information Systems*, Pasadena, CA, February 1992.
- [Kant et al. 1990] E. Kant, F. Daube, W. MacGregor, J. Wald, "Synthesis of Mathematical Modeling Programs", Schlumberger Laboratory for Computer Science Technical Report Number TR-90-6, February 1990.
- [Keller 1991] R. Keller, "Building the Scientific Modeling Assistant: An Interactive Environment for Specialized Software Design", Technical Report FIA-91-13, NASA Ames Research Center, Moffett, Field, CA, May 1991.
- [King & Walker 1991] T. King and R. Walker, "The UCLA Data Flow System," Technical Report #3522, Institute of Geophysics and Planetary Physics, University of California at Los Angeles, CA 1991.