Draco — A Data Reduction Expert Assistant

Felix Yen
Advance Planning Systems Branch
Space Telescope Science Institute
3700 San Martin Drive
Baltimore, MD 21218
yen@stsci.edu

Abstract

Draco is an attempt at partially automate data reduction and analysis. These notes describe our experiences with the project. We discuss the obstacles we have had to face and show how they influenced Draco's design.

Draco is a framework for existing software tools. It uses a minimalist representation system that focuses on the syntax of initializing and invoking programs. There is a working prototype which translates user-defined procedures and primitives into executable scripts appropriate for the user's data. A great deal of emphasis is placed on making sure the user understands what is happening at all times.

Introduction

Motivation

The Space Telescope Science Institute[1] is primarily concerned with operating the Hubble Space Telescope (HST). Like data obtained by other telescopes or other measuring devices, raw HST data contains ***instrument signatures***. A signature is an instrument-specific artifact that can sometimes be measured, e.g. by turning on an instrument without opening its shutter, and subsequently removed from the raw data in a process known as ***calibration***.

Calibrated data may require additional processing before it can be analyzed. For example, one might need to remove cosmic ray noise from the calibrated data before analyzing it. The process which renders raw data useful for analysis is known as ***reduction***. This process encompasses calibration, but the dividing line between reduction and analysis is not well-defined partly because there is no best reduction technique for any given data set. Two scientists could conceivably share the same set of raw data yet reduce it differently because they are interested in different phenomena, e.g. bright stars versus faint galaxies.

In any case, data reduction is a task often consigned to postdoctoral employees, graduate students, and such. It is time-consuming and it often requires a substantial amount of knowledge about mundane things like tape drives, data analysis software packages, and data formats. Relieving researchers of this drudgery is one of Draco's primary goals.

The sheer volume of data being collected today introduces a second, related goal of equal importance. Some projects may not be able to afford to pay people to manually reduce its data in a timely fashion.[2]

Note that the data reduction process is not endemic to the HST or to astronomy. Furthermore, the technology we are about to describe is applicable not only to those domains in which raw data is reduced and analyzed, but to any domain rife with repetitive, menial computations. Please keep in mind that many of the references to reduction and analysis in these notes could just as well be references to arbitrary computations.

Developmental History

Design and development of Draco began in October, 1991. The first prototype, about 3700 lines of Common Lisp, C, and Bourne shell code, was completed and released in August, 1992. Development will continue through most of 1993.

Problems

No Consensus

Our main problem is the astronomy experts' tendency to disagree. There are at least five algorithms for removing cosmic ray noise and some suspect that the proper choice of algorithm hinges on the type of analysis that will ultimately be performed. There are also at least five image restoration techniques that attempt

[1]Operated by the Association of Universities for Research in Astronomy, Inc. for the National Aeronautics and Space Administration (NASA). This work has been supported by NASA under grant number NAS5-31338.

[2]Observations taken with the HST become public after a one-year grace period so scientists have some incentive to analyze their data quickly.

to compensate for the HST primary mirror's spherical aberration. Scientists often wish to experiment with different algorithms in order to discover which ones are best suited for their data.

Neither the older paradigm of procedurally encoding domain knowledge or the newer expert systems paradigm are well suited to supporting this sort of experimentation. Furthermore, both paradigms suffer from other problems. For example, procedurally encoded systems can be extremely difficult to maintain because a single fact about the domain may be encoded in several places. On the other hand, rule-based expert systems can also be difficult to maintain because one finds that the individual facts, when placed in a domain-independent framework, often interact in unexpected ways. There is also the well-publicized knowledge acquisition bottleneck, i.e. determining what the facts are and when one has accumulated enough facts.

The problem could be summed up as follows: the ground-breaking scientist is often unsure about what the facts really are. The conventional approaches, including the expert systems approach, do not work well in such a domain. We feel that AI technology can still provide a solution.

Inertia

A scientist has good reason to be skeptical of new software claiming to replicate the functionality provided by his current analysis software package, especially since his package has probably earned not only his respect but that of his peers. Astronomers are particularly concerned with understanding how their data is being manipulated. This inertia, along with our limited project resources, convinced us that we needed to design a system that can use existing reduction software, and the astronomer's concern for his data convinced us that our system must be able to assure the user that the proper reduction steps have been carried out.

Heterogeneity & Antiquity

There are *many* software tools including several widely-used analysis packages and several data file formats. Integrating all these tools in a single framework will not be easy, especially since few of these tools were designed with this sort of integration in mind. These tools must *eventually* be replaced by a single, coherent, extensible analysis package, but integrating existing packages seems to be the most appropriate solution today.

Solutions

Overview

We make two observations. First, our goal is to optimize a man-machine system of scientists, software developers, and software. The system's purpose is to collect, reduce, and subsequently analyze data. The traditional software engineer attempts to optimize the software part of the system, i.e. make the software as powerful as possible. This does not necessarily optimize the system as a whole (but it is a good way to stay employed). Given the flexibility required by our domain, we feel that our efforts would be better spent developing software which is *not* knowledgeable in the sense that it does not anticipate user demands. What it does do is allow the user to easily adapt his software to the situation at hand. In short, Draco is not an expert system, but an environment that facilitates the integration of existing software tools.

Our second observation: purely syntactic manipulations are often quite useful. Draco can be likened to an algebra or a programming language. One defines a set of *primitives* and is allowed to combine them to form *procedures*. Draco does not know anything about the semantics of the primitives, but it does know which combinations of primitives are syntactically valid, and therefore potentially useful. The user is responsible for insuring that the procedures are truly useful.

Procedures and primitives are abstract entities. The user provides additional information about concrete entities such as primitive *implementations* and the data formats corresponding to these implementations and Draco uses this information to translate a procedure into an executable script tailored to the user's data.

Keeping the User Informed

In order to keep the user at ease, we have made every effort to generate an audit trail as the reduction proceeds. The trail begins with an inventory of the user's data files. This file inventory is generated by a collection of file type *recognizers* and *reporters*. In practice, these recognizers are very thorough; they often read the file in order to determine its type. A sample file inventory is given in Figure 1.

An executable script should produce a log file recording the data reduction or analysis steps performed. Once again, Draco does not know how to produce such a log file; it merely provides a little syntax enabling the user to define his entities so that appropriate entries will be entered in the log.

Integrating Existing Packages

Depending on the packages, i.e. if they all have fairly reasonable command-line interfaces, this can be a surprisingly easy task. Draco makes do simply by storing a character string template for each of the user's tools. For example, a user might define a primitive RCRN (Remove Cosmic Ray Noise) as in Figure 2.

RCRN is defined in terms of its input data type *image-set*, its output data type *image*, and the implementations list following the :concrete keyword. The :reconcile keyword determines how multiple inputs are to be handled. The default value *nil* specifies that multiple inputs should signal an error condition, a value of

Figure 1: Sample file inventory

```
Inventory for /marian/data2/mds
Generated on 02-Aug-1992, 16:32:16
Draco version: 1

GEIS-CALIB-IMAGE files -

w0ul1d03t.c0h =
    FILETYPE:  'SCI             '
    IMAGETYP:  'EXTERNAL        '
    INSTRUME:  'WFPC            '
    FILTNAM1:  'F555W           '
    FILTNAM2:  '                '
w0ul1a04t.c0h =
    FILETYPE:  'SCI             '
    IMAGETYP:  'EXTERNAL        '
    INSTRUME:  'WFPC            '
    FILTNAM1:  'F785LP          '
    FILTNAM2:  '                '

DIRECTORY files -

uparm:              directory

UNIX files -

Draco.inv:          Draco document
mbox:               mail folder
```

Figure 2: Sample primitive specification

```
(define-primitive
  :name            RCRN
  :documentation   "remove cosmic ray noise"
  :input           image-set
  :reconcile       :distributive
  :output          image
  :concrete        (STSDAS-RCRN)
)
```

Figure 3: Sample implementation specification

```
(define-implementation
  :name             STSDAS-RCRN
  :documentation    "a CR removal program"
  :draco-package    IRAF
  :input            IRAF-image-list
  :output           GEIS-calib-image
  :initialize-once  ("stsdas"
                     "wfpc"
                     "combine.logfile=\"-log\"")
  :initialize       ("combine.usedqf=yes"
                     "combine.outtype=r"
                     "combine.option=\"crreject\"")
  :syntax           "combine @-in -out"
)
```

conjunctive specifies that the corresponding implementation should process all inputs at once, whereas the value in the example, :distributive, specifies that the implementation should be invoked once for each input.

One popular way of removing cosmic ray noise is to take several exposures and then average their pixel values ignoring those which are unusually bright. The STSDAS[3] program wfpc.combine[4] can be used to perform this reduction step.

The implementation STSDAS-RCRN (Figure 3) is defined in terms of its analysis package IRAF[5], its input file type *IRAF-image-list*, its output file type *GEIS-calib-image*[6], its initialization command templates, and a template specifying its invocation syntax. In these templates, "-in" represents the implementation's input, "-out" represents its output, and "-log" represents the log file to be generated.

[3]STSDAS = Space Telescope Science Data Analysis System

[4]WFPC = Wide Field/Planetary Camera

[5]STSDAS is layered on the Image Reduction and Analysis Facility.

[6]The STSDAS data file format is known as the Generic Edited Information Set.

An implementation need not be part of a software package. Stand-alone Unix^{TM} programs and Common Lisp functions may also be used to implement primitives.

Conclusions

At this point, it should be clear that our initialization and invocation templates have trivialized the problem of automatically generating calls to existing reduction programs. The moral of our story appears to be that one need not accumulate vast amounts of declarative knowledge in order to create a useful "expert system," even (especially?) if one's domain experts frequently disagree with each other.

The first Draco prototype demonstrates that one only needs a minimalist representation system to harness existing data reduction tools. Instead of undertaking the costly enterprise of declaratively encoding algorithms and other techniques, e.g. iterative image deconvolution methods, we have made use of the vast amount of procedurally-encoded domain knowledge that already exists.

Future Directions

Our code generation problem would be much more difficult if our procedures had more structure. Data reduction procedures tend to be pipelines of programs that do little more than read data and write transformed data. More complex procedures might invoke predicates to control branching or looping. Draco would have to undergo substantial changes if it is to support additional procedural complexity, but these changes seem to be fairly straightforward, i.e. not challenging from a representational point of view.

One bit of complexity that might be implemented shortly is the ability to automatically invoke data format conversion programs in order to integrate software packages that do not share a common data format. Once again, this modification does not appear to be challenging from an AI point of view.

What would be challenging is giving the user the ability to specify a set of analysis goals and then selecting the procedure most suited to these goals. Such ambition would require a much richer representation language, and more time than we can spare.

Acknowledgements

I would like to thank Glenn Miller for his excellent review of earlier drafts of these notes and for his help in designing Draco. Mark Johnston also plays a critical role in this project.