RHESSI DATA ANALYSIS SOFTWARE: RATIONALE AND METHODS

R. A. SCHWARTZ[1], A. CSILLAGHY[2,3,4], A. K. TOLBERT[1], G. J. HURFORD[2], J. McTIERNAN[2] and D. ZARRO[5]

[1]*NASA Goddard Space Flight Center/SSAI, Greenbelt, MD 20771, U.S.A.*
[2]*Space Sciences Laboratory, University of California, Berkeley, CA 94720, U.S.A.*
[3]*Institute of Astronomy, ETH Zurich, CH-8092 Zürich, Switzerland*
[4]*University of Applied Sciences, CH-5210 Windisch, Switzerland*
[5]*NASA Goddard Space Flight Center/L3 Corp., Greenbelt, MD 20771, U.S.A.*

(Received 14 September 2002; accepted 16 September 2002)

Abstract. The Reuven Ramaty High-Energy Solar Spectroscopic Imager (RHESSI) performs imaging spectroscopy of the Sun with high spatial and spectral resolution from 3 keV to 17 MeV using indirect Fourier-transform techniques. We review the rationale behind the RHESSI data analysis software, and explain the underlying structure of the software tools. Our goal was to make the large data set available within weeks after the RHESSI launch, and to make it possible for any member of the scientific community to analyze it easily. This paper describes the requirements for the software and explores our decisions to use the SolarSoftWare and Interactive Data Language programming packages, to support both Windows and Unix platforms, and to use object-oriented programming. We also describe how the data are rapidly disseminated and how ancillary data sets are used to enhance the RHESSI science. Finally, we give a schematic overview of some of the data flow through the high-level analysis tools. More information on the data and analysis procedures can be found at the RHESSI Data Center website, *http://hesperia.gsfc.nasa.gov/rhessidatacenter*.

1. Introduction

The Reuven Ramaty High-Energy Solar Spectroscopic Imager (RHESSI) is a NASA Small Explorer (SMEX) mission to study the acceleration and transport of high-energy particles in solar flares (Lin *et al.*, 2002). Observationally this is accomplished by high-resolution imaging spectroscopy of X-rays and gamma-rays from 3 keV through 17 MeV. This energy range requires the use of indirect imaging techniques. In this case a set of rotating modulation collimators encodes the imaging information into a time-modulated lightcurve (Hurford *et al.*, 2002). A primary function of the data analysis software is to decode this modulated waveform back to a quantitative estimate of the original image. An additional challenge is the recovery of high-resolution spectra that require background subtraction and the deconvolution of the instrument response to yield the incident spectrum as a function of time with ~ 1 keV resolution up to 100 keV (increasing to 5 keV resolution at 17 MeV). Additional data products include the generation of conventional light curves (parameterized by energy), and imaging spectroscopy in the form of feature-based light curves and spectra.

Solar Physics **210:** 165–191, 2002.

This type of imaging spectrometer poses several distinct challenges and opportunities in the design of a data analysis system. The first challenge arises from the large volume of information that must be processed to make either the basic images or spectra. The imaging and spectroscopy both start from a Level-0 database of 4-byte 'photon-tagged events' that encode the detector ID, arrival time, and energy for each detector count, as well as some live-time information (Curtis *et al.*, 2002). (We refer to these events as photon events in this paper, although they include particle events as well.) The data set for some flares consists of hundreds of millions of these photon-tagged events, with rates up to a half million per second. Once calibrated using the aspect solution, the modulated count rates contain information that is equivalent to a set of Fourier components of the source distribution. From that stage, the image reconstruction task is comparable to that in radio interferometry. One helpful factor is that (unlike the case of radio interferometry) the RHESSI instrument response does not vary significantly with time. However, imaging spectroscopy, achieved by the interpretation of data cubes resolved into spatial, energy, and temporal coordinates, does place stringent requirements on the photometric accuracy of the resulting images.

A second challenge is posed by supporting the fraction of the RHESSI user community not familiar with either indirect imaging techniques or with many of the instrumental issues associated with X-ray or gamma-ray spectroscopy. This is particularly important since the RHESSI data are made freely available online within a few days of observation and it is imperative that interested users be able to analyze the data without waiting for secondary databases to be generated. At a minimum, they should be able to obtain lightcurves and to reconstruct images and photon spectra with minimum knowledge about the inner workings of the software. Therefore, it is essential that the user interface provide the option of performing the image reconstruction or spectral acquisition in a manner that does not burden such users with the details of those processes.

A third challenge is that the RHESSI mission relies on data obtained in other wavelength regimes by other instruments to provide the context information on which much of the RHESSI science is dependent. Therefore, special emphasis is placed on enabling convenient comparisons with external data sets. This latter task is eased somewhat by the intrinsic absolute accuracy and stability of RHESSI image locations to ~ 1 arc sec and times to one millisecond.

A unique opportunity is also afforded by the photon-tagged nature of the RHESSI data set. In most imaging-spectroscopy instruments, difficult tradeoffs must be made during the instrument design phase or during operations to optimize the allocation of finite telemetry resources in order to meet the conflicting requirements of imaging field-of-view and resolution, energy range and resolution, temporal coverage and cadence, etc. Since the RHESSI telemetry includes the arrival time, detector, and energy for each detected photon, all of these spectrometer choices can be made during the data analysis phase. Therefore, such decisions can be made iteratively and on a case-by-case basis in response to the unique

characteristics of the solar event under study and in response to the user's particular scientific objectives. This unique capability greatly enhances the scientific return from the observations. *Therefore, a key driver of our data analysis approach is the preservation of this flexibility for use by the data analyst*.

A further opportunity arises because the RHESSI imaging data are self-calibrating to a significant extent. Because of this and the expectation that the calibrations and imaging algorithms will become more refined over time, we anticipate significant improvements to the RHESSI image reconstruction during the course of the mission. The photon-tagged nature of the data set then permits these improvements to be applied retroactively to the entire mission data set.

To maintain the flexibility to choose time, energy, and imaging parameters and to conveniently exploit the anticipated improvements in calibration and algorithms, the generation of extensive secondary databases is minimized. Instead, most scientific analysis begins with the primary database with the most current calibration information. The software applies such calibration data to yield images and spectra with the most appropriate time, spatial, and spectral resolution and spectral range.

Other factors that define the data analysis approach are that the user interface and analysis kernel should support both interactive analysis and multi-event batch processing. It was also deemed a requirement to support both Unix and Windows platforms. Finally, event-driven science, such as flare observations, places an additional premium on the rapid dissemination of both data and software.

2. Meeting the Science Requirements

2.1. Database preparation

Before we discuss the top-level and underlying structure of the RHESSI software, we briefly describe the data products that we archive online to facilitate scientific study. These data products provide the entry point to any scientific study, and help guide the more detailed analyses required by most users. Another category of data products is created dynamically according to the needs of the analyst and we will describe them together with the software architecture in the sections that follow.

The first component of the archived data products is a database of observations and calibrations that can be used to meet the scientific objectives. The primary database includes the following principal elements: packed photon-tagged event lists; Solar Aspect System (SAS) and Roll Aspect System (RAS) data; Monitor Rates; Fast Rate Counter output (when appropriate); and housekeeping data. The Monitor Rates consist of one-second readouts of various detector events that give the operators a picture of the detector status and environment. The Fast Rate Counters contain broad energy-band event rates suitable for imaging during the largest solar flares that might paralyze the normal detector electronics. The SAS and RAS packets contain the aspect sensor data that allow us to know the modulation response at any location on the Sun as a function of time (Zehnder *et al.*,

2002). All of these data are included within our Level-0 data files. Except for some header information to make the primary database appear as a sequence of time-ordered, non-duplicated, and quality-flagged data, the primary database has no 'value-added' content over the raw telemetered data. To ensure prompt preparation, the primary database is generated without routine operator intervention in the Mission Operations Center and Science Operations Center (MOC/SOC). The Level-0 packets are organized into FITS files up to ~ 110 Mbytes in length covering no more than a single orbit between local midnights. During flares, with their higher data rate, there are often multiple files per orbit. The FITS format, for which there is a large and well-tested library of low- and high-level routines, is used to facilitate the documentation and the addition of useful tables within FITS extensions and completely preserves the telemetry packets. The first extension is a table of the packet headers, containing the packet number, packet type, and time range included. As each FITS file is completed by the MOC/SOC, the file is copied to the archive centers at Goddard Space Flight Center and the Swiss Federal Institute of Technology in Zürich, Switzerland (ETHZ) over the Internet and made available there for anyone to download.

Another archival component is the 'quicklook products' (QLP). These allow a quick survey via low-resolution lightcurves, spectra, and images. These are the Level-1 data products created by the MOC/SOC autonomously using the same software used for higher-level analyses. These are also referred to as catalog or summary data. Summary quantities describing the instrument status and spacecraft position, flare positions, flare meta-data, and data quality are included, too. All QLPs are created both as FITS files (or extensions) and browser-viewable image formats such as GIF, PNG, and text files. The SOC posts the most recent QLPs on the RHESSI web pages as they are created. On the web page, any user can easily access the entire QLP archive for either the graphic format or FITS format files. Additionally, a simple summary event catalog will be available for browsing and for searching via a web page. The browser-viewable products are well suited to public access, since no special analysis skills or RHESSI-specific software are required. We have also developed a software interface to the QLPs for use during analysis sessions. (As of August 2002, the QLPs are included with the Level-0 data, but in the near future, the QLPs will only be written to separate files and not to the Level-0 files. This difference should be transparent to anyone using the summary data on the Web or through the analysis software.)

2.2. Software to meet the science requirements

Conceptually the solar X-ray and gamma-ray input to RHESSI is in the form of a data cube dimensioned along four axes: two spatial, one for energy, and the other for time. The action of the instrument transforms this incident data cube of real photons into a time-ordered list of photon-tagged events. In this list, the spatial (more accurately angular) dimensions have been convolved into a temporal

modulation, and the true photon energy distribution has been convolved with the detector response. Although some of the spatial structure of the true data cube can be inferred from the temporal modulation, without the use of high-level software it basically appears only as a temporal modulation. In contrast to this, much of the spectral information can readily be seen in the photon-event lists binned in time and energy. This contrast suggests that the requirement for complete flexibility in the energy dimension for building images makes the data management task even more difficult for imaging. Thus, it is the full imaging spectroscopy task that has driven much of the structure of the data analysis software in order to meet our goal of a robust system capable of producing results for the general scientific user.

One of the first decisions the software team made was to build our system around SolarSoftWare (SSW) (Freeland and Handy, 1998) and the Interactive Data Language (IDL). Both SSW and IDL have a long heritage for solar data analysis, both are used extensively throughout *Solar Maximum Mission* (SMM), *Yohkoh*, and *Solar Heliospheric Observatory* (SOHO). SSW is a package of IDL programming utilities and libraries to facilitate the data input/output, manipulations, and comparisons needed for a data set obtained with a solar instrument. Moreover, most of the scientific software team for RHESSI had extensive experience with both the IDL language and the programming environment created using standard SSW setup procedures. SSW includes installation and update procedures to ensure that users will have the correct and current software environment for RHESSI analysis. Also, because of the wide usage of IDL for other missions, we believe that the commercial license required for IDL imposes a mild constraint on the community compared with the effort that would be required by the team and the scientific community to use any alternative. IDL provides tools to build Graphical User Interfaces (GUIs) which we believe are critical to creating a wide community of users. These are simpler to operate and do not have the burden of syntactic structures. From the beginning, we decided to fully support two of the operating systems supported by IDL, Unix/Linux and Windows, for several reasons. There has been and continues to be a large installed base of Unix servers, there is an increasing number of institutions deploying personal computers using Linux, and there is a large and growing base of users analyzing data on personal computers, in particular laptops, using all variants of Windows since Windows 95. At the time we started, there was minimal support for Windows within SSW. The RHESSI team has been a dominant contributor in making the joint utilization of SSW, IDL, and Windows routine seamless since then. We have found that personal computers running either Windows or Linux make the most cost-effective platform for RHESSI analysis. At present, we do not provide explicit support for the Macintosh operating system because it lacks the environment variables relied on by SSW. We expect that RHESSI software will work on the newest Macintosh operating systems (very Unix-like) supported by IDL version 5.6 and beyond.

The approach of the RHESSI data analysis scheme is to create a robust system that allows the routine production of Level-2 photon-calibrated images and spectra

by any scientist given the Level-0 FITS files, the QLPs, the calibration database, and the software installation. The RHESSI team members interact with the data set in the same way. Recalling our incident data cube of solar photons, the software's task is to take the analyst's specification of the resolution and range for the component desired, i.e., spectrally resolved image, spectrum, or lightcurve, and complete the processing required for the associated data product. Here we list the specific requirements of our software:

(1) The software must deal with large volumes of data, potentially processing nearly four Gigabytes of photon-tagged events to obtain a spectrogram covering a large flare, and do this with the relatively modest resources found on common notebook computers.

(2) The software must unerringly guide the user through the various processes needed to complete a task based only on the properties of the result such as range and resolution along the data cube axes.

(3) The software must be efficient. Some operations are very costly in time and other resources. The results of costly operations should be saved and reused wherever possible.

(4) The software must allow the recovery of intermediate data products to facilitate testing and debugging. This feature also allows building complex chains of processing from smaller and simpler elements.

(5) The action of the software must be highly configurable by user-selected parameters and algorithms, yet require only a few changes from its default configuration to produce meaningful results.

(6) The software must be callable from a command line interface (CLI) to facilitate scripting of repetitive actions.

(7) There must be a graphical user interface (GUI) to reduce the need for an analyst to learn our programming syntax.

3. Object-Orientation in the RHESSI Data Analysis Software

The architecture of the RHESSI data analysis software is based on object-oriented design concepts. In this section we explain what we mean by object-orientation, show how we implement this design technique in the RHESSI software, and show why we believe that the object-oriented approach helps us meet our software requirements.

3.1. Object-oriented software

Object-oriented programming describes a standard way of designing software in which the software is divided into independent parts – the objects – that communicate through well-defined interfaces. Object-oriented design can be seen as a realization of the 'divide and conquer' rule that has been driving software development for the past two decades: complex problems are easier to solve by first

dividing them into smaller, isolated problems. We refer to Booch (2002) for a discussion of object-orientation, or to Schach (2001) for a software engineering approach to the topic. An online resource on object-orientation is given by Appendix Item 11.

Object-oriented design has been used successfully in numerous programs, from operating systems to text editors. Today, it is the main support for the deployment of Web applications. Surprisingly, object-oriented design has been used little for developing data analysis systems in solar physics, even though this field has traditionally been in the lead for innovative computer technologies. We see three reasons for this: (1) Scientific data analysis systems are complex, and object-oriented code is harder to develop than 'procedural' code. These two characteristics collide. The rewards of object-oriented code come in the long term, but short-term considerations often prevail in scientific software development. (2) The vendor of IDL, the main data analysis programming language in solar physics, only recently introduced the object-oriented software syntax, delaying the build-up of experience in this domain. (3) There is a large amount of legacy, non-object-oriented software, available. There is a (false) tendency to think that object-orientation prevents the reuse of this existing code.

Object-orientation does not replace procedural programming, it extends it. Objects are containers for the data types used in an application and for the operations – also called methods – that apply to them (Gamma *et al.*, 1995). The data within an object can only be accessed through a limited number of access methods, which define the interface for that object. This is the principle of *encapsulation*: all operations on a data type are associated with the object that contains this data type. Furthermore, new objects can extend already existing objects. This is the principle of *inheritance*: an object that extends another object inherits all its definitions, i.e., the extended object can use all data types and methods defined in the basis object.

3.2. Object-oriented design applied to RHESSI

The RHESSI software was built to take advantage of the many strengths of object-oriented programming. One drawback is the current unfamiliarity of the syntax to many members of the user community. However, subsequent experience has shown that this is quickly overcome and of course it does not apply at all to those using the RHESSI graphical user interface. Other drawbacks for developers unfamiliar with the programming techniques are discussed in Section 6.

The advantages gained include minimizing the amount of processing required by keeping track of and reusing intermediate products, shielding the user from internal data analysis details (although all intermediate products are available), reusing of code for generic object-handling functions, and supporting both interactive and batch processes easily. The next section describes these advantages more fully. All of these features could be implemented with standard procedure-driven code, but it would be clumsier, require more coding and testing, and probably

TABLE I
A list of the main RHESSI data products.

Type	Object name	Product
Main RHESSI data products	hsi_image	Images
	hsi_lightcurve	Lightcurves
	hsi_spectrum	Spectra
Data products associated with a specific image algorithm	hsi_clean	Images processed by the CLEAN algorithm
	hsi_forwardfit	Images processed by the Forward Fitting algorithm
	hsi_pixon	Images processed by the Pixon algorithm
	hsi_mem_sato	Images processed by the Maximum Entropy algorithm (MEM SATO)
	hsi_memvis	Images processed by the Maximum Entropy algorithm using visibilities
	hsi_bproj	Back-projection images
Intermediate data products	hsi_psf	Point spread function
	hsi_modul_profile	Modulation profiles
	hsi_modul_pattern	Modulation patterns
	hsi_calib_eventlist	Calibrated event lists
	hsi_binned_eventlist	Binned event lists
	hsi_eventlist	Event lists
	hsi_packet	Telemetry packets

burden the analyst with keeping track of parameter changes and how that affects the reuse of intermediate products.

Now we discuss how we apply the concepts of objects to accomplish the goals of the RHESSI software. There are three top-level objects to handle the three primary RHESSI data products – images, spectra, and lightcurves. These three objects employ a set of intermediate data products for each step of the processing cycle: data retrieval, accumulation, time binning, energy binning, aspect calculation, image reconstruction, etc. These data products are listed in Table I. Each data product is associated with an object. Thus, there is a packet object, an event list object, etc. These intermediate objects are created when necessary and stored within the three primary objects. The RHESSI data analysis software can be considered as a chain of objects. To arrive at the final product, a chain of transformation of data products occurs.

Every object in the chain has identical access methods: a *Set* method for setting parameters in the object, a *Get* method for extracting parameters from the object, and a *GetData* method for processing and retrieving the primary data of the object.

A standard interface object called Framework was written to handle these methods so that access to every object (and hence data product) is the same. The framework object is reused (i.e., inherited) by each individual data product object to handle the generic parts of the access methods. The data product-dependent code (the transformation algorithm itself) is embedded into the process method of the specific object.

First, we will outline the steps an analyst would take to create an object and retrieve data from it, and then we will discuss the operation of the objects. We will use image reconstruction as an example (lightcurve and spectrum generation are similar). We proceed as follows:

(1) An instance of an image object is created. This creates a reference (a variable name) to an object and initializes all of the control parameters in the object to default values. At the SSW IDL command prompt, the command used is

obj = hsi_image()

(2) Parameters that need to be changed from the default are set via the *Set* method. We will usually want to set values for the time interval, energy range, spatial resolution, and position of the field of view on the Sun. For example, to set the position of the field of view, the command used is

obj − > Set, xyoffset = [−900., −250.]

(3) The image that corresponds to the current setting of the parameters is retrieved using the *GetData* method. If necessary, the image is processed, following the data transformation chain mentioned above.

data = obj − > GetData()

(4) Parameters that give information about what happened during image generation can be retrieved using the *Get* method:

params = obj − > Get()

There are two general classes of parameters – control parameters that the user sets to specify the data requested, and information parameters that are set by the object to show what happened during the processing of the data. Any of these parameters can be retrieved using the *Get* method. An important feature of getting and setting parameter values in RHESSI objects is that because objects are chained together with framework, the parameters can be set and retrieved from any object in the chain. For instance, we can set the image dimension (associated with the modulation pattern object) in the same way that we set the time range (associated with the event list object), both with the *Set* method of the image object.

Now we return to the chain of transformations of data products initiated by the *GetData* method. In this chain, telemetry packets are transformed into event lists, which are transformed into binned event lists, which are transformed into

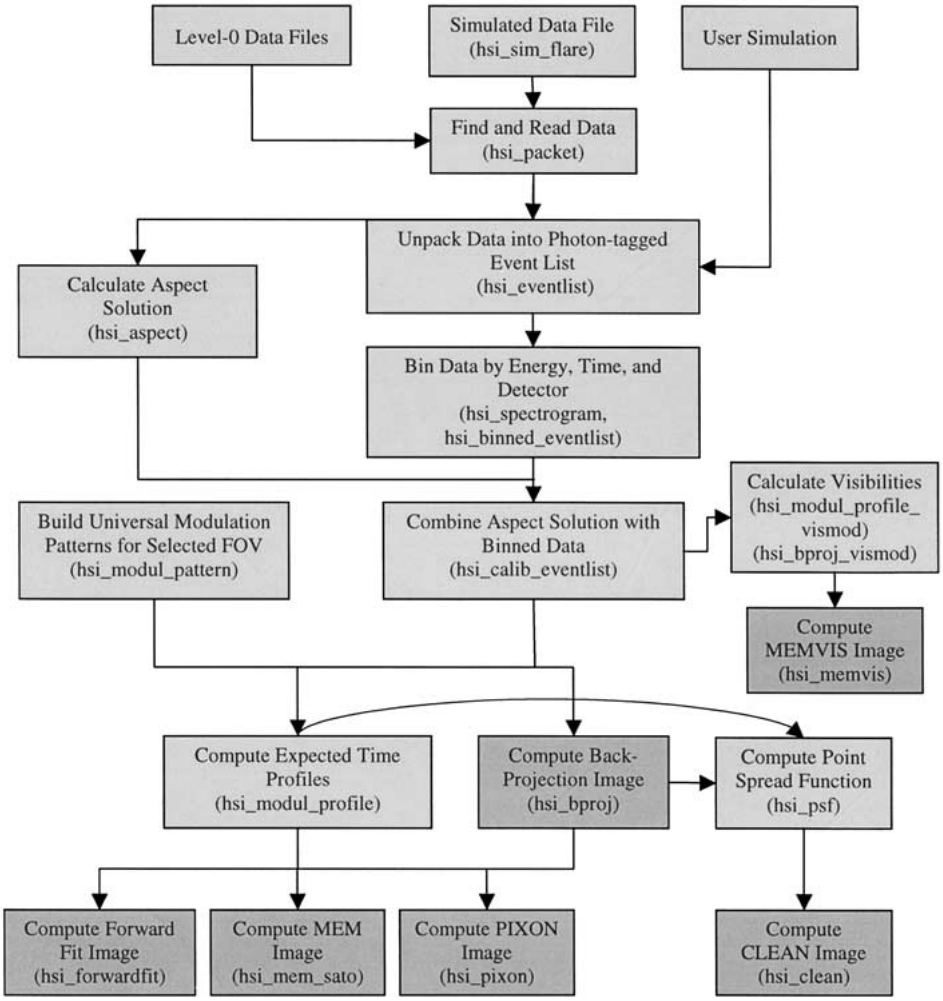

Figure 1. Image creation diagram. This figure shows the chain of transformations for an image reconstruction. The name of the object (or in some cases, program) corresponding to each task is shown in parentheses. The final products are in the darker gray boxes.

calibrated event lists, back-projections after including the modulation patterns, and finally images. This chain is shown in Figure 1.

When an image is requested, the image object checks whether it already contains an image corresponding to the current parameter setting. If it does, this image is returned to the user without any further processing. If it does not, the object needs to process a new image. To do this, it relies on several intermediate data objects. One way an image can be processed is by summing up back-projection maps, the basis of the RHESSI image reconstruction method (Hurford *et al.*, 2002). Thus, the image object relies on a back-projection object, which is responsible for building back-projection maps. Other objects of interest for the image object include those

implementing the different image algorithms (Forward Fit, MEM, Pixon, Clean, MemVIS), as shown in Figure 1.

The back-projection object now proceeds in a similar fashion. It first checks whether it contains back-projection maps corresponding to the parameter settings. If it does, these data are returned to the image object without further processing. If it does not, the object needs to process new back-projection maps. Back-projection maps are processed by building a set of modulation patterns weighted by the binned photon counts. Thus, the back-projection object relies on a modulation pattern object, which is responsible for processing the modulation patterns specific to a spatial and spectral resolution.

This process continues along the chain of objects, from the image to the Level-0 data files, with each object returning either previously-processed or newly-processed data products as required.

As for the access interface, the framework in which these transformations occur is the same for all steps in the chain. It is 'only' the data product in consideration that changes. Thus, the Framework object implements the generic part of the software (therefore, the name Framework). Extensions of the Framework implement the data product-dependent code.

3.3. Benefits of object-oriented design for RHESSI

We believe that the data analysis software is more structured, efficient, maintainable, and extendable by using object-oriented techniques than it would be with traditional procedural software for the following reasons:

Minimized reprocessing: The Framework object also implements the reprocessing policy. Whenever a parameter is set, the Framework checks the parameter value passed. According to the reprocessing policy, the Framework decides whether the data already generated in the object are consistent with the parameter settings. If not, the Framework sets a flag to regenerate the data at the time of the next request. This strategy allows not only for updating the data only when it is necessary, but also for waiting until a request comes to generate any new data product.

Shielded details: Shielding the user from the burdensome details of data analysis is accomplished by enabling the user to access data in a simple and obvious way. Access to a data product should involve as few commands as possible and the analyst should not have to be concerned about the intermediate data products. Objects provide the necessary tools by allowing the simplest user interface one can imagine. Not only are the *Set, Get* and *GetData* access methods available in all objects, all objects use the same implementation of those methods. This technique allows a straightforward access to virtually any byte in the data, while keeping the interface in its simplest form. And though the data analyst is shielded from many of the specific details, every detail of every process can be examined. All of the

intermediate data products can be examined, and due to the interactive nature of IDL any line in any procedure can be found and examined during processing.

Support of interactive and batch processing: Using objects also allows us to accomplish our goal to support both interactive and batch processing easily. The access interface described above can clearly be used for interactive processing in IDL's command-line interface (CLI) and in batch processing through the use of scripts. In addition, we have built a graphical user interface as a layer over the CLI. GUI development was significantly simplified because of the object design of the underlying software. With objects, parameter values and data products are accessed through a single object reference. Within the GUI, the object reference is used to pass parameters and data to individual widgets. Thus, widgets specialize in presenting options for the data analysis task, and delegate data and parameter management to the objects. This architecture allows a clear separation between the widget-based event processing and the data processing. This, on the one hand, avoids duplicating parts of the software, and, on the other hand, makes the localization of software bugs easier; hence, it contributes significantly to the robustness of the system. It also contributes to the power of the GUI by giving every widget access to the entire functionality of the data analysis software.

In addition to the support of our primary objectives, objects also provide support for further development and management of the software during the mission. The object-oriented design is an adequate response to issues in reuse, maintenance, extensions, and combination with other software packages.

Software reuse: The definition of a single interface for accessing any data product provides a strong potential for reusing code. The management of data products (getting and setting parameters, reuse policy) is written only once, in the Framework object.

Software maintenance: The principle of encapsulation allows us to locate bugs efficiently and correct the software with minimal effort. Furthermore, since changes remain localized, bug corrections in one area will not generate an avalanche of other changes in the software. Finally, specific objects can be assigned to specific programmers and the interactions in developing software are considerably easier to manage. This is an important consideration when a software team is spread around the world.

Software extension: Here again, encapsulation helps. New functionality for specific data products can be added by adding new methods to the objects. In addition, completely new objects can be added with relatively few changes in other parts of the software.

Combination with other software packages: Object-oriented software and procedural software are compatible. The RHESSI Graphical User Interface (GUI) is an example; the GUI is not object-oriented itself, but relies on object-oriented software. Conversely, an object-oriented interface can be built on top of legacy (procedural) software. This allows the definition of higher-level objects that can be designed to combine data from several instruments. Software development using this approach includes the Map Objects (Appendix Item 13) and is being considered for new projects such as the Virtual Solar Observatory (Hill *et al.*, 2002).

4. Using the RHESSI Data Analysis Software

Analysts use the RHESSI software either by typing commands through IDL's command-line interface (CLI), or by entering the RHESSI GUI. Using the CLI, analysts create and manipulate the RHESSI objects directly. The CLI offers the full power of the objects, but requires the analyst to be familiar with IDL syntax and RHESSI procedure and parameter names. The GUI shields the analyst from those kinds of details and makes it easy to use most of the functionality available in the objects, but has its own limitations.

4.1. Using the RHESSI objects from the CLI

As mentioned above, there are three main RHESSI objects that provide access to the main RHESSI data products – images, spectra, and lightcurves. They are listed along with their intermediate data products in Table I and are discussed more fully in Sections 5.1 and 5.2.

Another object is provided to access the observing summary and quicklook data. This object and its underlying objects are listed in Table II, and discussed in more detail in Section 5.3.

Because of the structure of the RHESSI objects, every object in either list can be used directly *or* as an invisible part of the chain of objects needed to produce a data product. To use an object directly it can be created interactively by the user as a stand-alone object, or it can be extracted from a higher-level object. The discussion that follows demonstrates the types of operations that can be used on any of these objects.

4.1.1. *Initialization and Methods*

Every object is created with the command obj = hsi_objectname() and has a *Set, Get*, and *GetData* method as discussed earlier. Many objects have additional methods that are specific to the object. For example, most of the top-level objects and some of the underlying objects have a *Plot* method. Appendix Item 2 provides a complete list.

4.1.2. *Object Parameters*

Every object has control and information parameters that are specific to it (Appendix Item 7). As mentioned earlier, the user sets control parameters to specify the data requested; the object sets information parameters to show what happened during the processing of the data. During initialization of the object, all control parameters are set to default values. Users only need to set parameters they want to be different from their default value. When using objects that are part of a chain of objects, users do not need to be concerned about which particular object a parameter applies to. The Framework manages setting the parameter in or getting the parameter from the correct object in the chain. Any number of parameters may be set or retrieved in one command. Some examples of setting and getting parameters follow:

obj − > set, obs_time_interval=[′29-mar-02 21:20′,′29-mar-02 21:30′],

energy_band=[6.,12.]

params = obj − > get(/time_range, /energy_band, /pixel_size)

Setting parameters can be combined with initialization of the object, or with *Get-Data, Plot*, or other method calls. When any method is called, it first calls the *Set* method to set any object parameters that were passed in the call, then retrieves whatever data are necessary to accomplish the function of the call.

4.1.3. *Keywords*

In addition to control parameters, control over the objects is implemented by using keywords in the method calls. Keywords allow us to refine the method calls. It is important to note the distinction between keywords and control parameters, because they are syntactically similar. Keywords apply only to the immediate method call – they do not persist in the object; control parameters are stored in the object and, once set, persist until changed. Some keywords are data product-specific, others are generic. Refer to Appendix Item 2 for a complete description of available keywords. For instance to retrieve all of the information parameters from an object, we use the **INFO_ONLY** keyword:

all_info_parameters = obj − > get(/info_only)

or to retrieve a specific parameter, we can specify that parameter as a keyword. For example, to retrieve the control parameter specifying the image dimensions we want in our reconstructed image, we type:

image_dim = obj − > get(/image_dim)

4.2. GRAPHICAL USER INTERFACE

We have constructed a graphical user interface (GUI) as a layer on top of the objects (Appendix Item 8). The GUI provides an easy way to use the RHESSI software without requiring any knowledge of the RHESSI data objects, or the IDL command names and syntax. To start the GUI, we simply type *hessi* at the SSW IDL command prompt. The main GUI window is essentially a control panel – it provides entry points into more specialized widget interfaces to access the various forms of RHESSI data, while managing the storage, display, and user interaction with plots of retrieved data. It also provides access to the Synoptic Data Archive (see Section 5.4), and manages overlaying those data with RHESSI data.

Because the GUI is built around the RHESSI objects, it has access to the entire functionality of the data analysis software. The GUI manages the user input to the objects and the data extraction from the objects, but all of the data processing takes place within the objects. Therefore, there is no duplication of code, and there should be no difference in results obtained through the GUI and command line.

There are some limitations of the GUI: (1) While the GUI can *potentially* access all the functionality of the objects, it is limited in practice to the options programmed into each widget interface. All of the primary options are currently available, and with time, as we learn which additional options analysts use most frequently, we will add new features. (2) There is currently no scripting capability available in the GUI. Analysts must enter repeated sequences by clicking buttons every time.

A powerful feature of the GUI that allows the GUI and command line to share objects mitigates the first limitation. A combined approach of using the GUI for operations most easily done through the GUI and using the command line for purposes that are more esoteric offers endless possibilities. We will address the second limitation in the future.

The GUI employs a Plot Manager object (Appendix Item 9) written for the RHESSI project that has potential for general usage. The Plot Manager object provides most of the generic plot-handling features of the GUI. All of the different data types retrieved in the GUI (spectra, time plots, images, contours, etc.) are registered in the Plot Manager object, which then uses the same code, regardless of the origin of the data, to allow the user to interactively manipulate the plot, print it, save it in a plot file, etc. This code is not RHESSI specific, and could be applied to data from other missions.

5. Elements of RHESSI Data Analysis Software

Analysts can access all of the data elements discussed below through the IDL CLI or through the RHESSI GUI. Refer to Appendix Items 2 and 7 for a table of methods that apply to each object and the parameters that apply to each object.

5.1. IMAGES AND IMAGING SPECTROSCOPY

Appendix Items 3 and 5 provide step-by-step guides to RHESSI imaging. There are six image reconstruction algorithms available: Back-Projection, Clean, Forward Fit, Maximum Entropy, Maximum Entropy with Visibilities, and Pixon. Hurford *et al.* (2002) discuss these algorithms in detail. Figure 1 shows the chain of transformations required to produce an image with each algorithm starting with the Level-0 data files or a user simulation.

We recall the example given in Section 3.2 to generate an image. Here in addition we set the time range and use the plot method to display the result:

```
obj = hsi_image()
obj -> set, time_range= ['2002/7/23 00:30:00','2002/7/23 00:30:04'],
    xyoffset= [-900, -250]
image = obj -> getdata()
obj -> plot
```

As shown in this example, setting just two parameters – the time range and the position of the field of view on the Sun – and leaving all other parameters at their default setting will produce a reasonable image. Many other parameters can be set however, such as image dimensions, pixel size, detectors, energy range, and image algorithm.

There is an important feature of the objects that we can take advantage of in imaging. Since the different algorithms are each implemented as their own strategy in the same object, the results from each algorithm are stored simultaneously in the object and can be retrieved easily or processed further. This allows us to compare images using different algorithms with full confidence that all other parameters in the object are identical.

There are several tools for performing feature-based imaging spectroscopy. The GUI has options to specify multiple time and energy bins and produce a 4-dimensional image cube FITS file. Once we have created the image cube file, there are several options for examining it: (1) The GUI has an interface for selecting regions of interest and computing quantities such as the flux, centroid, and peak through the image cube. (2) The GUI has an option to show the image cube as a movie either in the time or energy dimension. (3) There is a standalone program called IMSPEC, which computes spectra through the cube for regions of interest. Imaging spectroscopy capabilities within the RHESSI objects are under development.

5.2. SPECTRA AND LIGHTCURVES

Smith *et al.* (2002) discuss RHESSI spectroscopy in detail. Figure 2 shows the chain of transformations required to produce a spectrum or lightcurve starting with the Level-0 data files or a user simulation.

The lightcurve object is based on the same objects and methods as the spectrum object; in fact, the lightcurve object *inherits* the spectrum object. The main difference is that the data returned by the *GetData* method are ordered by time rather than energy.

The RHESSI software offers complete flexibility in selecting energy and time bins for spectra and lightcurves. The bins can be of fixed or arbitrary size.

An important feature of the RHESSI software is that an event list of any size can be processed; the computer should not run out of memory. This means the user can select any duration, even during periods of high photon counts, for accumulating spectra. This is accomplished by reading the data in bunches (a fixed number of data packets), and reusing the memory after processing a bunch. The only limitations are therefore the final size of the spectrum array (which depends on the number of energy and time bins) and the amount of time it takes to read and process the data (which can be considerable).

Normally the spectrum object returns a count rate spectrum. There is also an option to return a 'semi-calibrated' spectrum. A semi-calibrated spectrum does not include a background subtraction and uses only the diagonal elements of the response matrix. It is not a complete solution, but gives a first-order approximation below 100 keV. Another element is needed for complete analysis of the spectra – the spectral response matrix (SRM). There is an SRM object incorporated into the spectrum object that offers varying levels of complexity in computing the SRM, ranging from just the diagonal elements to the full response matrix including all off-diagonal elements.

Using the *FileWrite* method in the spectrum object, we can export the accumulated spectra and the SRM to FITS files. This serves two purposes – it saves results that may have taken a long time to accumulate, and it is the first step in performing complete spectral analysis. Currently we do not fully analyze spectra in the RHESSI object software (and hence the GUI). After exporting the data to FITS files, we proceed with external spectral analysis software to produce more accurate photon spectra and compute best-fit function parameters to the spectral data. Implementing spectral analysis within the object environment (and hence the GUI) is under development.

The external packages we use to analyze spectra are SPEX (see Smith *et al.* (2002) for a description of the process) and XSPEC (Arnaud, 1996). SPEX is a multi-mission SSW/IDL tool written by Schwartz 8 years prior to the RHESSI mission; XSPEC is a FORTRAN-based package written by the Laboratory for High Energy Astrophysics at GSFC. Both packages provide an interface suitable for X-ray spectral analysis of data from a number of astronomical instruments. The RHESSI spectra and SRM are imported into SPEX via FITS files we create from the RHESSI objects. We have not yet built the software to complete the conversion of the spectrum and response-matrix files to XSPEC formats, but this is straightforward using the tools in SSW and could be done on demand. Appendix Item 4 provides a guide to getting started with SPEX.

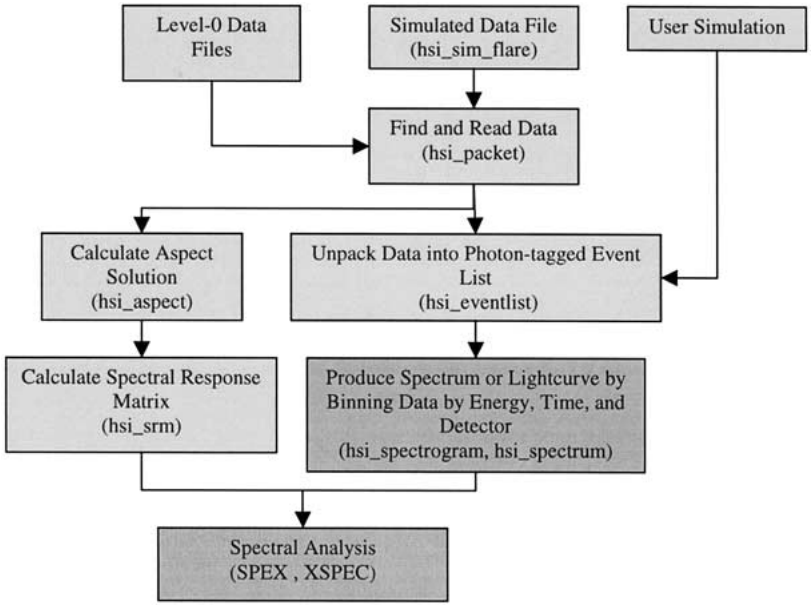

Figure 2. Spectrum and lightcurve creation diagram. This figure shows the chain of transformations for calculating a spectrum or lightcurve.

An example of accumulating a spectrum and writing spectrum and SRM FITS files follows:

obj = hsi_spectrum()

obj − > set, obs_time_interval=′23-jul-2002′+[′ 00:10:00′, ′ 01:20:00′]

obj − > set, sp_energy_binning=14, sp_time_interval=10.

obj − > filewrite, /buildsrm

5.3. OBSERVING SUMMARY INCLUDING QUICKLOOK PRODUCTS

The observing summary consists of the Level-1 data products listed in Table II. Each type of data in the observing summary is stored in a separate binary extension of a FITS file (originally the Level-0 FITS files, but as of September 2002, a separate daily catalog file). Refer to Appendix Item 6 for a complete description of the contents of each observing summary data type, and Appendix Item 10 for documentation on using the hsi_obs_summary object.

5.4. RHESSI SYNOPTIC DATA ARCHIVE

The RHESSI Synoptic Data Archive (hereafter referred to as the Synoptic archive) has its origins in the SOHO Synoptic Archive that was constructed to meet the

TABLE II
A list of the Observing Summary and Quicklook Data Products.

Type	Object name	Product
Main object	hsi_obs_summary	All obs. summ. data
Individual Data Products	hsi_obs_summ_rate	Observing summary rates
	hsi_mod_variance	Modulation variance
	hsi_ephemeris	Spacecraft position
	hsi_qlook_pointing	Quicklook pointing
	hsi_qlook_roll_angle	Roll angle
	hsi_qlook_roll_period	Roll period
	hsi_obs_summ_flag	Obs. summ. flags
	hsi_flare_list	Flare list
	hsi_qlook_image	Quicklook images
	hsi_qlook_spectra	Quicklook spectra
	hsi_qlook_summary_page	Summary page
	hsi_qlook_monitor_rate	Quicklook monitor rates
	hsi_qlook_packet_rate	Quicklook packet rates
	hsi_qlook_soh	Quicklook state of health data

planning and joint observation requirements of experiments onboard the SOHO mission. The SOHO archive is a central database of selected datasets from cooperating space- and ground-based observatories. The database is updated daily and is accessible via Web and IDL interfaces developed and maintained at the Solar Data Analysis Center (SDAC). The RHESSI Synoptic archive is an extension of the SOHO archive to include concurrent observations of RHESSI-observed solar flares at different wavelengths. (Appendix Item 12.)

5.4.1. *The Synoptic Archive*

The purpose of the Synoptic archive is to complement RHESSI observations by providing additional context information (e.g., magnetic morphology and geometry) and physical parameters (e.g., temperature, density, velocity, etc.) that cannot be obtained directly from RHESSI observations. Synoptic archive datasets are selected according to the following requirements. They must

- overlap spatially and temporally with RHESSI-observed flares;
- include different datatypes (image, spectra, lightcurves, spectroheliograms);
- cover a wide range of wavelengths (optical, radio, EUV, soft X-ray);
- be in a standard data format (e.g., FITS);
- be available online for easy access.

Table III lists datasets and their corresponding sources that have been identified to meet the above requirements.

TABLE III

Synoptic archive datasets that complement RHESSI observations. SXI observations are expected to become available January 2003.

Datasets	Source
Magnetogram (including vector)	Mees, NSO, BBSO, MSFC, SOHO/MDI
H-alpha imaging/spectroheliogram/polarization	Mees, Meudon
Optical imaging/spectra	Mees, Meudon, NSO, BBSO, Kanzelhöhe, Kiepenheuer
Radio imaging/spectra/lightcurves	OVSA, Nobeyama, Phoenix, Nancay
Soft/hard X-ray/EUV imaging	Soft X-ray Imager (SXI), SOHO/EIT, TRACE
Soft/hard X-ray lightcurves	GOES, HXRS (Czech)

The task of obtaining and disseminating Synoptic datasets is performed by mirroring data files from remote sites around the world to a central server at the SDAC where they are made available for searching, browsing, and download (via ftp or http) to individual user systems. In order to facilitate the mirroring of datasets, each site is required to store its data in FITS-compliant files using date-based naming conventions. For example, a Big Bear Solar Observatory H-alpha image file with an observation start date of 01:00 UT on 12-December-2002 would have the filename: *bbso_halph_20021202_00100.fts*. To facilitate mirroring, remote files are stored on an anonymous FTP server in directories that are organized by date.

5.4.2. *The Synoptic Archive Software*

The bulk of the original IDL software that supports the Synoptic archive was written several years prior to the launch of RHESSI. The software is fully integrated into SSW and has been tested on current versions of Unix and Windows operating systems using the most recent version of IDL. The software consists of general routines to read FITS files, specific readers for particular instruments (e.g., SOHO/Extreme Ultraviolet Telescope [EIT]), and tools for coaligning and displaying images.

The task of integrating the Synoptic and RHESSI data analysis software systems has been guided by the requirement that the two systems be capable of passing data and relevant parameter information in a seamless fashion. Why is this requirement important? Consider the typical task of overlaying a reconstructed RHESSI image on a ground-based H-alpha image or magnetogram to identify sites of thick-target electron precipitation. This task is accomplished by first creating a RHESSI image using available techniques, finding the relevant H-alpha or magnetic dataset that is nearest in time and spatial location to the RHESSI event, and finally reading and plotting the two images on a common spatial scale.

A typical user cannot be expected to know all the details of module names and procedure calls for different Synoptic datasets in order to perform the above overlay task easily. The solution that we have adopted is to develop object-oriented IDL wrappers for each different dataset stored in the Synoptic archive. For example, the SOHO/EIT data reader is embodied as a read method of an object class named 'EIT', while a SOHO/MDI data reader is embodied as a read method of an object class 'MDI'. The advantage of this formalism is that different synoptic data types can be read into memory using the same *Read* method name. To enhance the utility of synoptic data objects further, we have developed *Set, Get*, and *GetData* interface methods analogous to those of RHESSI objects. With these methods, one can retrieve the underlying image data and associated parameter information such as solar pointing, observation time, etc. In addition to image datasets, we have developed corresponding object-oriented wrappers to lightcurve datasets such as GOES soft X-ray, and *Phoenix*, OVSA, and RSTN radio observations. The lightcurve objects use the same interface method names as RHESSI and, accordingly, can be used to overlay with RHESSI lightcurve observations.

The development of IDL object wrappers for Synoptic archive datasets and software has resulted in several benefits. First, it has minimized the development of new software. Existing procedural code that has already been written and debugged for particular synoptic datasets can be reused as methods of the corresponding object. Second, because they share similar interface method names, synoptic data objects have the same 'look and feel' as RHESSI data objects. Consequently, RHESSI analysis software such as the GUI can conveniently operate on different synoptic datasets using the same method calls as RHESSI objects.

6. Discussion

In the previous sections, we have described the RHESSI data analysis software in terms of its successes. Real software development projects encounter problems, especially when new software methods are employed. In this section, we discuss some of these problems in an effort to help development teams for other projects plan their strategies, as well as to preview some of the improvements we are bringing to our own software.

6.1. Management of the software development

To meet the software challenges posed in Sections 1 and 2.2, we employed object-oriented design methods. Nevertheless, this solution had its own costs, most of which were related to IDL's meager support for objects, and the software team's lack of expertise with objects.

IDL's support for objects is mainly syntactic, so many utilities were needed for our chain of RHESSI objects to function optimally. Consequently, the Framework

and other management objects were being developed and enhanced throughout the software development period, sometimes hindering progress in other areas.

Initially, only one member of the software team, A. Csillaghy, was versed in object-related methods and philosophy so all of the object-specific development flowed through him. There was clearly a learning curve as additional team members became familiar with some of the principles so they could successfully code and debug modules without his assistance.

An area where the lack of experience hurt was in defining the correct object structure and determining when objects should be used instead of ordinary procedures or functions. As an example of the former, we have found in hindsight that the SRM (spectrum response matrix) object and the SPECTRUM object have the wrong relationship in the processing chain. As written, any change in SRM parameters would cause reprocessing of the SPECTRUM object, when in fact the two objects are mostly independent. We have modified the SPECTRUM object's reprocessing policy by writing special cases to prevent the extra reprocessing; however, if the objects were properly related that would not be necessary.

The example above illustrates the need to build software not from a functional point of view but from a data-modeling point of view. We believe that the best way to design software is by first understanding the data products that are involved at every data processing step. Defining the correct data products makes it possible to infer the software functionality from the data. Nevertheless, many software developers already think in terms of data products, but program in a procedural way. For them, the shift to object-oriented programming is reduced to the adoption of a few syntactic structures that will *help* them in their development, and increase their efficiency. Had we understood this at an earlier stage our software development would have been faster and more robust.

Programmers also need experience to decide when an object class is required to supplant procedural methods. For this we have found that a figure of merit is obtained in the product of the number of times a result is accessed by the time it takes to produce a result. As that product becomes a larger proportion of the overall processing time, an object with clear rules that determine reprocessing should probably be utilized. We have identified several procedures within the SPECTROGRAM classes related to energy and time binning that would perform better using our Framework object.

For future projects requiring complex data analysis methods, we anticipate that object methods will become a necessity rather than a choice. In this context, our experiences should prove educational. Studying several concepts and techniques used in our software might be beneficial: (1) Our basic template for object development (Appendix Item 1) should be examined for its simplicity and reusability. Using that template, a simple processing chain could be built to gain experience. (2) We use several techniques to encode variety and exceptions within the object classes. For example, we use the STRATEGY_HOLDER class to achieve differentiation in our image processing. This technique allows us to process images

using different strategies (image algorithms) but with identical input and output structures, and with the maximum reuse of intermediate data products. Another example is the variety achieved in the SPECTROGRAM objects by using a post-processing routine. The SPECTROGRAM objects (SPECTRUM, LIGHTCURVE, and BINNED_EVENTLIST) start with identical data and are processed through the inherited SPECTROGRAM process method, but then each apply their own additional processing routines. These techniques are based on standard software engineering methods, the design patterns, described by Gamma *et al.* (1994). Our innovation is in applying them to scientific data analysis. We believe that design patterns have a significant potential of application in scientific data.

6.2. Lessons learned post-launch and anticipated changes

How well were our pre-launch expectations for our software and the deployment of our databases met? As expected, the RHESSI hardware behaved somewhat differently in space than on the ground. The main unanticipated problems that required immediate software solutions were data dropouts (Smith *et al.*, 2002) and the need to use the PMTRAS data for roll aspect (Hurford and Curtis, 2002), both of which required non-trivial solutions. In addition, our original plan for organizing the data files was flawed. We originally attached Level-1 Quicklook Products (QLPs) to the Level-0 FITS files. This meant that any change in the QLPs necessitated rewriting the entire Level-0 database, and propagating all of the rewritten files to mirror sites and individual local archives. The files are large and the time to regenerate and propagate them became prohibitive. As of September 2002, we are removing the QLPs from the Level-0 FITS files, and appending them instead to a much smaller daily catalog file.

It was necessary to address the issue of compute-time, particularly since users were starting from a photon-based data set and since many algorithms are intrinsically iterative. (Maximizing re-use of intermediate data products is valuable in this context.) Our initial efforts were directed at speeding the image reconstruction task, where one to two orders of magnitude were saved by algorithm innovations (Hurford *et al.*, 2002). Speed is not a dominant issue with the current package except for large flares where hundreds of millions of photons must be processed. We believe that we can significantly reduce the processing time needed to build the spectrograms by processing the event list once and saving the results in a file. The size of this file will depend on the flare X-ray intensity and the user-selected time range. Further processing in the same time and energy range will then use this file instead of the Level-0 file.

Simulation software for RHESSI played several roles during the development phase. Simulation output could be expressed either as an event list, a simulated Level-0 data file or as a series of timed 'photon' stimuli into the flight hardware. This permitted testing of data handling and image reconstruction software as well as end-to-end testing of hardware/software systems. After launch, simula-

tion software is expected to provide a tool for evaluating the robustness of image reconstruction. Documentation on how to use the simulation tools can be found in Appendix Item 5.

The simulation software was based on many of the same geometry modules used for image reconstruction. While this enabled significant savings in the effort required to develop the simulation capability, it compromised the value of simulations for the discovery of some classes of 'geometrical' bugs since errors in simulation were exactly compensated during analysis.

Another area in which our preparations were incomplete is the tracking of instrument configuration changes. The software team was unaware that complete housekeeping information was not included in the telemetry stream, and consequently made no provision for compiling logs of changes in housekeeping information. We are now compiling these logs, but a better solution would have been to have this information encoded in telemetry.

A series of ongoing hands-on workshops provided a combination of presentations by software developers and hands-on experience for potential users of RHESSI software. This not only represented an additional channel by which to reach potential users, it also provided useful feedback to the software development team. The workshops were an integral part of our documentation strategy and provided the impetus for the creation for many of documents found on the RHESSI datacenter website. We believe these workshops to have been successful and to have enabled many scientists to begin scientific analysis with RHESSI data within a few weeks after the launch.

Finally, we expect to improve the RHESSI analysis software through the life of the mission. Improvements will be made to the structure of our objects, to the speed of our process methods, and to the quality of our scientific algorithms. The focus of our efforts will be, as always, to maximize the scientific return from the RHESSI project.

Acknowledgements

We are grateful to M. Aschwanden, P. Bilodeau, A. J. Conway, M. Fivian, S. Krucker, T. Metcalf, E. Schmahl, J. Sato, and D. M. Smith for significant contributions to the RHESSI software package. We also thank R. Bentley, B. R. Dennis and C. Johns-Krull for their valuable insights and roles in documenting the software. Without contributions such as these, the package would not exist. We also thank R. P. Lin for his support in allocating sufficient resources and priority to this task. This work is funded by NASA grant NAS5-98033-05/03. A. Csillaghy is partially funded by the Swiss National Science Foundation (Grant No. 2000-061559) and UAS Grant No. FHA-02-07-007.

Appendix

The main RHESSI Software Web site is located at
http://hesperia.gsfc.nasa.gov/rhessidatacenter.

The following is a list of Internet documents referred to in this paper. The URLs given for the references are relatively stable, but in the nature of Web documents, could change. The CD included with this paper contains the top-level page of the current versions of these Web documents.

1. Csillaghy, A.: Object-Oriented Data Analysis Software Concepts
   URL: http://hessi.ssl.berkeley.edu/software/hessi_oo_concept.html
   A description of object-oriented concepts, especially as applied to HESSI Data Analysis.

2. Csillaghy, A.: RHESSI Objects Reference Manual
   URL: http://hessi.ssl.berkeley.edu/software/reference.html
   Tables of the classes, keywords, and methods used in the RHESSI object construct, as well as descriptions of each object class.

3. Dennis, B.R.: RHESSI Imaging – First Steps
   URL: http://hesperia.gsfc.nasa.gov/∼dennis/imaging/first_steps.htm
   A beginner's tutorial-style guide to using the GUI to create images. Includes instructions for each step starting with generating a lightcurve through creating a back-projection image, and provides screen snapshots of the widget interfaces and the expected output.

4. Dennis, B.R.: RHESSI Spectroscopy – First Steps
   URL: http://hesperia.gsfc.nasa.gov/∼dennis/spectroscopy/first_steps.htm
   A beginner's tutorial-style guide to using the GUI to generate spectra, and using SPEX to analyze spectra. Provides screen snapshots of the widget interfaces and the expected output.

5. Johns-Krull, C.M.: An Overview of the Command Line Interface for RHESSI Data Analysis Software
   URL: http://hessi.ssl.berkeley.edu/∼cmj/hessi/doc.html
   A thorough tutorial-style guide to using IDL's command line interface to run the RHESSI software including imaging, spectroscopy, simulations, and access to quicklook data. Numerous examples demonstrate the syntax and commands to use and output to expect.

6. McTiernan, J.: RHESSI Quicklook Data Products
   URL: http://sprg.ssl.berkeley.edu/∼jimm/hessi/hsi_obs_summ_soc.html
   A description of the data and info structures returned for each class of observ-

ing summary data.

7. Tolbert, K.: RHESSI Data Object Parameters
URL: http://hesperia.gsfc.nasa.gov/ssw/hessi/doc/hsi_params_all.htm
A table listing all control and information parameters for all RHESSI objects. Includes a brief description of each parameter, its default value, and the name of the object that uses it.

8. Tolbert, K.: HESSI GUI Guide
http://hesperia.gsfc.nasa.gov/ssw/hessi/doc/gui_help.htm
User's Guide to the HESSI Graphical User Interface.

9. Tolbert, K.: Plot Manager (PLOTMAN)
URL: http://hesperia.gsfc.nasa.gov/ssw/hessi/doc/plotman_help.htm
A user's guide to the Plot Manager, a generic plot-handling package.

10. Tolbert, K.: Using the Observing Summary and Quicklook Data
URL: http://hesperia.gsfc.nasa.gov/ssw/hessi/doc/obs_summ_access.htm
A guide to using the hsi_obs_summary object to retrieve and plot the observing summary and quicklook data.

11. Usenet: Object FAQ
URL: http://www.cyberdyne-object-sys.com/oofaq2/
A complete explanation of the concept of object-orientation, its background, why it should be used, and how to implement it.

12. Zarro, D. M.: The RHESSI Synoptic Data Archive
URL: http://orpheus.nascom.nasa.gov/∼zarro/synop
A description of the RHESSI Synoptic Data Archive, an example of a database input and search engine, a tutorial on overlaying images from different sources, and a user's guide to reading and plotting GOES data.

13. Zarro, D.M.: IDL Software for Analyzing Solar Images
URL: http://orpheus.nascom.nasa.gov/∼zarro/idl/maps.html
An explanation of map objects, how to create and manipulate them, and how to apply them to solar images.

References

Arnaud, K. A.: 1996, in G. Jacoby and J. Barnes (eds.), *Astronomical Data Analysis Software and Systems V, ASP Conf. Series*, Volume 101.

Booch, G.: 2002, *Object-Oriented Analysis and Design with Applications*, 3rd edition, Addison-Wesley, New York.

Curtis, D., Berg, P., Gordon, D., Harvey, P. R., Smith, D. M., and Zehnder, A.: 2002, *Solar Phys.*, this volume.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J.: 1995, *Design Patterns: Elements of Reusable Object-Oriented Sofware*, Addison-Wesley, New York.

Hill, F., Csillaghy, A. *et al.*: 2002, in A. Szalay (ed.), *EGSO in Need for a Global Schema, Proc. SPIE* **4846**, in press.

Hurford, G. J. and Curtis, D. W.: 2002, *Solar Phys.*, this volume.

Hurford, G. J., Schmahl, E. J. *et al.*: 2002, *Solar Phys.*, this volume.

Freeland, S. L. and Handy, B. N.: 1998, *Solar Phys.* **182**, 497.

Lin, R. P. *et al.*: 2002, *Solar Phys.*, this volume.

Schach, S.: 2001, *Object-Oriented and Classical Software Engineering*, 5th edition, McGraw Hill, New York.

Smith, D. M. *et al.*: 2002, *Solar Phys.*, this volume.

Zehnder, A. *et al.*: 2002, in Szalay (ed.), *Proc. SPIE* **4853**, in press.