Plan for the ARCS Software Project

Brent Fultz and Michael Aivazis
*California Institute of Technology, Pasadena CA 91125 USA*
btf@caltech.edu, aivazis@caltech.edu

February 4, 2003

**Abstract.** A plan is presented here for the ARCS software project, with descriptions of the mission, scope, capabilities, architecture, implementation, schedule, and resources. The scope of this project is consistent with the scientific missions of ARCS and inelastic chopper spectrometers with similar characteristics. The software architecture is based on the data flow paradigm, where analysis is performed with reusable computational components that are connected to each other with data streams. The software architecture could serve other instruments too, and the difference in scope of such an effort is explained. For ARCS, however, the software project will provide a complete solution to the data analysis and visualization needs, with an interface for theory and simulation.

**Keywords:** neutrons, software, software architecture, data flow, distributed computing, data analysis, SNS

1. Mission Need

Neutron scattering research in the United States is poised for a spectacular leap after the Spallation Neutron Source (SNS) is commissioned in 2006. With the power of its source and the efficiencies of its instruments, the SNS will provide improvements in scattered neutron flux of typically two orders of magnitude. Such an improvement is particularly important for inelastic neutron scattering, where experiments to date have been invariably constrained by low flux and detector efficiency. The ARCS instrument, a wide-Angle Range direct-gometry inelastic Chopper Spectrometer, located at the high power target station of the SNS, will solve these countrate problems for traditional experiments performed with chopper spectrometers. It will open the door to new types of experiments, and it will encourage more thorough analyses of its data, which are expected to be of higher quality and more complete than we have yet seen from Fermi chopper spectrometers.

Data from inelastic chopper spectrometers can reveal the fundamental dynamical processes in materials or condensed matter, but not without a substantial data analysis task to produce a basic graph of \( S(\vec{Q}, E) \), the intensity as a function of momentum and energy transfers. Software to do this has become available in the past couple of years, and software to obtain \( S(|\vec{Q}|, E) \) has been available only somewhat longer. These computer codes are not readily integrated into larger data analysis packages, however, and for architectural reasons such integration is impractical. Increasingly, however, cutting-edge science requires software of greater flexibility, especially when experimental data are to be examined from different viewpoints, and compared to results from theory or models. For example, detailed simulations of the structure and and dynamics of materials are now possible with *ab initio* calculations, and a real opportunity exists to build an interface between these simulations and neutron scattering experiments. It is difficult to integrate software for data analysis with software for theoretical modeling when they are standalone.
applications running on specialized platforms. Likewise, it is difficult to use different software or hardware tools for the visualization of data when doing so involves the mechanics of data conversion and transport to different computing platforms.

These issues of software flexibility have led us to a new architecture for the ARCS software. The concept for the ARCS software architecture includes interchangeable, modularized, software components that operate on data streams. Each component encapsulates and controls a core code that could otherwise have been written as a standalone application. The components negotiate their communication through data streams, which could be sent across a network. This architecture enables distributed computing, and is ideally suited to hardware architectures on high-performance networks such as the TeraGrid project supported by the NSF, in which Caltech is an active participant. In practice, users will direct the processing of their data through a web browser by accessing a central server, which distributes data and computing jobs to the appropriate resources. This data analysis architecture could be used by all instruments at the SNS, or all neutron sources in the U.S., and in principle, throughout the world. This would require an effort well beyond the scope of the ARCS software project. The Appendix explains the extension of scope that would be required for this larger effort.

2. Scope of Services for Data Analysis

2.1. Data Reduction

A schematic of the ARCS software is presented in Fig. 1. This figure, first presented as a roadmap in Sept. 2001, shows three paths for extracting scientific results from the raw data in the upper left corner.

The first path, horizontally across the top, is the traditional approach to data reduction and visualization. The goal of this analysis is to obtain the intensity as a function of momentum transfer and energy transfer, $S(\vec{Q}, E)$. To do so, the data arrays of counts acquired in terms of instrument parameters such as detector pixel and arrival time must be converted into normalized intensities with physical units such as Å and meV. Instrument backgrounds and other distortions must also be removed. Occasionally it is necessary to correct for other distortions caused by, for example, multiple scattering or multiple excitations.

Today all software for inelastic scattering has breaks in the horizontal chain across the top of Fig. 1. These breaks require operator interventions such as transporting data into different operating systems. The ARCS project will provide a set of components that can be arranged to extend continuously across all data analysis steps. Furthermore, components in the chain can be rearranged to test different processing algorithms. It is also important that data streams can be piped into visualization windows for inspecting the data after the different steps of data processing. An example of a component for energy rebinning of data from a chopper spectrometer is described in the context of the data stream architecture in Sect. 3.
2.2. Modeling

The second and third paths for extracting scientific results from experimental data are designed to connect experimental data to theory or analytical models.

The second path, the vertical chain in the center of Fig. 1, is for comparing experimental results to models of sample dynamics. This path is especially appropriate for analytical models with adjustable parameters. Our group at Caltech now uses this path routinely for work on phonon dynamics. For example, a measured $S(Q,E)$ from an elemental polycrystalline sample can often be converted into a phonon density-of-states using a thermal correction procedure. On the other hand, a phonon DOS from a binary compound cannot be obtained from the measured $S(Q,E)$ because the different elements in the compound do not scatter neutrons with equal efficiencies, causing a “neutron-weighting” of the experimental spectra. Without knowing the lattice dynamics of the compound, it is impossible to know the distortions of an experimental DOS obtained from a measured $S(Q,E)$ after a thermal correction procedure.

The phonon scattering efficiencies of the different atoms are well known, so a lattice dynamics model can be used to calculate an experimental spectrum. We do so with an iterative procedure where the force constants in the dynamics model are varied to obtain the best fit to the experimental data. We therefore can extract both the force constants and perform the “neutron weighting” correction to obtain the true phonon density-of-states from the measured $S(Q,E)$ of the compound. This is now a routine procedure for us, and new experiments can be designed around this capability. Over the next few months this code for iterative analysis needs to be better integrated into the ARCS software architecture. Over the next year the algorithms need to be extended to utilize the momentum transfer information available from a pixelated detector array on an instrument like Pharos or ARCS.

Besides phonon dynamics in ordered compounds, there are many dynamics models that can be compared to experimental data. Applications include data corrections, the determination of parameters such as force constants or exchange stiff-
nesses, or testing if the model is in fact consistent with the measured $S(Q, E)$. We will implement a set of four standard types of dynamics models in the baseline design for the ARCS software project.

1. lattice dynamics with a Born–von Kármán model (periodic structure)
2. spinwave dynamics with a Heisenberg hamiltonian on a periodic structure
3. lattice dynamics on a disordered structure, using a moments analysis of the dynamical matrix
4. spin dynamics in a paramagnetic model

These four dynamics models are established sufficiently well to justify their inclusion as the initial set of models in the ARCS software. Adding additional models in the future should be easier once we have some experience with these four, but these four are included in the project baseline. It is possible that the ARCS IDT will request additional models, and their inclusion would be considered on the basis of available resources in the project and the scientific need.

2.3. DIRECT EXPERIMENT SIMULATION

The third path from data to science is shown on the left of Fig. 1. This approach is a direct simulation of the data measured at the detectors. It is based on Monte Carlo codes that are used in the neutron community for simulation of instrument performance. These codes have been tested and validated against the performance of real instruments. We will put these Monte Carlo simulations together with molecular dynamics simulations to perform direct simulations of experimental data. A number of molecular dynamics simulations are available to the theory community. These simulations are complementary to the analytical models of Sect. 2.2, and are sometimes advantageous. For example, no approximations are needed when implementing structural models of disordered solids, which are not handled well by the methods of Sect. 2.2 (except when the $Q$ information is ignored as in methods 3 and 4).

One possibility is to include the sample into the Monte Carlo computations as a component of the instrument, transforming an individual incident neutron into a neutron scattered into the detector array. Alternatively, simulation results for the primary flight path could be stored and used for several simulations. For lattice dynamics simulations, we are tentatively planning on treating the scattering as individual events in the first Born approximation. This is equivalent to sampling the velocity-velocity correlation function of the atoms in the sample. These motions will be obtained from a molecular dynamics simulation, embedded as a core in a Python component (core and component are defined in Sect. 3). After we gain experience with this direct simulation of data with lattice dynamics, we plan to develop this same path for spin dynamics simulations. Although the exchange forces are known with sufficient detail to implement such a simulation, there are no standard packages to simulate spin dynamics. The spin dynamics simulations will require development either by us, or in collaboration with a group working in this field. This effort is at the edge of the scope of the ARCS software project.
3. An Architecture for Distributed Data Analysis

Most common data analysis tasks can be cast as the results of components acting on data streams, very much like the electrical engineer’s concept of a signal processing system. As an example, consider Figure 2. The module NeXusReader is responsible for reading a file that contains raw data and instrument information in some standard format and converting it into a data stream. This stream is fed to three filters, each of which selects a particular subset of the information in the stream, such as instrument information, arrival times and raw detector counts. The outputs of the three filters are data streams that are in turn fed to Bckgrnd, which corrects for the instrument background. The conversion to a histogram of intensity as a function of energy is carried out by Energy, which requires details about the instrument, the times of flight and the background-corrected counts from Bckgrnd. In this example, the resulting stream is fed to NeXusWriter for storing in a file, but one can easily imagine that other components, perhaps those of a visualization system, might be involved in the further manipulation of the data stream. Similarly, the input stream in this example was generated by reading a file. However, one can easily envision an interface to the instrument that makes the data available as it is collected by an experiment in progress.

We plan to organize the data analysis software using this data flow paradigm as the basic abstraction. The proposed architecture is a set of services that enable the encapsulation of the computational engines, establish the transport of data between these engines, and provide uniform access to user input. The framework shields the computational engines from the user interface, allowing the construction of interfaces that are suitable for a variety of end-user environments. Advanced user interfaces will allow the direct manipulation of the analysis network and provide visual ways to interact with the components and display the results of the analysis.
The remainder of this section describes in detail some of the framework services and their interactions.

3.1. COMPONENTS

A component is the architectural element that acts as the mediator between a low-level module and its environment. We will refer to these low-level routines as the component’s core, as illustrated in Figure 3. A component is responsible for the initialization of its core, which may involve delivering user-supplied information. We will refer to such information as the component’s properties. When a component is instantiated, it provides the framework with a table of pairs of strings with property names and default values. The framework makes this information available to the user interface, which is responsible for soliciting the user’s input. The property table is read at component execution time. It is used to initialize the core by converting each property string to its native representation. This mechanism bypasses much of the complexity of component initialization and allows for the complete decoupling of the core from the user interface.

Components interact with each other by exchanging information in data streams. Streams are a useful abstraction because they promote a weak, standardized coupling between components. For each application domain, there is typically only a handful of different data types that are exchanged between components. For the analysis of neutron scattering data, the majority of data exchanges involve tables and generalized histograms, described in more detail in Section 3.2. Components negotiate the actual details of the data exchange when they are first connected to each other.

When a stream delivers data at the input port of a component, the component must make the data available to its core. Conversely, the component must place results in the data stream connected to its output ports. The details of this process depend rather strongly on the choices of programming languages, operating systems and compilers. Fortunately, these platform dependencies are rather generic and can be handled as part of the services offered by a component.

Typical component cores manipulate large data sets and can be rather computationally intensive. They will be implemented in low-level languages. Python bindings are most easily written for component cores that are written in C++, but some support for FORTRAN codes will be provided when this is easier than rewriting...

---

1 This implies delegating part of the task of resolving the platform dependencies to the configuration management system.
FORTRAN codes in C++. Although it is desirable to provide a forward migration path for legacy codes by embedding them as component cores, within the ARCS project this may prove too big an effort in many cases. On the other hand, we intend to provide explicit support for parallel programming, in anticipation of the increased computational complexity necessitated by state-of-the-art scientific analyses.

3.2. Data Streams

Data streams are the conceptual encapsulation of the mechanism for data exchange among components. Streams can be thought of as single-port components that merely copy data present in their input port to their output port. They are an essential architectural element because they enable the decoupling of components from one another and hide the details of the data transport mechanism.

The conceptual decoupling of components from each other through the use of streams enables their physical decoupling. This makes it possible to distribute the computation among multiple process spaces. The user can choose whether components reside in the same process space, as separate threads, as separate processes on the same machine or are deployed across the network. Connections that cross process boundaries can discover where the components they connect are physically deployed and then determine an appropriate mechanism for data transfer. The choice is completely transparent to the components themselves and therefore the control and data transfer mechanisms can be implemented independently by an expert. Further, one can take advantage of emerging protocols and services for distributed computing without disturbing any of the existing components.

Connections between components are established at run time when one cannot rely on compilers to ensure that an input port receives data compatible with its expectations. A frequently-used (but extreme) solution is to bypass the problem by allowing streams to carry only one data type, such as three-dimensional arrays of doubles. At the other extreme is a solution commonly employed by industrial-strength component systems that allows components to exchange arbitrary data types. A specialized language allows components to describe the types of data streams they consume or generate, providing in essence a run time typing system. The first extreme is rather restrictive, while the latter is fairly complex to implement and maintain. Instead, we propose a compromise that takes the requirements of our specific application domain into proper account and restricts the data stream types without sacrificing generality.

We have identified two abstract types, tables and histograms, that appear to be sufficiently general to satisfy the data exchange needs of our components for the analysis of neutron scattering data. Tables are inspired by the database concept of the same name. Potential uses include the storage of raw events in a detector. The data are conceptually organized in rows, each of which has the same number of named columns. A description of the table stores the name and data type of each column, along with the number of rows, if known. This information constitutes the table meta-data. Histograms are a generalization of multi-dimensional dense arrays of floating point numbers and are the typical structure for exchanges between components that manipulate processed data. The histogram meta-data consists of the number of dimensions plus an array for each dimension that describes the histogram...
bins. Other data types may be added in the future if enough components exchange information not expressible as histograms or tables.

There is a clear separation of large data sets from the lightweight information that describes them. Data storage and data transport are entirely opaque to the data producers and consumers, and these data handling functions are considered implementation details that differ for each platform. Transport of the actual data is considered an expensive operation, since it scales with the amount of data in the stream, whereas the meta-data can be exchanged freely between components. We will construct a simple stream description language based on XML to allow components to explain to each other the content flowing in the data streams connected to their ports. The meta-data will be part of the negotiation protocol for component connections and will be used to issue proper diagnostics when irreparable incompatibilities are detected.

4. Implementation

4.1. Software Structure

Modern object-oriented programming allows for a beautiful mapping of physical concepts into software objects. This elegance is a priority for software development in the ARCS software project. With clear connections to physical concepts, the code can be much better understood, maintained, and modified in a professional way. We have set high standards and have spent much effort to endow software objects with physical significance, and arrange them with the hierarchies and interrelationships of scientific concepts. Using physical concepts as principles for software organization means that scientists must write the code. In general it is easier to teach scientists object-oriented programming than to teach programmers quantum mechanics, although exceptions do occur.

4.2. Programming Languages

The ARCS software comprises components in Python, a modern very-high-level computer language. Component cores are written in C++ with Python bindings. These C++ cores are then accessible as Python functions, but they perform as compiled code. The Python interpreter makes it easy to arrange components into custom scripts without recompilation. Interpreted Python scripts can therefore provide users with great flexibility in constructing custom analysis procedures from a well-stocked software toolkit. Scripts for common types of data analysis will be nested into standard data analysis packages, but their customization should be readily possible. Some cores will be adapted from legacy codes written in FORTRAN either directly, or by rewriting them in C++ It is unclear how much support for FORTRAN programming can be provided by the ARCS software effort. A larger software engineering effort such as DANSE (see Appendix) could provide the tools needed for a better support of FORTRAN.
4.3. DISTRIBUTED COMPUTING

Data analysis is performed as a service accessed through an Internet web site. Figure 4 shows the architecture. This distributed computing architecture allows the components to be maintained and run centrally on well-tested platforms. A compute server, not the user, arranges for computation on the appropriate hardware. A user could elect for little code and no raw data to reside on his or her local computer, while still directing the reuse and reconfiguring of the Python components, including those that require specialized hardware such as Beowulf clusters. Our intent is to ensure that any user can utilize the highest-performance hardware without buying and maintaining it.

Today the computing resources are located at Caltech, where we are relatively free to experiment with them. The essence of a working web portal and user interface have been demonstrated. The user logs on to our web portal through a standard browser such as Internet Explorer, and receives a Java applet that runs under the browser. The present applet provides a “Labview-like” graphical user interface (GUI) where the user “wires together” a set of data analysis components. Changing the present GUI is not difficult because of a clean separation between the user’s depiction of the data analysis procedure, and the server’s execution of the data analysis procedure. With this separation, a user could select among several GUIs, depending on needs or preferences.

After a user arranges an analysis procedure in the GUI, the Java applet transmits instructions to the web server using XML-RPC protocols. The software on the server arranges an appropriate Python script and executes it. Execution may occur locally or on other computing resources, today accessed by XML-RPC protocols. The response to the user is through his or her Internet browser.

Subsets of the full ARCS data analysis system will be released to run locally on a user’s platform, and platform support will be provided for single processor computing on Windows, Linux, and UNIX operating systems. We expect that these single processor software packages would include automatic and transparent Internet access to the central ARCS compute server so that the user could specify what parts
of the analysis would be performed locally or remotely.\(^2\) With such access to the central service, capabilities for large simulations on the ARCS system, for example, would be available to the user through a familiar software package. Allowing the user to select between running the ARCS software locally or remotely will permit users to select between the prompt service of a local computer for smaller-scale analyses, or a more powerful but occasionally less-responsive central computing service.

5. A User Service

5.1. Advantages of a User-Directed, Distributed Architecture

- The architecture offers access to the best combination of hardware and software.
- The architecture allows the user to select which processes are executed locally, and which ones are distributed automatically to the central hardware. The user can analyze data with any platform, although it may be advantageous to have a local capability for some frequently-used computations.
- When experimental data and analysis codes reside on central servers, interaction with the data across the Internet requires little data bandwidth.
- A clean separation of the user interface from the analysis code allows interchanging the user interface without affecting the core service.
- Centralization of the main codes simplifies their maintenance.
- Computing resources can be changed without affecting the user.
- One web portal could serve all chopper spectrometers at the SNS.\(^3\)

5.2. Issues with Users

A limited version of the ARCS web portal was demonstrated publicly on Sept. 30 and again on Dec. 13, 2002. Follow-up discussions with users about this delocalized architecture were interesting. Several were delighted that they would not have to maintain substantial computing facilities themselves. Users were generally agreeable to not running codes locally, provided that all source code is available and documented, and customization is possible on all levels. Not all users will be inclined to develop custom procedures, however, and will rely primarily on standard analysis scripts. In the later stages of the ARCS project we should know better the programming support required by users in the operations phase once the project is itself complete.

\(^2\) These local codes could include the GUI, although this remains to be decided.

\(^3\) We will work towards this goal because it is in the best interest of the ARCS IDT. We cannot promise full support for these other instruments because this requires close cooperation and some effort on the part of the other spectrometer projects.
Another user concern was that web access or the data service might be unreliable, or inconvenient for simple operations. We are well aware that high reliability and convenience are central issues for any national user facility, and often the basis by which users judge its success. We are also aware that some quick manipulations are best performed locally, and fortunately there is no serious problem in allowing the components of the ARCS software architecture to be run locally on a user’s computer. Security and graphics are also challenges for the development effort, and will remain so during operations.

To ensure that users are satisfied with the product, the ARCS software project is starting to gain experience with users by providing data analysis services for the Pharos spectrometer at the Lujan Center. There is simply no substitute for the experience of doing real science on real data.

6. Coherence

Documentation is a challenge for any large software project. The ARCS project has elevated this task from a dry mopping-up operation at the end of the project to an intellectually-stimulating challenge now. The scope of the software project is being defined in detail with the book, *Experimental Inelastic Neutron Scattering*, under development by the ARCS team. This document includes textbook content to explain the theory of inelastic neutron scattering and excitations in condensed matter. Such background is required reference material for the software algorithms and modules. The book will explain the practice of inelastic scattering with a chopper spectrometer, and software algorithms for broad families of inelastic scattering experiments. It will serve as the reference manual for software modules. This broad scope of the book ensures that the software modules are constructed to follow the underlying theory, and do so with consistent notation. Several authors are now actively writing different sections. A living draft is available at the ARCS wiki: http://arcs.caltech.edu:8000/arcs/1

The plan is to publish this book electronically through the Caltech Open Digital Archives. Books published this way are open to broad public access, and will be maintained and archived by the Caltech library. The Caltech library will be responsible for data format migration (e.g., from HTML or pdf into the future formats). The electronic book will be structured to have access to the source code of the data analysis algorithms, and will be the on-line documentation for users of the ARCS instrument. Perhaps the book will be available in printed form too.

7. Schedule

The management of any project involves the management of complexity. This is especially true of software project management. Although we have taken steps to define the architecture and enforce coherence, the numerous interconnections between different pieces of the software will be fully clear only after the system is built for the first time. Invariably there are inconsistencies in performance of the first build, and possibly even bugs. A new cycle of identifying requirements, designing,
building, and deploying will follow, and likely another cycle before the first release of the software. The capabilities of the software system will improve with each cycle. The milestones, presented in Table I, are arranged around the builds.

The project schedule allows for a first build in the summer of 2003, but not with the full functionality of Fig. 1. A second build in the summer of 2004 will allow for the full capabilities of data reduction to $S(Q, E)$, and the interface to semi-phenomenological models, as explained in Sections 2.1 and 2.2. We have selected a milestone of July 2004 for a working version of the software that would be useful for our own analysis of data from the chopper spectrometer Pharos at Los Alamos, and LRMECS or HRMECS at IPNS. Users of this software will be graduate students and postdoctoral fellows at Caltech, although we may consider providing access to very friendly users at other institutions.

The beta release of the software will occur in March of 2005. It is expected that some capability for Monte Carlo simulation will be provided in this release. Specifically, we propose to provide an instrument simulation capability comparable to McStas, but scripted in Python. A scattering kernel will be provided, as will at least one package for molecular dynamics simulation. We need to release a basic capability for full experiment simulation to the user community so we can learn how how this capability will be used by both experimentalists and theorists. We also need suggestions for the different types of simulations that would be useful. Any problems in the early test version of the software with the basic data reduction components for Pharos and LRMECS and/or HRMECS should have been solved by the beta release, and the basic modeling capability of lattice and spin dynamics should be appropriate for friendly users.

The experience of friendly users with the beta release will be important for helping us perfect the GUI(s) and documentation, which will be a focus of 2005. Today as we contemplate the development of the GUI, we are considering seriously the ISAW package. ISAW is written in Java, and appears to be a solid, modular design. We have compiled a number of ISAW modules and used them independently, for example. It seems possible to use at least some of the functionality of ISAW in a Java applet for a user interface running under a web browser. The present form of the ISAW scripting language is not well implemented, however, so we are not sure how to extend it. We expect decide on the role for ISAW before mid-2004.

The first release of the software will be in March, 2006, one month before the operational readiness review for the ARCS instrument. The remainder of the ARCS

Table I. Milestones

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Baseline Design</td>
<td>January, 2003</td>
</tr>
<tr>
<td>Software First Build</td>
<td>July, 2004</td>
</tr>
<tr>
<td>Software Beta Release</td>
<td>March, 2005</td>
</tr>
<tr>
<td>Software Release 1.0</td>
<td>February, 2006</td>
</tr>
<tr>
<td>End ARCS Project</td>
<td>September, 2006</td>
</tr>
</tbody>
</table>
Figure 5. Schedule for the ARCS software project

project will likely be focused on performance improvements for the capabilities listed in Fig. 1.

8. Personnel and Resources

The level of effort budgeted for the ARCS software effort is 0.58 M$/year. This support is primarily for salaries of postdoctoral fellows, graduate students, and software professionals who will be designing the code, developing it, and testing it. A relatively small amount of money is used to purchase standalone computers, computer clusters, and visualization systems. It should be pointed out that the ARCS software budget and the ARCS hardware budget are not funded independently. The total project budget is fixed, and to date we have been underspending the ARCS software effort to help with the overall project contingency.

The postdoctoral fellows are responsible for the components of the scientific mission, specifically the three paths shown in Fig. 1 and discussed in Sect. 2 — data reduction, modeling, and direct experiment simulation. Graduate students working for their Ph.D. theses need components from these three paths, and contribute to the effort in both code development and testing. Support for their Ph.D. thesis work is typically not provided by the ARCS project, however. A couple of undergraduate
students have made significant contributions to the project. The project also requires the services of software engineers to oversee the architecture, set standards, and integrate visualization tools into the ARCS software.
Comparison to DANSE Project

The ARCS architecture described in Sect. 3 could be used for instruments other than the chopper spectrometers at the SNS. This would require an enlarged scope for both science and software engineering, but in principle the same Internet web portal could provide the data analysis needs for all instruments at the SNS. Other neutron facilities could also be served by such a national facility for distributed data analysis for neutron scattering experiments (DANSE). The authors have prepared a white paper outlining this DANSE project, and this Appendix describes the differences in the ARCS and DANSE software projects.

Although the proposed DANSE system is based on the ARCS architecture, the DANSE project differs considerably in scope. For DANSE to provide a complete solution to the data analysis and visualization needs of the entire neutron scattering community, it must include the full scope of neutron scattering science, not just the inelastic scattering that is the concern of the ARCS project. Software development activities in elastic scattering, engineering diffractometry, large-scale structures, neutronics, and theory would be centered at institutions other than Caltech.

If the DANSE proposal were funded as envisioned, the software engineering effort would be expanded considerably at Caltech. It would be necessary to develop carefully the standards for the data stream paradigm and for the components. Software tools would be developed to make it easier to encapsulate legacy computer codes as cores of the components. A standard object model, or a set of these, would be developed so that programs in different languages could be used for component cores. In contrast, the ARCS effort will probably be restricted to C++ and occasional FORTRAN cores, and Python bindings will be written by hand. The DANSE project would include a larger effort to develop GUIs, with the likely requirement that existing GUIs of different user facilities be supported in the DANSE architecture. A rule of thumb from F. Brooks, *The Mythical Man Month*, is that if it takes $X$ hours to develop software for use by one person, it takes $3X$ hours to make this same code convenient for others. Supporting multiple platforms is another multiplier of 3. It is not surprising that the DANSE project would require a much larger software development effort at Caltech, although less than a factor of 9. The DANSE project also requires much more substantial hardware, since numerous users would demand prompt service with high reliability.

A national facility such as DANSE would facilitate national coordination of neutron instruments and facilities. Simulations that compare the performance of different instruments for a particular type of experiment could be performed readily. For individual neutron facilities, simulations already provide important guidance on instrument upgrades or operational procedures. A rational comparison of instruments at different facilities is not yet possible, because different instruments have different strengths and weaknesses, which are emphasized differently in different experiments. A broader simulation capability, built in part on user experience with specific experiments, would be a better tool for planning the usage and upgrades to neutron instruments. Simulations of instrument performance will also be useful for teaching, and for reaching out to the theory community.