Acta Crystallographica Section D
Biological Crystallography
ISSN 0907-4449

PHENIX: building new software for automated crystallographic structure determination

Paul D. Adams,[a]* Ralf W. Grosse-Kunstleve,[a] Li-Wei Hung,[b] Thomas R. Ioerger,[c] Airlie J. McCoy,[d] Nigel W. Moriarty,[a] Randy J. Read,[d] James C. Sacchettini,[e] Nicholas K. Sauter[a] and Thomas C. Terwilliger[f]

[a]Lawrence Berkeley National Laboratory, One Cyclotron Road, Mailstop 4-230, Berkeley, CA 94720, USA, [b]Biophysics Group, Mail Stop D454, Los Alamos National Laboratory, Los Alamos, NM 87545, USA, [c]Department of Computer Science, Texas A&M University, 301 H. R. Bright Building, 3112 TAMU, College Station, TX 77843, USA, [d]Department of Haematology, University of Cambridge, Cambridge Institute for Medical Research, Wellcome Trust/MRC Building, Hills Road, Cambridge CB2 2XY, England, [e]Department of Biochemistry and Biophysics, Texas A&M University, 103 Biochemistry/Biophysics Building, 2128 TAMU, College Station, TX 77843, USA, and [f]Los Alamos National Laboratory, Mailstop M888, Los Alamos, NM 87545, USA

Correspondence e-mail: pdadams@lbl.gov

Received 6 May 2002
Accepted 12 September 2002

Structural genomics seeks to expand rapidly the number of protein structures in order to extract the maximum amount of information from genomic sequence databases. The advent of several large-scale projects worldwide leads to many new challenges in the field of crystallographic macromolecular structure determination. A novel software package called *PHENIX* (Python-based Hierarchical ENvironment for Integrated Xtallography) is therefore being developed. This new software will provide the necessary algorithms to proceed from reduced intensity data to a refined molecular model and to facilitate structure solution for both the novice and expert crystallographer.

1. Introduction

1.1. Structural genomics

The database of known sequences is growing at an exponential rate and has already resulted in complete genomic sequences for several organisms. This huge database of information is expected to provide great insight into individual organisms and life in general. The sequence information can be interpreted in the light of known structures to deduce the functional significance of coding regions of the genomes. However, only a fraction of the expected structural information is currently known (Orengo *et al.*, 1999). Although the rate of protein structure determination is increasing, it has been overtaken by the concerted efforts of the sequencing projects. Trying to match the rate of sequence determination is not a feasible task. Instead, it is proposed to direct structure-determination efforts towards specific targets that would produce information about unknown folds of structural representatives of sequence families. By analogy with sequencing projects, this has become known as structural genomics (Burley *et al.*, 1999). An expanded structural database, in combination with knowledge of function, would permit extraction of much more information from the sequence databases than is currently possible. It is hoped that in the long term it will become possible to derive at least an approximate model for any sequence using the information in the expanded structural database (Sali, 1998).

1.2. The need for automation

For structural genomics to be possible, structures will need to be solved significantly faster than is currently routinely achievable. This high-throughput structure determination will require automation to reduce the obstacles related to human intervention. Several projects are under way worldwide to automate the process of sample preparation, crystallization and data acquisition. Currently, one of the main bottlenecks to

completion of a macromolecular crystal structure is computational. Manual interpretation of complex numerical data and the repeated use of interactive three-dimensional graphics are often required. This is time-consuming, often of the order of weeks or months, and also has a significant subjective component (Mowbray *et al.*, 1999) that can lead to delays in reaching the final structure. Thus, the automation of structure solution is essential as it has the potential to produce minimally biased models more efficiently. Automation will rely on the development of algorithms that minimize or eliminate subjective user input, the development of algorithms that automate procedures that were traditionally performed by hand and finally the development of software packages that allow a tight integration between these algorithms. Truly automated structure solution will require the software to make decisions about how best to proceed in the light of the available data.

1.3. Existing crystallographic software

Improving the efficiency of structure solution has been a goal for many software developers. The creation of faster and more accurate algorithms serves to increase productivity. However, as algorithms become more complex they often become less portable, with expert knowledge being required to implement or modify them. It is our experience that the technical problems involved in combining algorithms to generate a more powerful system are often prohibitive. Similarly, there can be significant technical barriers that make it difficult to apply existing algorithms to new problems. Therefore, the design limitations of existing crystallographic software packages become a serious obstacle to their advancement.

Most existing programs have legacy code written in an era of procedure-oriented software design and monolithic applications. Owing to the lack of consistent software design, most crystallography projects require the application of different and incompatible software packages for various steps in the calculation (data reduction, phasing, map calculation, model building, model refinement). There have been advances in the standardization of file formats within the widely used *CCP*4 package (Collaborative Computational Project, Number 4, 1994). The use of programs has been simplified with the availability of graphical user interfaces. For example, programs within *CCP*4 can be accessed through a graphical user interface (ccp4i) and task files in the *Crystallography & NMR System* (*CNS*) can be accessed through a web-browser interface (Brünger *et al.*, 1998). However, even for the experienced structural biologist the need to use several diverse software packages presents a formidable barrier. Difficult choices must be made between alternative methods, relying on documentation that often inadequately describes the underlying algorithms. This is exacerbated by the need for repeated reformatting of data files in order to accommodate the different packages. Combined with the need for frequent manual intervention, the entire process can take months even if it requires only modest computer time. The novice crystallographer is often overwhelmed by the apparent technical complexities and has to rely heavily on personal expert advice to accumulate the required knowledge. Lacking the benefits of modern design principles, it is extremely difficult to correct these deficiencies.

1.4. Scripting languages, object-oriented design and rapid prototyping

The use of scripting languages is widespread in many fields. Indeed, the success of the World Wide Web lies partly with the availability of languages such as PERL (Schwartz & Christiansen, 1997). The use of a powerful scripting language can potentially reduce the problems associated with technical complexity by expressing scientific algorithms in a plain easy-to-understand syntax. *CNS* provides such a system for X-ray crystallography (Brünger *et al.*, 1998). It includes a high-level interpreted scripting language, which makes it very easy to test and implement new ideas. Basic scripting languages are also used in other crystallographic systems, frequently in the form of Unix shell scripts to control the application of programs. However, a tighter integration between the scripting language and the underlying scientific algorithms is required for true automation.

Many of the scripting languages available follow a procedural programming style. This should be contrasted with the object-oriented style used by many modern programming languages. In the object-oriented programming model, *abstraction*, *encapsulation* and *modularity* provide mechanisms to organize a problem in a hierarchy so that the higher level objects need not know how the lower level objects operate. The resulting code is very flexible, easy to maintain and lends itself to shared collaborative development by multiple groups. Since tasks are organized into discrete modules with extendable properties, it is easy to reuse existing code to handle new situations.

Although object-oriented design is clearly superior for organizing large computational projects and scripting is beneficial for rapid development of high-level ideas, the two

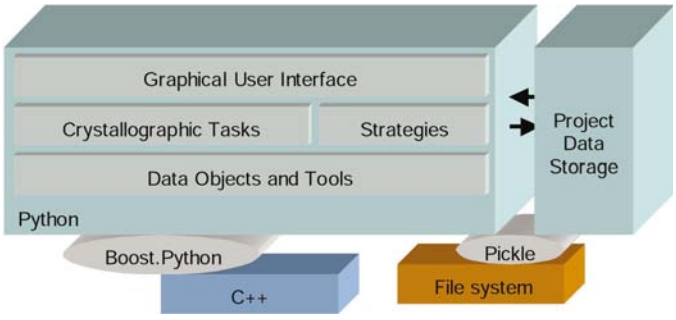


Figure 1
Architecture of the *PHENIX* system. The Python scripting language provides the backbone of the system. The Boost.Python library is used to integrate core C++ code into Python. On top of this, the data objects, crystallographic tasks, strategies and finally a graphical user interface are constructed. The Project Data Storage makes use of the pickle mechanism in Python to store data on the file system.

frameworks have historically been mutually exclusive. In the early 1990s, however, the Python programming environment was designed to merge the virtues of both approaches. Python is a portable interpreted object-oriented programming language (Lutz & Ascher, 1999). It incorporates modules, dynamic variable typing and classes. The dynamic typing distinguishes Python from statically typed object-oriented languages such as C++ and Java. A recent study came to the conclusion that program development with Python is approximately twice as fast as development with C++ or Java (Prechelt, 2000). Therefore, Python is often referred to as a language for *rapid prototyping*. In addition, there are now a vast selection of modular and object-oriented open-source packages implementing graphical user interfaces, databases, network communication and numerical algorithms available for Python. Building a new software system with Python as a foundation provides the developer with access to this large pool of pre-existing resources.

2. Software design and implementation

2.1. Architecture

One of the primary goals of the *PHENIX* project is to create a tight integration between reusable software components written both in a compiled language and a flexible scripting language. Our prior experience implementing *CNS* has shown that this promotes highly efficient software development. High-level algorithms such as complex refinement protocols or phasing procedures can be developed most rapidly in a scripting language. By contrast, numerically intensive core algorithms such as the computation of structure factors or discrete Fourier transforms must be implemented in a compiled language for performance reasons.

An evaluation of available technologies led us to choose Python (http://python.org/) as the scripting language and C++ as the compiled language (see Grosse-Kunstleve *et al.*, 2002 for further details about these choices). Importantly, the Boost.Python Library (http://www.boost.org/) is available for conveniently integrating C++ and Python. It is used to directly connect C++ classes and functions to Python without obscuring the C++ interface. We have worked with one of the primary authors of the Boost.Python Library (David Abrahams) to extend its functionality to better suit the needs of the *PHENIX* project. The overall architecture of the *PHENIX* system is shown in Fig. 1 and our progress to date is presented below. Everything described here has been fully tested on three different Unix platforms (Redhat Linux, Compaq Tru64 and SGI Irix version 6.5) and Windows 2000.

2.2. Data objects and tools

In order to build a complex integrated system such as *PHENIX*, certain basic data objects must be available. We have implemented some of the important objects required for crystallographic computations.

(i) Structure-factor objects, which hold reciprocal-space data. Data containers have been implemented in C++ and are made available in Python using the Boost.Python library. This design permits the reuse of the 'objects' by future developers within either a compiled C++ program or an interpreted Python script (Fig. 2). For the typical end user, complex calculations can be performed on structure-factor data from the Python scripting language (see Fig. 3).

(ii) Map objects, which hold real-space data such as electron-density maps. Data containers have been implemented in C++ and will be made available in Python in the near future.

(iii) Molecular objects, which hold the coordinates and topology of a structure. Data containers have initially been implemented in Python for speed of testing and development. Coordinate files from the Protein Data Bank can be read into *PHENIX* and the appropriate connectivity between atoms determined.

Implementation of these objects makes use of the *Computational Crystallography Toolbox* (*cctbx*; Grosse-Kunstleve *et al.*, 2002). This is a library of *reusable* core crystallographic software components for macromolecular structure determination that have been designed for integration into large modular layered software systems. The *cctbx* source code is freely available under an Open Source license for both non-profit and commercial use at http://cctbx.sourceforge.net/.

2.3. Algorithms

A broad range of algorithms for structure solution will be implemented. Experimental phasing using both MAD/SAD and MIR/SIR methods will be optimized by the use of well established automated Patterson methods for heavy-atom location (Terwilliger & Berendzen, 1999; Grosse-Kunstleve & Brunger, 1999) combined with new maximum-likelihood scoring functions. Once sites have been located, efficient new algorithms for phasing that take account of correlations in the errors between derivatives and wavelengths will be used (Terwilliger & Berendzen, 1996, 1997; Read, 1991). Alternatively, the use of new maximum-likelihood targets for molecular replacement, which have been tested in the program *BEAST* (Read, 2001), will increase the success rate of this method using search models of lower structural similarity. The phases obtained from experimental phasing or molecular replacement will be optimized by the application of maximum-likelihood density-modification algorithms, currently implemented in the *RESOLVE* program, to produce minimally biased electron-density maps (Terwilliger, 2001). Electron-density maps will be automatically interpreted using template matching (Terwilliger, 2001) as implemented in *RESOLVE* and pattern-recognition methods as implemented in *TEXTAL* (Holton *et al.*, 2000). Automated map interpretation will be iterated with maximum-likelihood refinement targets (Pannu & Read, 1996; Pannu *et al.*, 1998) and simulated-annealing optimization algorithms (Brünger *et al.*, 1987; Adams *et al.*, 1997, 1999). We expect that this combination will permit automated structure completion even when diffraction data are only available up to a modest resolution limit (approximately 3 Å).

```
#include <xarray/xarray.h>
using xarray;

int main() {
double SigCut = 1.0;
double DifCut = 0.5;

//CALCULATE AMPLITUDES
DoubleSet F    = Iobs.F();
DoubleSet sigF = Iobs.sigF();

//SELECTION ON SIGMA (F)
IntegerSet S1 = F > SigCut*sigF;

//CALCULATE ANOMALOUS DIFFERENCES
DoubleSet anom_diff = F.plus() - F.minus();
DoubleSet sigdiff = sqrt (pow2(sigF.plus()) +
                    pow2(sigF.minus()) );

//SELECTION ON SIGMA (delta F)
IntegerSet S3 = abs(anom_diff) > (DifCut * sigdiff);
SelectedDoubleSet dfsquare = (anom_diff *
                              anom_diff).select(S1 & S3);

//CALCULATE E VALUES
Binner ebin = A.Binner(8, 3.0);
DoubleSet amp = sqrt(dfsquare);
DoubleSet E = amp / ebin.average(amp);
DoubleSet Esquare = E * E;

//REMOVE ORIGIN
DoubleSet Esq_rem = Esquare - ebin.saverage(Esquare);

//CALCULATE AND OUTPUT PATTERSON MAP
MapForm proform = MapForm(A,0,3.0,0.25);
Xarray pro = Xarray(proform);
pro.loadArray(Esq_rem);
}
```

(*a*)

```
from xtbx_boost.xarray import *

SigCut = 1.0
DifCut = 0.5

#CALCULATE AMPLITUDES
F    = Iobs.F()
sigF = Iobs.sigF()

#SELECTION ON SIGMA (F)
S1 = F > SigCut*sigF

#CALCULATE ANOMALOUS DIFFERENCES
anom_diff = F.plus() - F.minus()
sigdiff = sqrt (pow2(sigF.plus()) +
          pow2(sigF.minus()) )

#SELECTION ON SIGMA (delta F)
S3 = abs(anom_diff) > (DifCut * sigdiff)
dfsquare = (anom_diff * anom_diff).select(S3)

#CALCULATE E VALUES
ebin = A.Binner(8, 3.0)
amp = sqrt(dfsquare)
E = amp / ebin.average(amp)
Esquare = E * E

#REMOVE ORIGIN
Esq_rem = Esquare - ebin.saverage(Esquare)

#CALCULATE AND OUTPUT PATTERSON MAP
proform = MapForm(A,0,3.0,0.25)
pro = Xarray(proform)
pro.loadArray(Esq_rem)
```

(*b*)

Figure 2
C++ (*a*) and Python (*b*) interfaces for *PHENIX* code. The two code samples use as similar a syntax as is permissible by the constraints of the respective languages, yet the first is compiled and the second interpreted. There is a performance penalty associated with running a scripting language. However, because the majority of the time is spent in the compiled C++ routines, the execution times for the two interfaces are roughly equivalent on all platforms tested.

2.4. Strategies and the graphical user interface

Many crystallographic software packages either provide the user with a collection of tools for analysis of the data or with a monolithic 'black box' application that performs all the relevant tasks in an automated fashion. Neither situation is optimal for the user. In the first case, the user is required to use the tools in the correct sequence and also make decisions based on the results at each stage. Although this has proved a successful mode for the expert user, it makes software difficult to use for the non-expert user and often leads to time-consuming mistakes. The use of self-contained automated systems that lack user control can be productive for the novice user. However, when problems are encountered (*i.e.* the automation fails), it is often very difficult to identify the problem or implement a solution.

We have introduced the concept of *strategies* into *PHENIX* to avoid these problems. Strategies provide a way to construct complex networks of tasks to perform a higher level function. For example, the steps required to go from initial data to a first electron-density map in a SAD experiment can be broken down into well defined tasks which can be reused in other procedures. Instead of requiring the user to run these tasks in the correct order, they are connected together by the software developer and can thus be run in an automated way. However, because the connection between tasks is dynamic, they can be reconfigured or modified and new tasks introduced as necessary if problems occur. This provides the flexibility of user input and control, while still permitting complete automation when decision-making algorithms are incorporated into the environment.

The tasks and their connection into strategies rely on the use of task files written using the Python scripting language. This implementation was chosen so as to not restrict the use of *PHENIX* to a graphical user interface. These task files and the Python objects describing a strategy can be used from other programs or in a non-graphical environment. However, they can also be readily displayed in a graphical environment. We have used the wxPython graphical interface development tool to implement a graphical user interface (GUI) for *PHENIX*. This GUI permits strategies to be visualized and manipulated (Fig. 4). These manipulations include loading a strategy distributed with *PHENIX*, customizing and saving it for future recall. Customization of the task input parameters is achieved *via* the interface displayed on the right of the figure. Insertion of new tasks is performed by choosing from the tree menu on

the left of the main window. The GUI also provides a means for the user to view results of calculations (Fig. 5). *PyMOL* (DeLano, 2002), a molecular-graphics system written in C and Python, allows easy viewing of structures and maps *via* close integration with *PHENIX*. Graphical data is transferred from *PHENIX* to *PyMOL* through a socket connection. Simple results can be represented by native *PHENIX* display windows as shown in the left of the figure.

2.5. The Project Data Storage (PDS)

One of the major problems facing the crystallographer is the organization, tracking and archiving of data. Programs produce an array of different output files, often in different formats. Different trials in the structure-determination process can generate output that is eventually discarded because this path was later abandoned. These problems are significantly compounded by the move to high-throughput crystallography,

```
from xtbx_boost import xarray
from cctbx_boost import sgtbx, uctbx

from math import sin,cos,pi
inputfile = "/net/cci/paul/model_fcalc_R3.list"
outputfile = "R3.xplor"
uc = uctbx.UnitCell((104.4,104.4,124.25,90,90,120))

F = open(inputfile,"r")
for i in xrange(7): F.readline()

A = xarray.Ispace("R3",uc)
Fcalc = A.newMemberComplexSet("Fcalc",0)

for j in F.readlines():
       k = j.split()
       amplitude = float(k[3])
       phase = float(k[4])*pi/180.
       Miller = (int(k[0]),int(k[1]),int(k[2]))
       FA = amplitude*cos(phase)
       FB = amplitude*sin(phase)
       Fcalc.hklLoad(Miller,complex(FA,FB))

proform = xarray.MapForm(A,0,2.0,0.33)
pro = xarray.Xarray(proform)
pro.loadArray(Fcalc)
pro.real2CNSfile(outputfile)
```

Figure 3
Left, a *PHENIX* script file that reads amplitudes and phases from a text file, then performs a Fourier transform and generates an electron-density map. Right, the electron-density map visualized using *PyMOL* (DeLano, 2002).

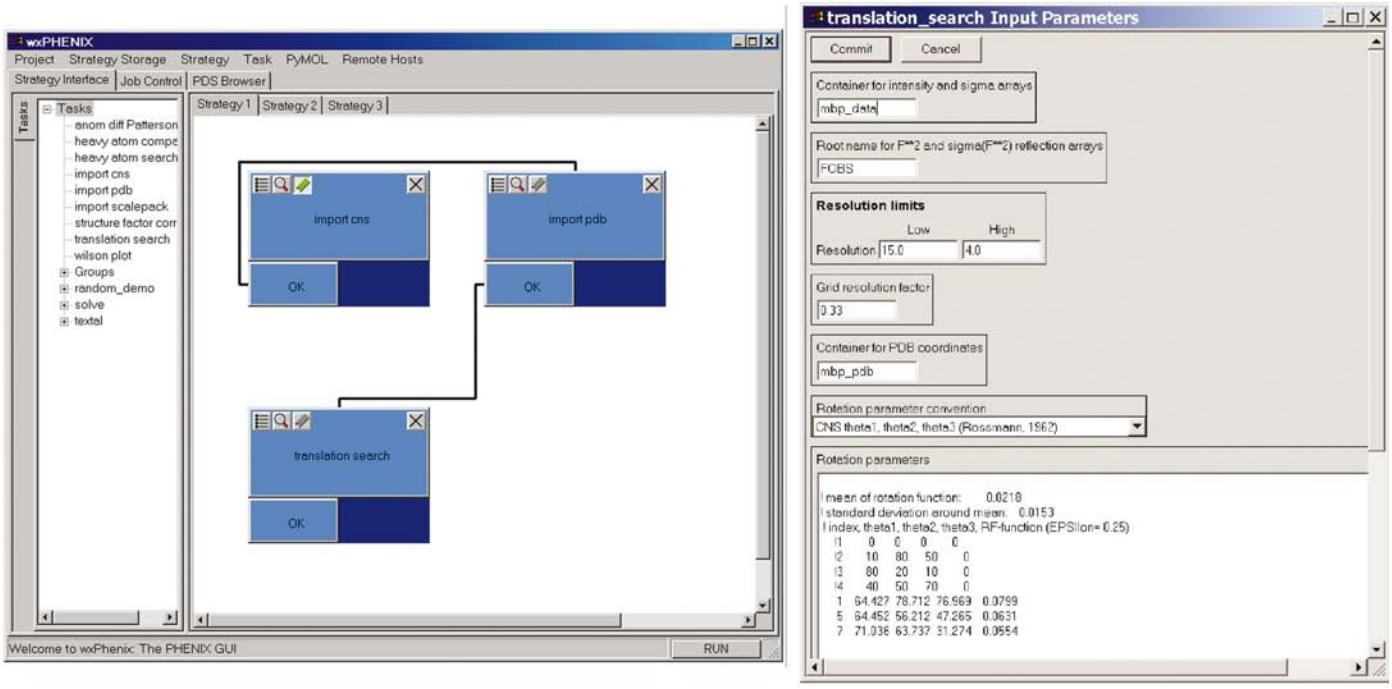
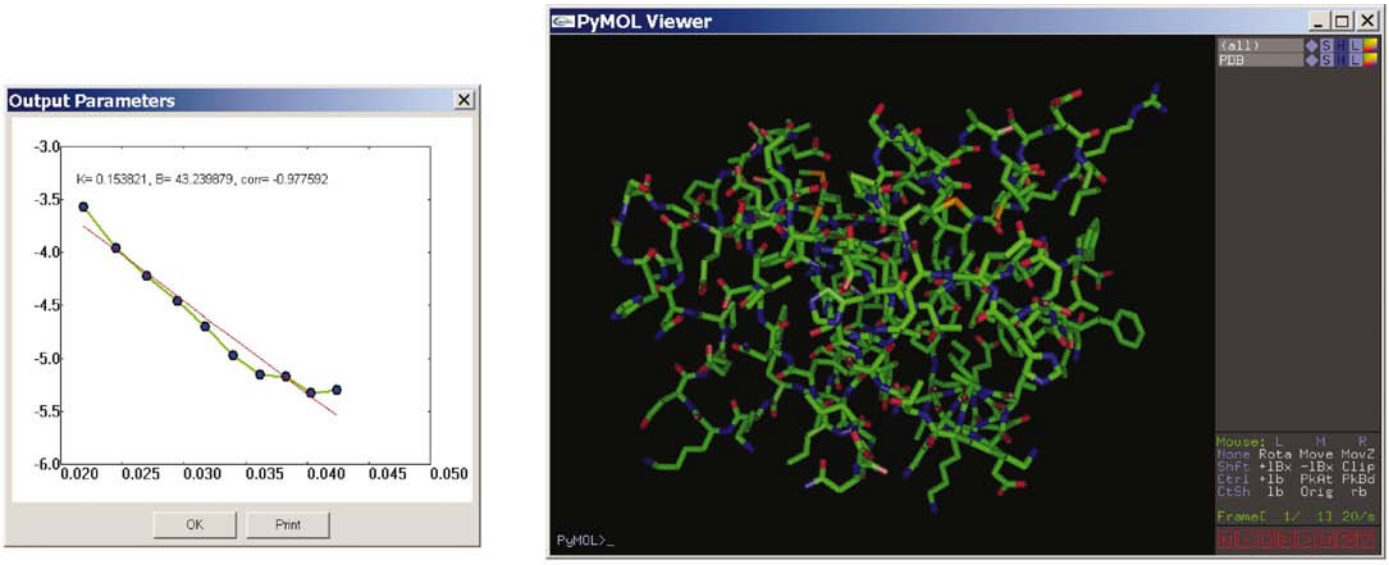


Figure 4
PHENIX strategy interface showing a simple strategy and a task-parameter input.



(*a*) (*b*)

Figure 5
PHENIX strategy interface showing graphical output. (*a*) A simple graph showing the Wilson plot, fitted line and parameters derived from the plot. (*b*) A graphical representation of a molecule imported in *PHENIX* for a molecular-replacement translation search. Visualized using the *PyMOL* program, which has been integrated into the *PHENIX* environment.

which leaves no time for user control of data management. Therefore, in *PHENIX* we have introduced the concept of the Project Data Storage (PDS). This is a data-management system that oversees the information generated for each structure determination (or 'Project') and contains a complete history of each structure solution, along with all of the generated structural information. The PDS is essential for the purposes of preserving data integrity, communication between the components of *PHENIX*, data mining and ultimately structure deposition to the PDB (Berman *et al.*, 2000). The ability for reliable information gathering *via* the PDS will be crucial in determining the criteria and figures of merit that are required to develop automated decision-making algorithms.

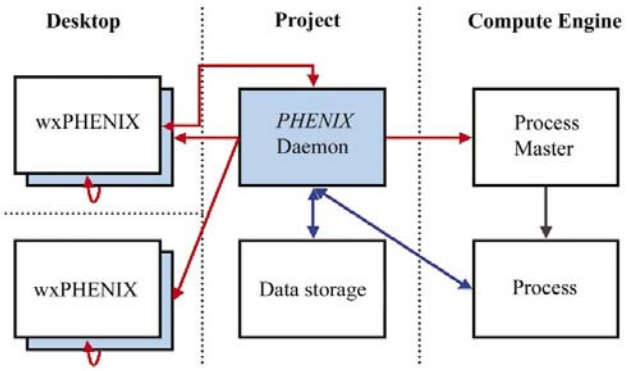


Figure 6
PHENIX distributed computing mechanism. The *PHENIX* daemon coordinates the flow of information between the various components and also controls the creation and termination of processes. Multiple instances of the GUI can interact with the daemon to view the progress of a job. 'Desktop', 'Project' and 'Compute Engine' components can be run on the same machine or optionally on different computing hosts and communicate with each other over the internet using sockets.

2.6. Distributed computing

In order to make full use of the computing resources typically available to researchers, we have developed and implemented a distributed computing model for *PHENIX*. This permits the remote execution of computationally intensive tasks (for example, a job can be set up on a user's desktop PC, but sent to a high-performance computing platform for execution). Once a job has been started it can also be monitored and controlled remotely by multiple instances of the graphical user interface running on different machines. This distributed computing model relies on a *PHENIX* daemon that coordinates the information flow for each user project (see Fig. 6).

3. Conclusions

The development of *PHENIX* is a collaborative project whose primary goal is the creation of a comprehensive integrated system for automated crystallographic structure determination. However, we also hope that *PHENIX* can be more than this alone. The *PHENIX* infrastructure is designed to be open and easily shared with other researchers. Source code will be distributed to academic groups and the use of the Python scripting language will facilitate interfacing with the system and its use in different contexts. Other developers will be able to 'plug in' their algorithms to the *PHENIX* environment, thus providing easy access to a large number of crystallographic and computational tools. The high-level graphical programming environment in *PHENIX* is designed to let researchers easily link crystallographic tasks together, thus creating

complex algorithms without having to resort to low-level programming. The graphical interface and the underlying Python scripting language also provide a framework suitable for implementing decision-making algorithms that will be critical for robust and reliable automation.

Many of the features of the *PHENIX* system are not specific to macromolecular crystallography. The graphical strategy-manipulation interface provides a generic tool for visual programming that is based on the Python scripting language. This interface could be used to link together more traditional command-line software packages while still presenting the user with an integrated system. Alternatively, the graphical interface and other underlying tools could be used in other areas of structural biology such as single-particle cryo-electron microscopy, electron diffraction and NMR. The *PHENIX* system thus provides a framework for the integration of different experimental approaches to probing macromolecular structure.

This work was funded by NIH/NIGMS under grant number 1P01GM063210, with initial funding to PDA from the Department of Energy under contract No. DE-AC03-76SF00098.

References

Adams, P. D., Pannu, N. S., Read, R. J. & Brünger, A. T. (1997). *Proc. Natl Acad. Sci. USA*, **94**, 5018–5023.
Adams, P. D., Pannu N. S., Read, R. J. & Brunger, A. T. (1999). *Acta Cryst.* D**55**, 181–190.
Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N. & Bourne, P. E. (2000). *Nucleic Acids Res.* **28**, 235–242.
Brünger, A. T., Kuriyan, J. & Karplus, M. (1987). *Science*, **235**, 458–460.
Brünger, A. T., Adams, P. D., Clore, G. M., Gros, P., Grosse-Kunstleve, R. W., Jiang, J.-S., Kuszewski, J., Nilges, N., Pannu, N. S., Read, R. J., Rice, L. M., Simonson, T. & Warren, G. L. (1998). *Acta Cryst.* D**54**, 905–921.
Burley, S. K., Almo, S. C., Bonanno, J. B., Capel, M., Chance, M. R., Gaasterland, T., Lin, D., Sali, A., Studier, F. W. & Swaminathan, S. (1999). *Nature Genet.* **23**, 151–157.
Collaborative Computational Project, Number 4 (1994). *Acta Cryst.* D**50**, 760–763.
DeLano, W. L. (2002). *The PyMOL Molecular Graphics System*, http://www.pymol.org
Grosse-Kunstleve, R. W. & Brunger, A. T. (1999). *Acta Cryst.* D**55**, 1568–1577.
Grosse-Kunstleve, R. W., Sauter, N. K., Moriarty, N. W. & Adams, P. D. (2002). *J. Appl. Cryst.* **35**, 126–136.
Holton, T., Ioerger, T. R., Christopher, J. A. & Sacchettini, J. C. (2000). *Acta Cryst.* D**56**, 722–734.
Lutz, M. & Ascher, D. (1999). *Learning Python.* California, USA: O'Reilly & Associates.
Mowbray, S. L., Helgstrand, C., Sigrell, J. A., Cameron, A. D. & Jones, T. A. (1999). *Acta Cryst.* D**55**, 1309–1319.
Orengo, C. A., Pearl, F. M., Bray, J. E., Todd, A. E., Martin, A. C., Lo Conte, L. & Thornton, J. M. (1999). *Nucleic Acids Res.* **27**, 275–279.
Pannu, N. S., Murshudov, G. N., Dodson, E. J. & Read, R. J. (1998). *Acta Cryst.* D**54**, 1285–1294.
Pannu, N. S. & Read, R. J. (1996). *Acta Cryst.* A**52**, 659–668.
Prechelt, L. (2000). *An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl for a search/string-processing program.* Technical Report 2000-5, University of Karlsruhe, Germany.
Read, R. J. (1991). *Proceedings of the CCP4 Study Weekend. Isomorphous Replacement and Anomalous Scattering*, edited by W. Wolf, P. R. Evans & A. G. W. Leslie, pp. 69–79. Warrington: Daresbury Laboratory.
Read, R. J. (2001). *Acta Cryst.* D**57**, 1373–1382.
Sali, A. (1998). *Nature Struct. Biol.* **5**, 1029–1032.
Schwartz, R. & Christiansen, T. (1997). *Learning Perl.* California, USA: O'Reilly & Associates.
Terwilliger, T. C. (2001). *Acta Cryst.* D**57**, 1755–1762.
Terwilliger, T. C. & Berendzen, J. (1996). *Acta Cryst.* D**52**, 749–757.
Terwilliger, T. C. & Berendzen, J. (1997). *Acta Cryst.* D**53**, 571–579.
Terwilliger, T. C. & Berendzen, J. (1999). *Acta Cryst.* D**55**, 849–861.