LHSG-CT-2003-503420

BioXHIT

A European integrated project to provide a highly effective technology platform for Structural Genomics.

Life Sciences, Genomics and Biotechnology for Health

WP5: De5.1.10 Report on the evaluation/identification of the toolbox functions

Due date of deliverable: **31.12.2005**
Actual submission date: **21.12.2005**

Start date of project: **1.1.2004** **Duration: 60 months**

Organisation name of lead contractor for this deliverable: EMBL-EBI
European Molecular Biology Laboratory - European Bioinformatics Institute,
Hinxton, Cambridge, UK. **Author** Peter Briggs

Section 5: Data Bases and Networking (managed by Partner 1C)
The tasks in Section 5 are "horizontal" in nature and cut across the tasks described in Sections 1 to 4. The main focus of this Section is on the management, transfer and access of data and of software components that are the results of the developments in those earlier sections. This is to be achieved through the setting of BIOXHIT standards for software and for data sharing, which will be used in implementations both in this Section and in others. Section 5 also contains an element of remote access to data and services provided by BIOXHIT. Remote access will also be facilitated by the setting and observance of standard protocols, for example by the use of emerging GRID technologies.

WP 5.1: Standardisation in Software and Data Transfer. Coordinator Avi Naim (Partner 1C), contributing Partners 1C, 10, 15.

This workpackage will ensure the transferability both of software components and of the data that they produce and use.

De5.1.10 *Report on the evaluation/identification of the toolbox functions*

Evaluation report on the requirements for a PX software toolbox

1. Introduction

Part of Task 5.1.11 (formally task 4.7.1) of BIOXHIT, "Automation in Computation", was to investigate and report on the requirements of the relevant Partners for a "PX software toolbox". It was envisaged that such a toolbox could be created by abstracting low-level and medium-level functions found in CCP4 and other software packages, which are commonly used within automated software pipelines (along with additional useful functionality not commonly available at present in CCP4). It was also important to consider the context that such functions might be invoked from (for example, from specific programming languages or environments), and the suitability of any existing toolbox-like projects.

A joint BIOXHIT/e-HTPX/CCP4 workshop was held in February in Cambridge UK as part of the investigation into defining the toolbox requirements (the workshop also focused on defining the decision-making XML schema of deliverable D 5.1.9). The workshop gathered together representatives from the principal automation pipeline efforts within BIOXHIT in addition to a number of similar efforts outside of the project.

The full report from the workshop is included as appendix A at the end of this report.

2. Pipeline overview: toolbox context

The pipeline systems considered were overall very similar in construction: well-defined "computational units" (usually compiled programs) that implement high-level crystallographic functions are linked together within a scripting environment. The scripting environment typically performs tedious but relatively easily automated lower level actions such as preparing data for input into the higher level components, extracting data from the output, and analysing this data to make decisions.

The choice of high-level units was generally quite consistent across the pipelines surveyed, consisting in the main of programs that (for various reasons) would not be considered as suitable candidates for incorporation into a toolbox. However it would

appear that much of the work of pipelines is involved in dealing with the needs of preparing input to and analysing the output of these higher-level units, and the lower level operations that are required to achieve this are better suited to being made into a toolbox. It is this set of functions that is considered in the next section.
It should be noted that this report does not consider functions that would provide "automation infrastructure" as part of the requirements. Such a function set might be useful when building a new automation pipeline, but is more limited use in the context of existing pipeline projects such as those discussed in the workshop report.

3. Functionality requirements for a PX software toolbox

Analysis of the software pipelines considered in the workshop identified 6 broad areas of common low- and medium-level functionality that most pipelines seemed to share. These are summarised below, with examples given in italics:

- Processing and analysis of program output
  - *Analysing the log file from a program to extract useful data*
- Querying & extracting information from common data files
  - *Extracting header information from reflection data files*
- Conversion of data files between different formats to prepare input for a subsequent program or function in the pipeline
  - *Switching between reflection file formats, switching hand*
- Conversions between different conventions
  - *Converting between different asymmetric unit conventions*
- Manipulations of crystallographic data
  - *Sorting reflections, changing map limits*
- Basic crystallographic algorithms
  - *Peak picking from an electron density map, solvent content analysis*

(See appendix B.2 of the workshop report for a more detailed summary that links back to the specific pipelines. An additional area of functionality identified in the appendix – namely interactions with remote services such as BLAST – is not considered here, as it only appeared in one of the pipeline projects.)

We will now look at each of these areas in more detail.

3.1 Processing and analysis of program output

A standardised way of extracting data items from the output of high-level programs such as MLPHARE or SHELX would greatly facilitate automated pipeline efforts (as well as making them easier to maintain). This could be incorporated into a PX software toolbox in a number of ways:

- Standard "translator" functions which are able to process the logfile output of a particular program into a standardised machine-readable format (such as XML), or
- A set of writing functions that could be incorporated into and called directly from a compiled program.

In either case there is a requirement for a standard set of defined tags, such as those being developed elsewhere in BIOXHIT for data exchange. The major difference between these approaches

The corollary of writing data in this form is that of course one must also be able to read it. Thus a complementary set of functions would be required to facilitate the extraction of the required data from the machine-readable format.

3.2 Querying and extracting information from common data files

Specifically this refers to extracting information from the various file formats used for storing reflection data, density maps and coordinate files. The information in question is generally metadata (so-called "header information") for example the total number of reflections, the extent of a density map or the spacegroup associated with a set of atomic coordinates.

This functionality could be implemented using wrappers or by writing XML from programs intended purely to dump the header information (for example the MTZDUMP program in CCP4, which prints out data from an MTZ format reflection file). Alternatively a set of library functions that implemented this function could be wrapped in lightweight programs as well as being made available via interfaces to a scripting language.

3.3 Conversion of data files between different formats to prepare input for a subsequent program or function in the pipeline

A significant overhead in creating automated pipelines concerns the transformation of data between different file formats. Programs exist within CCP4 that can be used to convert between file formats but these often require a degree of user intervention that makes them less than optimal for incorporation into pipelines.

Ideally the toolbox functions for performing file format conversions should operate as automatically as possible, and require minimal additional input – for example, automatically determining the type of input file so that only the desired output format would need to be specified.

These functions could be implemented as scriptable standalone programs. However if the functionality could also be made available as a library that could be called from compiled programs then the possibility exists of extending these programs to be able to easily read and write any number of different formats.

3.4 Conversions between different conventions

This is similar to the previous category, of conversions between file formats. Essentially the problem is that the same information (for example spacegroup symbols or coordinate systems) can be represented in different ways. A set of functions for performing the most common conversions would therefore save considerable programming effort. Conversions would include (but would not necessarily be limited to):

- Spacegroup symbol/number conversions
- Asymmetric unit conversions
- Inverting hand
- Moving between different coordinate systems

It should be noted that existing toolboxes such as Clipper already provide objects that perform these kinds of conversions "behind the scenes". So one approach to providing these functions would be to wrap e.g. the Clipper objects in small jiffy programs.

3.5 Manipulations of crystallographic data

To some extent these functions are extensions of the file format conversions mentioned previously, except that they are conversions of data within a particular file format. For reflection data the kinds of operations that are commonly performed include sorting and reindexing reflections; for density maps it includes extending the limits of the map or rotating the density into another frame of reference.

The CCP4 programs SORTMTZ, MTZUTILS, REBATCH, REINDEX, SFTOOLS and CAD together provide a small suite of programs for reflection data manipulations; similarly for maps there is MAPMASK and for coordinate data PDBSET and PDBCUR.

3.6 Basic crystallographic algorithms

These are the medium-level functions most often used in analysis steps, for example peak picking, solvent content analysis or estimation of the number of molecules in the asymmetric unit. By making these functions available as a library of functions available to compiled programs as well as at the script level, one has the option of building these algorithms easily into existing programs.

4. Language requirements for a PX software toolbox

The pipelines considered in the workshop used components written in a variety of different compiled and interpreted languages (the phrase “interpreted” is often used instead of the more common epithet “scripted”, as a language such as Python is powerful enough to be considered a fully fledged programming language). The breakdown of languages can be seen in appendix B.1 of the workshop report.

However this disguises the fact that generally it is only the high- and medium-level functionality that is implemented in compiled languages such as C or Fortran; the lower level functionality is more commonly implemented using an interpreted language. A key requirement for a toolbox to be used in this context is therefore that it should be *scriptable*, that is it should be accessible from within the context of interpreted languages such as shell, Tcl or Python (the key scripting languages used in the pipelines that were examined).

Traditionally this can be achieved most easily by providing the functionality as a compiled standalone program, which is then can be wrapped with a script interface or invoked directly from a script. This approach echoes the traditional Unix way of working, and has been adopted (largely successfully) by the CCP4 software suite. An alternative approach is to implement the functionality within a C or C++ library that has an API (application programming interface) that exposes the library to a scripting language. This approach has been adopted within the CCTBX project, which exposes C++ functions to Python.

5. Case studies: consideration of existing toolboxes

The workshop considered a number of different existing projects that could be characterised as toolboxes. It is possible to critique their suitability based on the criteria above.

- **Clipper** is a C++ crystallographic library that provides software components highly suited for writing crystallographic applications (and where it has been used very successfully). As a basis for building more automated applications it would seem to be an excellent choice of toolbox, however as it is not easily accessible from a scripting environment is not so useful in the context under consideration here.

- **CCTBX** is a set of crystallographic libraries that provide a wide range of different functions, ranging from fundamental algorithms to jiffy functions such as import and export to different crystallographic file formats. As such it has many of the desirable functions outlined above, easily accessible from within a scriptable environment. CCTBX would therefore appear to be a good choice of toolbox for developing new automated pipelines.
- **PyWARP** aims to provide a general framework for running programs and making decisions in automated fashion. It consists of two basic components, “controllers” (wrappers for running programs) and “deciders” (for decision-making). By providing an infrastructure for these two common components used in building automated pipelines, PyWARP would appear to be worth investigating. It is not clear however that PyWARP yet provides a generalised toolkit to be easily used outside of its original context.

Both CCP4 and CCPN were also considered as toolboxes. CCPN aims to provide an integrated environment and set of tools for working with NMR data and as such provides some useful insights into how a similar infrastructure might be created for PX. However it offers few tools that are presently directly useful for a PX software toolbox.

CCP4 has for a long time been treated as toolbox of useful functions that has been incorporated into a number of different projects. It can clearly fulfil many of the requirements that are outlined above in terms of the functionality that it can offer. The drawback is that in many cases the functionality is not as automatic as it could be, which leaves the pipeline developer with more work to do to use the programs as components.

6. Conclusions: recommendations for a PX software toolbox

This report doesn’t aim to completely specify a PX software toolbox; it is concerned with setting out recommendations for what such a toolbox might ideally look like if it is to meet the requirements of automated software pipeline developers.

The key functionality set can be summarised as:

- A set of standardised wrappers for the key program units used in automated pipelines, which standardise input to and output from those programs.
- A set of functions callable from compiled programs to enable reading and writing of XML marked-up files
- A set of functions to interrogate data file contents and header information
- A set of functions for automatically converting between common data file formats and common data conventions
- A set of basic manipulation and analysis functions for crystallographic data

This function set is based on the common functions identified by examining a number of existing pipeline initiatives. Providing these in a toolbox for all pipelines would encourage standardisation and code reuse, and help with issues of long-term maintenance.

The toolbox would be able to provide these functions in a variety of ways, ideally making them available both as a library available to compiled programs and as functions available from the scripting layer.

7. Next steps

The next step is to start trying to build a toolbox along the lines of based on reusing as much existing code as possible, and in particular looking more closely at the functionality available in libraries like Clipper and CCTBX. In either case it may be that the functionality can be accessed with relatively little extra effort.
Beyond that, the most obvious step is to define the XML output for the key high-level programs used in the pipelines, in order to create a set of wrappers or translators for extracting data from logfiles (this is discussed in the workshop report).

Finally further investigation is needed into which basic crystallographic algorithms should be generally provided, and what form they should take. To some extent this will rely on pipeline developers being willing to share their code and ideas freely.

Appendix A: Supplementary Material from the BIOXHIT/eHTPX/CCP4 Joint Workshop Automation standards and frameworks: *from data reduction to structure*

Wednesday 09 - Friday 11, February 2005 Jesus College, Cambridge
http://www.ebi.ac.uk/msd-srv/docs/9-11Feb2005/bioxhit05_2.html

Aims: A number of ongoing initiatives are interested in setting up so-called "structure determination software pipelines" which will be able to take X-ray diffraction data and produce atomic coordinates. One problem which is addressed by WorkPackages in the BioXHIT project is that of maintaining compatibility between these initiatives, by setting guidelines for:

(i) **Standards for frameworks for automation:** A number of different projects are currently involved in automating the PX structure determination process, each of which has its own architecture, philosophy and choice of components. In addition there are projects which have already automated parts of the pipeline, often highly successfully. Development of such pipelines would benefit greatly from agreed standards defining how components should interact, which would enable units from one development to be more easily slotted into another - in particular defining interfaces for different tasks in the structure solution software pipeline. As well as establishing useful standards this would inform BioXHIT task 5.1.1 ("Setting BIOXHIT guidelines for software developers")

(ii) **Standards for data exchange between computational units in the structure determination software pipeline:** BioXHIT task 5.1.2 ("Setting BIOXHIT guidelines for data exchange (Metadata)") aims to produce a data model which encompasses the data that are to be exchanged between the different "computational units" (programs or other crystallographic functions) used in software pipeline projects. This task will look at what meaningful data needs to be transferred "downstream" at each stage of the pipeline, as well as what information needs to be collected for decision-making and archiving (i.e. deposition) purposes. Addition to defining the data model in task 5.1.2, this will also inform BioXHIT task 5.2.1 (which is concerned with defining the data to be stored in a project tracking database).

(iii) **Toolboxes for Automation:** BIOXHIT task 4.7.1 is concerned with the requirements for the provision of a "PX software toolbox for automation". Such a toolbox should offer a set of basic functions which can support crystallographic applications, act as a "glue" in automated pipelines (for example by providing basic manipulations of data), and which could possibly be used to rapidly prototype new and novel applications. A number of developments already exist which could already be described as toolboxes and which offer a wide range of functionality. As well as defining the functions which a generic toolbox should offer, this task is also concerned with how the functionality should be presented (for example, which programming languages should the functions be accessed from?)

"Automation Standards & Frameworks": workshop report

1. Introduction

There are a number of ongoing initiatives for setting up automated structure determination software pipelines" that will take diffraction data from X-ray macromolecular crystallography experiments and produce atomic models. This report describes the joint BIOXHIT/e-HTPX/CCP4 workshop that was held at Jesus College, Cambridge from 9th-11th February 2005, to look at setting guidelines and standards for various software automation projects, to ensure compatibility and interoperability between developments (as part of the BIOXHIT commitment to standards). The workshop was divided into three sessions looking at different aspects of these projects, and this report provides summaries of the key points of each in section 2:

Section 2.1: Standards for frameworks for automation
Section 2.2: Standards for data exchange between computational units in the structure determination software pipeline
Section 2:3 Toolboxes for automation

The presentations themselves are available on the web at: **http://www.ebi.ac.uk/msd-srv/docs/9-11Feb2005/bioxhit05_2.html** (where provided by the speakers after the meeting) and should be referred to for more information. Section 3 summarises the conclusions and actions from the meeting. Appendix A provides a glossary that defines some of the terms used in this report. Appendix B summarises details of the various pipeline efforts based on proforma descriptions provided by the contributors. The proformas are also available on the web at **http://www.ebi.ac.uk/msd-srv/docs/bioxhit05_1.html**.

2. Summaries of the sessions

The following sections summarise each of the sessions from the workshop.

2.1 Standards for Frameworks for Automation

This session aimed to address some of the issues in BIOXHIT task 5.1.1 ("Setting BIOXHIT guidelines for software developers") and was based around presentations on a number of ongoing automated software pipeline developments:

- Decision-making in DNA (Graeme Winter, Daresbury Laboratory)
- AutoSHARP (Clemens Vonrhein, Global Phasing)
- CRANK (Steven Ness, University of Leiden)
- CHART (Paul Emsley, University of York)
- AutoRickshaw (Santosh Panjinkar, EMBL Hamburg)
- Small Molecule Pipelines (Simon Coles, University of Southampton)
- CS2 (Tassos Perrakis, NKI)
- PHENIX (Paul Adams)

In addition to these each of the speakers kindly provided proforma descriptions of their pipelines. The aim of the session was to set guidelines on how pipeline components should interact. One way to do this is to examine commonalities between the current pipeline developments. However the presentations made it clear that there are a wide range of different aims, methodologies, technologies and philosophies amongst these different projects.

Table 1 below summarises the key differences and commonalities:

Differences Commonalities

- Modular versus monolithic approaches (affects maintenance and extensibility)
- Process-driven versus data-driven approaches
- Speed of operation: for some systems (e.g. DNA, Auto-Rickshaw) speed is an issue and this can affect choices of programs and input parameters
- Variety of programming languages (Python, shell, Tcl, Scheme, Perl, Java)
- Variety of methods for storing and passing data (via common file formats such as MTZ and PDB, XML, grep-able flat files/directory hierarchies)
- Different target end users and delivery methods (e.g. webservice versus desktop, novice versus experienced users). This is also reflected in the differences in user interfacing and the level of control given to the user – for example, CRANK is

intended as a teaching aid and exposes more of the control to the user than e.g. DNA.

- Approach is to connect together large number of computational units to generate the pipeline functionality
- Common set of key computational units (SHELX, ARP/wARP, REFMAC…)
- CCP4 used as a source of jiffies
- Programs tend to be wrapped inside an interface layer that facilitates integration into the pipeline (although actual wrappers are specific to each development)
- A lot of effort is needed for converting/reformatting data (e.g. between data file formats or program input)
- A lot of effort is needed to parse and extract data from program output
- User interfaces are important

Table 1: Summary of differences and commonalities between pipeline developments

Some key points from the presentations (NB these are not intended to be summaries of the presentations themselves – please refer to the proforma descriptions and to the presentations themselves for more information):

DNA: highlighted issues of getting information from programs that form the computational units. This has been circumvented for some units (e.g. MOSFLM, SCALA) by adding new code to the programs to write out in XML. Using "grep" or equivalent is unsatisfactory as it is unstable (if program output changes at the next release) and scales poorly for large amounts of data (important if speed is a factor). However the DNA "standards" for its XML are ad hoc as there is no current standard for the content or format of the XML that is used.

AutoSHARP: starts with minimal user input and asks, "what do I know?" rather than "what do I need?" (i.e. a data-driven approach). Will always present information (notes/warnings/errors) to the user, which might trigger a decision outside the framework by the user. Issues with stability of its component computational units.

CRANK: runs a number of different crystallographic programs. XML is used ubiquitously within CRANK: input parameters, run commands and (most) decision-making criteria are stored in XML; XML is translated into input for specific programs; the program output is converted to XML storage.

Advantages of using XML:

- Object oriented datastore; language neutral external data structure
- Only need 2N conversion jiffies (rather than N2) to convert between formats
- XML is a standard in computing (so lots of tools are already available)
- Possibly of exchanging XML with other packages that also use this format

CRANK is implemented in Tcl within CCP4i (gives familiar interface, job control and help system – useful for both novices and experts). It is noted that the GUI separates "generic" and "specific" parameters (for datasets versus programs). Automated test set generation allows easy testing of CRANK with different parameters.

CHART: based on Scheme and developed up until 2000. Links together CCP4 programs plus SHELX. Data analysis (i.e. reading and understanding program logfiles) is the trickiest part (issues with parsing and interpretation). There were some issues with upgrading to CCP4 5.0.2 from 4.2 (e.g. changes to symmetry library). Notes that grep/awk are lightweight alternatives to XML. Parallelisation is possible by running several SHELX or MLPHARE jobs on different machines, but no suitable batch queuing system available.

Auto-Rickshaw: primary aim to achieve an interpretable electron density map & a partial structure in minimal time. Runs on a 16-CPU cluster and is available to users as a webservice. Characterise programs in different areas of parameter using speed as one of the criteria.

UK National Crystallography Service Grid Facility: Simon Coles described a complete highthroughput system (going from initial application through to automated structure solution) standardised as a Grid service, for small molecule crystallography. Automated structure solution uses “public domain programs” (mainly SHELX) in an intelligent manner and applies validation throughout the refinement. Dissemination is impeded by current publication protocols, circumvented by using an “open archive” solution (ecrystals.chem.soton.ac.uk) – this is possible because the process is very standardised.

PyWARP/CS2: pyWARP is new control system for ARP/wARP – made up of Python modules, stores data in and reads from XML. CS2 is toolbox developed to implement pyWARP. Uses data driven approach: a program is the means to move between decisions & decision-making is isolated from running the programs. A “step manager” stores dictionary values & the decision trajectory. History of decisions is stored in real time as XML. CS2 is designed to be easily extensible to other task (not just ARP/wARP).

PHENIX: aims to provide a environment:

- For users: automated structure solution that includes flexibility and control and a feedback system
- For developers: access to modern software tools that allow rapid development within an open system

Offers a number of interfaces (command line, automated strategies and “wizards”) for different users. Design considerations include long-term maintainability & reusability, requirement to be able to integrate different components, avoid duplication of efforts of others. Planned to use object serialisation to map data into storage and then provide import/export methods to existing file formats like MTZ – in practice MTZ & PDB etc files still needed to transfer information between programs (due to limitations of programming languages & program use).

Discussion & Comments

A number of issues were raised doing the discussion. It was recognised that defining standards is difficult for a dynamic field such as this, where there are always new algorithms, methodologies and programs appearing.

Defining standards should be possible in the case of archival standards (for final results) but is much harder to do for “intermediates”, which are very dependent on the user of the information and the algorithm. A prerequisite for useful information exchange is therefore the definition of realistic expectations about what is being exchanged. There was also an issue about whether data history should also be exchanged as part of this process.

It was suggested that an “object exchange mechanism” would be the best approach but is unrealistic (presumably because it requires that all programs use a common set of standard objects – unlikely to happen given the diversity of the available software). A hierarchical description of the data should be considered, as this would be able to capture the complexity of the data and would make it easier to deal with object changes. XML might be good choice (as it has a hierarchical structure), definitely try to avoid fixed formats (like PDB) and “flattened” formats (like mmCIF).

It is also important to separate the content (i.e. the information being transferred) from the format (i.e. the XML or other file format being used to transfer it).

2.2 Standards for Data Exchange between Computational Units in the Structure Determination Software Pipeline

The aim of this session was to contribute to BIOXHIT task 5.1.2 ("Setting BIOXHIT guidelines for data exchange (Metadata)") by examining what data is required at each stage of the structure determination process for:

- Decision making and feedback
- Data transfer
- Archiving and deposition

We hoped to be able to define a practical hierarchy of *functional blocks* (see glossary) within the pipeline. The interfaces of these blocks would be where we could usefully answer the questions above. Three broad areas were discussed, with introductory scene-setting presentations:

Section 2.2.1: Images to reduced reflection data
- Data Exchange with MOSFLM (Andrew Leslie)
- Diagnostics after processing with XDS, MOSFLM/SCALA & HKL2000 (George Sheldrick)

Section 2.2.2: Reduced reflection data to phases
- Phasing from heavy atoms (George Sheldrick)
- Phasing from molecular replacement (Airlie McCoy)

Section 2.2.3: Phase information to refined model
- Model building (Victor Lamzin)
- Refinement (Garib Murshudov)

The key points from the presentations and the discussions are given in the subsequent sections.

2.2.1 From images to reduced reflection data

Key points from the presentations:

MOSFLM: outlined the key sets of input and output parameters from MOSFLM (input includes diffraction images & physical parameters of the experiment, output includes batch MTZ file and a summary of the processing results). Some items are not currently calculated/output/stored (e.g. total elapsed exposure time) but could be if useful. Consideration needs to be given to other quantities (e.g. standard deviations of cell parameters) in order for these to be realistic/useful. The MTZ file output contains the principal information to be passed downstream. Recommended archiving the summary file for deposition purposes.

Diagnostics after data processing: the BIOXHIT test crystal (intended as a diagnostic tool, to provide a quick empirical method to check hardware/software pipeline) revealed differences in integrating/scaling software. Overall indicators of data quality were identified as

- Internal consistency index for equivalent reflections (Rint)
- Redundancy independent merging R-factor (Rrim)
- Precision indicating merging R-factor (Rpim)

Patterns in test crystal data gave indications of defects in detector/hardware, problems in experimental set-up (e.g. crystal not centred, or beam missing centre of rotation). Also suggests issues with integration software.

Discussion & Comments

Gerard Bricogne led the discussion section. He questioned whether "images to reduced reflection data" are the correct limits to be considering, and called for the standardisation of software rather than hardware. Data processing involves numerous programs, each of which still needs new features and which are likely to continue to evolve considerably in terms of their scientific content. Abstraction (moving away from thinking of individual programs, towards what they actually do) and "Automation standardisation is required to create new connectivity, moving away from the current "post-mortem" approach of data collection towards a closer relationship between the strategies of data collection and of structure solution. He recognised that both getting data out of programs (harvesting intermediate results & preserving their hierarchy) and getting non-default commands into programs (to circumvent the current limitations) are both issues to be addressed. Gerard proposed the notion of a "Protocol Descriptor", which will have a number of properties:

- Allow the description of a feasible data collection protocol in a specified experimental context, in such a way that the descriptors can be combined
- Enable ranking of protocols by simulation
- Able to be translated into sequence of commands in order to execute the experiment (this will require a command vocabulary)
- Able to monitor, evaluate and if necessary revise the process as it's happening
- Allow annotation

It was suggested that the DNA data model could be used as a starting point for these ideas. There are issues with how the evolution should be managed, the choice of communication tools, embedding in a data model which could also provide tools for defining beamline configuration files and then querying them, and the use of a tool like CCPN to turn the standard data model into a set of APIs for different media (Python, Java, XML, SQL, Perl).

2.2.2 Reduced reflection data to phases

Key points from the presentations:

Phasing from heavy atoms: George Sheldrick gave an overview of the SHELX pipeline for heavy atom substructure determination. Noted that it is important to work with umerged unsorted data, as there is more statistical information in unmerged data for assessing the quality of weak anomalous signals. Notes also that philosophy in SHELXD is to make assumptions and do things as quickly as possible (this seems to work) – for example it ignores user input of element type, as this information isn't used.

SHELXD requires α (angle added to calculated heavy atom phase φA to obtain the native phase φT) – notes that no CCP4 program calculates this (needs XPREP or SHELXC).

Critical parameters for SHELXD:

- Choice of resolution cut-off for the ÄF-data
- Number of sites to search for (should be within 20% of true value for occupancy refinement to work)
- Need to check if atoms lie on special positions

- Beginner's error is not to give SHELXD enough "tries" when searching for solutions – it can take 100+ tries to get one good solution

General assumptions for phasing (SHELXE):
 Don't refine heavy atom coordinates & occupancies output from SHELXD, and ignore Bvalues
 Only assume that one type of anomalous scatterer is present
 Ignore complicated probability distributions

PHASER/Phasing from molecular replacement: Airlie McCoy noted that there are two elements to consider: algorithms (computational units) and the pipeline (automation). With PHASER the interest is in improving the algorithms to push the boundaries and deal with difficult cases – better algorithms means better solutions (less need to consider multiple potential solutions, or cases where there are no solutions). Better automation allows more possibilities to be explored. Notes that PHASER has 9 distinct modes of operation for MR. Input to the program is via an "input object" that offers options of keyworded input for users, and direct input from Python scripting layer for developers. PHASER has "internal" & "external" archives for data transfer. Internal archives include:

- Results from rotation function with or without known parts of the structure
- Results of the translation function

External archives include:

- MTZ file containing phases and map coefficients
- PDB file with coordinates

Airlie commented that the results from different MR programs are not interchangeable (the results are inextricably linked with the algorithm used).

2.2.3 Phase information to refined model

Key points from the presentations:

Model building & ARP/wARP: Victor Lamzin observed that there are the following four functional blocks:

- Chain tracing
- Ligand building
- Helix search
- Solvent structure

The first of these is the main application of ARP/wARP. The required data are:

- Parameter file
- MTZ file with phases
- Optionally also coordinates (PDB) and/or sequence

Refinement & automatic molecular replacement: Garib Murshudov outlined what output information is required by external programs from refinement using REFMAC:

- R-factor before & after each refinement cycle

- Geometry and other internal consistency indicators
- Coefficients for map
- Refined coordinates

He also outlined a wish list of inputs into the refinement procedure (many of which are not yet used):

- Experimental reflection file (intensities/amplitudes and some other information e.g. phases – possibly provided in a lower symmetry spacegroup, or even unmerged data)
- Coordinate file
- Bond angles and other chemical information
- End signal – when to terminate the refinement process

For communications/data transfer, the following three areas need to be separated:

- *Communications*: use XML as a tool to pass information between procedures
- *Deposition*: the required information should be specified by the deposition centres
- *Debugging*: the program logfile is ultimately a diagnostic tool for the programmer and is not the ideal medium for communications.

Discussion & comments:
Victor Lamzin noted that in this part of the pipeline the ultimate goal is "the model", and that three things are needed to achieve the goal:

- The input (to be assessed)
- The path (to be chosen)
- The constraint (e.g. quality versus quantity)

2.3 Toolboxes for Automation

BIOXHIT task 4.7.1 ("Automation in Computation", and now 5.1.11 in the 2nd implementation plan) is concerned with gathering the requirements for the provision of a "PX software toolbox for automation". The requirements should be informed both by the needs identified by the existing automation efforts, and by considering the functionality offered by existing toolbox libraries. This session looked at a number of developments that could be described as toolboxes, and which offer a wide range of functionality. As well as defining the functions that a generic toolbox should offer the task is also concerned with how that functionality should be presented (for example the choice of programming languages). Presentations were made on the following toolbox/library projects:

- CCP4 Automation Project (Charles Ballard, Daresbury Laboratory)
- The Clipper Library (Paul Emsley, University of York)
- PyWARP (Marouane Ben Jelloul, NKI)
- CCPN (Rasmus Fogh)
- CCTBX (Paul Adams)

Key points from each presentation (NB these are not intended to be summaries of the presentations themselves – please refer to the presentations themselves for more information):

CCP4: provides large number of jiffies (FFT, SFALL etc) and "flagship" programs (REFMAC, MOLREP etc). Positive aspects include: common delivery to end user on a wide range of supported platforms; standardised environment and file formats; limited data tracking & "automation-lite" via CCP4i; strong contribution from clever developers; long term commitment to maintenance and curation of programs.

Negatives: difficult to extract information from program output; poor interface for scripting. Forthcoming developments: more marked up output; scripting interfaces to core libraries and jiffy code; extension to database outside of CCP4i. Plans: characterise program units (input/output/errors and key values), then abstract to XML and data model. Provide python wrappers to programs. Possibly move towards use of XML for interprocess communication, tracking & workflow, deposition, and decision-making.

Clipper: C++ libraries for X-ray crystallographic computation. Philosophy: extensible, replaceable, optional ("doctrine of fallibility"), & small (to reduce maintenance overhead). Key point is that objects have crystallographic knowledge (e.g. symmetry) built into them. Used in CCP4 molecular graphics, Coot, ARP/wARP.

PyWARP: provides framework for automation in the form of "controllers" (wrappers around programs) and "deciders" (make decisions about which program to run & which decision-maker to ask next, after the program has finished running). Work is in deciding where the decision points are and what the program modules are. It is possible to start with small modules, and then combine these to build up into larger modules. It was suggested that PyWARP could be extended into a general automation toolkit if there is sufficient interest in using these tools from other projects.

CCPN: goals are to make a data exchange standard for macromolecular NMR, and to promote software integration. Start off with an abstract data model expressed in UML (ObjectDomain, Rational Rose), & provide a standard API (auto-generated code) – support multiple storage formats, multiple programming languages. Interested in merging X-ray crystallography into this system.

CCTBX: philosophy is to provide fundamental crystallographic algorithms required for new software, and allow them to be used in the context of an interpreted language as the most efficient way of developing new algorithms & building pipelines. The CCTBX module is made up of libtbx (deals with software installation), scitbx (general scientific applications e.g. ffts and arrays) and cctbx (crystallographic functions e.g. unit cells, spacegroups, structure factors). Includes functions that can convert between different file formats automatically. Used for applications within PHENIX (e.g. HySS). Currently implementing fundamental algorithms for structure refinement and atom selection.

Comments & observations

It seems that the toolboxes presented in the workshop offered three distinct types of functionality:

- Scientific/crystallographic functions (e.g. calculating FFTs)
- Administrative functions (e.g. converting between file formats and conventions)
- Framework functions (e.g. infrastructure for creating pipeline, as in PyWARP)

In all cases there are some desirable features for a toolbox:

- Offer the ability to access the tools both "programmatically" and as functions from within both compiled (e.g. C/C++) and scripted languages (e.g. Python)
- Toolboxes shift some of the maintenance burden away from the core project onto the toolbox library – so there needs to be a commitment from the toolbox provider that it will be maintained and supported in the long term
- Toolboxes should adhere to available standards

Some analyses of the pipeline efforts presented at the meeting are given in Appendix B of this report; however these were not discussed at the meeting.

3. Conclusions and Actions

3.1 Key points

This workshop was successful in bring together many of the major European players in the automation of structure determination software, and beginning a dialogue between software developers about the role of standards in automation projects. Standardisation of the information that is exchanged between the key computational units used in automation projects has some clear benefits:

- Ensures that the programs are being used in the optimal way, i.e.
  Programs are provided with the correct input
  The output is interpreted and acted upon correctly
- Makes extracting key values from program output less difficult (and protects from changes in logfile formats)
- Provides a mechanism for validating input and output
- Avoids confusion over different quantities which have been given the same name (for example different programs calculate different R-factors), and the same quantity which can be expressed in different ways (for example spacegroup names)
- Makes it easier to integrate programs into pipelines (as the interfaces are well defined)

The immediate actions from the meeting (in section 3.2.1) focus on collecting standardised abstract descriptions (already used e.g. in CRANK) that can be used in this way. The abstract descriptions can initially be expressed in XML, but could ultimately be used to generate language-specific wrappers for the programs. It is also hoped that the minimum generic parameters can be identified as part of this process. The longer-term actions from the meeting (in section 3.2.2) focus on ways to create new connectivity within the structure determination pipeline, and particularly between the data collection/processing and the other components that are further "downstream". This task ultimately falls within the remit of Section 4 of the BIOXHIT project.

3.2 Actions from the meeting

3.2.1 Immediate Actions

This meeting identified a number of areas where standardisation would be beneficial in enabling computational units to be able to be used more easily within existing and future automation efforts, by having more formal definitions of program function, inputs and outputs. We note that *de facto* standards must already exist for a number of programs as a result of active ongoing collaborations between their authors (for example, REFMAC and ARP/wARP). Other information must already exist in some form in current automation projects (for example, hard coded in scripts). An agreed action from this meeting was for Charles Ballard (CCP4) and Avi Naim (EBI) to collect the following information from program authors and pipeline developers, for the key computational units used in the pipelines:

Program inputs

Files & input parameters
Plus annotation (description of meanings, formats etc)

Program outputs

Files and output parameters, with annotation (as for inputs)
Diagnostics associated with output:

What (range of) values (or trends) indicate that the program function was successful?
What values indicate that there is a problem
Characterise problems and suggest possible corrective actions

It is possible that multiple descriptions may be needed for each program, where that program is capable of multiple functions, for example in the case of PHASER – as Gerard Bricogne suggested, this will move away from thinking in terms of "programs" and towards thinking about what they do. Once these descriptions have been gathered they will be standardised into some form of XML and made available to the user community.

3.2.2 Longer-term actions

Gerard Bricogne observed that there is a significant overlap between the standards being developed in BIOXHIT Section 5, and the developments planned in Section 4 (which focuses on "Data Processing and Structure Determination"). The main determining elements for plans for the next year are:

- Growth of the DNA project as a natural "nucleation point" for a series of enhancements towards smart data collection
- Success of further integration efforts providing criteria for decision-making during data collection (e.g. Auto-Rickshaw)
- Growing awareness of specific requirements for data exchange standards

The main directions are then:

Contribute to extending DNA towards
Multiple wavelengths
Multiple passes
Inverse beam for single axis goniometers
A first set of ê goniometry capabilities

Recognise & extend data exchange with and within pipelines according to the agreed standards, concurrently with the progressive definition & implementation of these standards across the other sections & other projects.

3.3 Other issues

A number of issues were raised during the presentations and subsequent discussions that are not within the scope of the workshop, but which are mentioned here for future reference.

Dependencies: many programs depend on external libraries, and pipelines depend on external software packages. Dependencies are useful as they can reduce maintenance and development overheads, however changes to these dependencies can also have a detrimental impact on the programs and pipelines that use them.

Acknowledgement: to a large extent the pipelines rely on the excellent "third-party" software that is running underneath them, however as automation becomes more successful these programs will be increasingly hidden from the end user. How do we ensure that the authors of these programs receive due recognition?

Licensing: institutions are increasingly keen to commercialise software and this means the placement of tighter licensing restrictions that affect how the programs may be used within automated pipelines. Will this make the construction of effective pipelines impractical?

Appendix A: Glossary

In putting together this workshop we found that we needed terminology to express some of the concepts that we encountered. This glossary is our attempt to define what these terms mean.

Computational unit: any program, script, library function etc that is invoked from within an automated pipeline in order to perform some crystallographic task (for example, generating an FFT, locating heavy atoms or refining atomic coordinates).

Functional block: a conceptual unit that has a set of defined inputs and a corresponding set of outputs. A functional block could be the operation of a single program or of a section of pipeline.
A functional block could itself be made up of other smaller functional blocks.

Jiffy: a utility program or function to perform a basic operation such as data format conversions (e.g. between file formats or coordinate conventions). (From the CCP4 manual: "`Jiffy' has been adopted in this community to mean a small utility program c.f., mainstream jargon use as a unit of time.")

Pipeline: refers to the automated processes that run several components in an automatic fashion. It is not accurate to speak of these as "pipelines" (a term that implies a purely linear flow), as the processes using include looping and feedback. However this term has now achieved common currency and it is in this sense that the word pipeline is used here.

Appendix B: Analyses of automation developments

This appendix offers summaries of the information supplied by the pipeline developers in the proforma descriptions.

B.1 Programming languages

Table 2 below lists the main programming languages used in the automation projects discussed at this meeting. Some projects use more than one language. This is not a definitive list.

Language	Number of projects	Pipelines
Python	4	PyWARP/CS2, PHENIX, CRANK, Processor.py
Shell	3	AutoSHARP, Auto-Rickshaw, CRANK
C/C++	3	PHENIX, CRANK, Processor.py
Fortran	3	AutoSHARP (jiffies), PHENIX, CRANK
Perl	2	AutoSHARP, Auto-Rickshaw (GUI)
Tcl	1	CRANK
Java	1	Auto-Rickshaw (GUI)
Javascript	1	AutoSHARP (GUI)
Scheme	1	CHART

Table 2: Programming languages used in pipeline projects

B.2 Jiffy functions

Table 3 below groups the jiffy functions (as described in the proforma descriptions) into broad classes, with examples.

Class of function	**Example jiffies from pipelines**
Processing & analyzing program output	• MLPHARE logfile analyser (CHART) • Convert program output to XML (CRANK)
Querying/extracting information from formatted data files	• Extract information from MTZ, SCALEPACK & CCP4 map files (AutoSHARP) • Extracting information from reflection files (CRANK, Processor.py)
Format/convention conversions	• Convert bewteen asymmetric unit conventions (AutoSHARP) • Switching hand (AutoSHARP) • Mapmask (pyWARP) • Convert between reflection file formats (Processor.py)
Data manipulation	• SORTMTZ, MTZUTILS, REBATCH, REINDEX, CAD (Processor.py) • MAPMAN (Auto-Rickshaw)
Interactions with remote services	• Run BLAST (pyWARP) • Fetch PDB file from internet database (pyWARP)
Basic crystallographic operations	• Peak picking (AutoSHARP) • Solvent content analysis (AutoSHARP) • Determine most likely number of molecules in ASU (Processor.py)

Table 3: Classes of jiffy functions used in pipelines

B.3 Major programs

The table below lists the major crystallographic programs used in the pipelines, as described in the proforma descriptions. This is not a definitive list.

Pipeline	Programs
Processor.py	MOSFLM, SCALA, TRUNCATE, LABELIT, BEST
AutoSHARP	SHARP, SOLOMON, ARP/wARP
CHART	SHELXD, MLPHARE
PyWARP/CS2	REFMAC, FFT, ARP/wARP, PHASER
PHENIX	PHASER, SOLVE, RESOLVE, TEXTAL
Auto-Rickshaw	SHELXC/D/E, MLPHARE, NANTMRF, BP3, SHARP, DM, RESOLVE, ESSENS, ARP/wARP
CRANK	CRUNCH2, BP3, SOLOMON, SHELXC/D/E, DM

Table 4: Major programs used in pipeline projects

Acknowledgements

The BIOXHIT Project is funded by the European Commission with its FP6 Programme within the thematic area “Life sciences, genomics and biotechnology for health,” contract number LHSG-CT-2003-503420.
The CCP4 Project is supported by the BBSRC, by income from commercial distribution of the software, and by CCLRC Daresbury Laboratory.
e-HTPX is an e-science pilot project funded by the BBSRC.

The workshop programme was organised by Peter Briggs, Kim Henrick, Graeme Winter and Charles Ballard, with many thanks to Janet Copeland for the local organisation. We would also like to thank all the speakers and workshop delegates for their contributions.