REPORT

The First DNA 2.0 Spike

Abstract

The primary goal of the DNA (automateD collectioN of datA) collaboration is to provide a software framework to facilitate fully automatic data collection from macro-molecular crystals on SR (Synchrotron Radiation) beamlines. The future DNA version 2.0 will have a completely new design compared with the current DNA version 1.1 installed and used on SR beamlines. In order to test new frameworks and collaboration tools it was decided in the full DNA meeting at Diamond in February 2007 to make a "spike", i.e. a rapid development using new ideas without any commitments to future development. The first DNA 2.0 spike was concluded in a meeting held at the ESRF in June 2007. This report describes the results of this first DNA 2.0 spike.

Table of Contents

1. Introduction - Aim of the spike

The DNA (automateD collectioN of datA) project is a collaboration between synchrotron radiation facilities and laboratories with the aim of fully automating the collection and processing of macro-molecular X-ray crystallography data including rapid crystal screening. The DNA collaboration has successfully developed a software system capable of automatic screening, automatic ranking and automatic data collection with on-line data reduction (integration and scaling). For more information about the DNA project see the following links: DNA web site[1] and DNA article[2].

The initial collaboration divided the DNA system into a set of modules, where one developer was responsible for one module. This way of collaborating was initially successful but has made it difficult for new developers to enter the project. Moreover, the overall goals of DNA have gradually been extended so that the initial inflexible design had to be "stretched" in order to implement new features. This has led to code which is difficult to maintain and new features became more and more costly in terms of development efforts to implement. Therefore a decision was taken to start a completely new collaboration with a new way of collaborating and a completely new design. In software development it's a common practise to develop a prototype in order to test new ideas. Since the new version of DNA would have both a new way of collaborating and a new design it was decided in the full DNA meeting in Diamond (February 2007) to start the new DNA development by developing a prototype, also called a "spike". Here are the main ideas behind a spike development:

- The spike development should allow tests of different ideas without committing to future developments.
- All developed code within a spike should be considered to be for the spike only. If appropriate, the code can be reused later, but of course it should however all code could equally well be disposed of at the end of the spike.
- The experience gained through the spike development should be written up as a report and should be used for planning future developments.

When starting a new software development it is important to avoid re-inventing the wheel, i.e. If code is already developed and can be reused with no or little cost in terms of development efforts (and if its license allows it) it should be. At the time of defining the spike the following two frameworks relevant to the DNA developments were ready to be tested:

- The CCPN framework (see Appendix 5)
- The AAlib framework (see Appendix 6)

The initial aim was to use these two frameworks to develop a prototype which would run one of the most important parts of the automated data collection and processing chain : given a successful indexing of reference images the calculation of the strategy for the data collection.

We also wanted to test many other things during the spike development:

- A project management tool (Trac, SourceForge, Google code etc, see Appendix 8)
- A new video conference tool (Marratech, see Appendix 12)

1 http://www.dna.ac.uk
2 http://journals.iucr.org/d/issues/2002/11/00/ba5026/index.html

- If enough time different programming languages (Python and Java)

The work with the prototype started with adopting the spike development plan (see Appendix 1).

2. Evolution of the spike development

As decided in the spike plan (see Appendix 1), the development of the spike application and the plugins was based on AALib and the CCPN framework was used inside three plugins to implement the BioXDM-style data model APIs. The spike developers could thus test both AALib features (rapid application development, logging, mutli-threaded plugin manager, external process manager etc, see Appendix 6) and the CCPN framework (automatically generated APIs and data persistence, see Appendix 5).

The first part of the spike was concentrated around the development of the BioXDM-style data model. The main goal of this data model is to model the information needed by and output from different data processing programs. This data model was developed from scratch with many developers contributing.

During the spike development of the BioXDM style data model a part of the development team felt that this model had too strong dependence on the CCPN framework and the modelled objects were too strongly coupled (coupling is a measure of how strongly one element is connected to, has knowledge of or relies on other elements). Therefore the development of a second data model "Experiment-StrategyPoint" was initiated. The main design goals for this second data model were to make it independent of any framework and its modelled object as uncoupled as possible.

Due to the difficulties of interpreting the BEST output (more HTML-like syntax than XML) two external packages were introduced: 4 Suite XML and Amara. The Amara package (which is dependent on 4 Suite XML) is used to parse the BEST XML output.

A working prototype was finalised (see Appendix 2). The last weeks of the spike were used for writing the spike report (Appendices 2 – 19).

The spike was concluded in a “DNA Technical Workshop” organised at the ESRF between June 5th and June 7th. The minutes from this meeting are presented in the next chapter.

3. Conclusions of the spike prior to the workshop

It is impossible to draw a single conclusion after this work because we have tested many different things: frameworks, collaboration tools etc. We will in this section draw a couple of conclusions. However the final conclusions and decisions for the DNA 2.0 developments will be made during the spike / DNA 2.0 developers meeting in June 2007.

- The development of the BioXDM-style data model was implemented according to with the spike plan. Even though the absence of ObjectDomain prevented testing of the modelling part by all developers, we could test the generated API and the underlying CCPN framework. The API works within the prototype, and so the main initial aim of the prototype has been reached: The prototype starts BEST with a given command line, it recuperates the results, it populates the BioXDM-style data model with the results and finally an XML file is generated which contains the results.
- The most important conclusion from the spike development is that we cannot develop a data model without having a set of clearly defined use cases. These use cases must define the

overall process, from the first to the last processing steps. The idea to develop a data model centred around the strategy turned out to be difficult if not impossible to realise without use cases that define the whole process, especially the following data processing steps. The absence of these use cases made the discussion around the data model very confusing, because it was not clear for all developers what information from the strategy needed to be kept in the data model for following data processing. Therefore the first development goal for DNA 2.0 must be the development of a complete set of use cases (however before starting development of DNA 2.0 we must sign a project agreement and agree on the DNA 2.0 license).

- The prototype development was based on the Aalib framework. During the development improvements were suggested but no major drawbacks of this framework were discovered. Therefore we conclude that the Aalib framework should continued to be used for the DNA 2.0 developments.
- The testing of the CCPN framework resulted in a list of short-comings. The most serious one is that the API generated by the CCPN framework cannot be unit tested independently. By design, all CCPN objects must be created with all their hierarchy, hence there is no obvious mechanism to independently instantiate them and unit test an application that uses the API. Because of this, and other shortcomings, it is not possible at this stage for us to recommend the usage of the CCPN framework in its current form in the development of the DNA 2.0 core, although it could be used as a persistence mechanism independently of its full adoption.
- As the development of the "Experiment-StrategyPoint" data model started in the middle of the spike, the limited time did not allow the implementation and testing of this second data model. The conclusion is therefore that for the development of the DNA 2.0 data model we must continue to examine this proposal.
- The combination of eclipse IDE and subversion worked very well and should be continued to be used for the DNA 2.0 developments.
- The Marratech video conference tool worked very well a part from the limitation of 10 seats. We should continue to use Marratech unless we cannot increase the number of seats and an equivalent tool allowing more seats is found.
- The Diamond hosting of the Trac and subversion servers worked well but also had disadvantages. The federal IDs were felt to be an obstacle to the development for many reasons, mainly because it is hard to remember which federal ID belongs to which developer. A translation from federal ID to developer name was partly implemented in the Trac server but will not be possible to implement for the subversion repository. The conclusion is therefore that we must discuss alternatives, e.g. SourceForge (private or public), in the June meeting.

4. Minutes from the DNA technical workshop Wednesday 6th – Thursday 7th June 2007

Presents:

Alun Ashton (Diamond, UK), Gleb Bourenkov (EMBL Hamburg, Germany), Gerard Bricogne (Global Phasing, UK), Sandor Brockhauser (EMBL, Grenoble), Jose Gabadhino (ESRF, France), Alexander Gobbo (EMBL Grenoble, France), Elspeth Gordon (ESRF, France), Marie Françoise Incardona (ESRF, France), Pierre Legrand (Soleil, France), Andrew Leslie (MRC LMB, UK), Karl Levik (Diamond, UK), Peter Keller (Global Phasing, UK), Romeu Pieritz, (ESRF, France), Sasha Popov (ESRF, France), Harry Powell (MRC LMB, UK), Takasi Tomikazi (SLS, Switzerland), Peter Turner (Australian Synchrotron), Darren Spruce (ESRF, France), Olof Svensson (ESRF, France)

Partly presents:

Rudolf Dimper (ESRF, France), Stephanie Monaco (ESRF, France), Vincente Rey (ESRF, France)

4.1 Introduction – presentation and demonstration of the prototype

The workshop was opened with a short introduction by Olof and a demonstration of the prototype by Karl.

4.2 Project agreement

There was an unanimous agreement that a project agreement is essential. The PIMS agreement was suggested to be used as a template. Alun agreed to circulate a first draft by 15th June.

The executive committee will be responsible to decide who will sign the agreement. The individual executive members will have to decide whether they want to involve their respective technology transfer divisions.

Initial partners for the DNA 2.0 project are : ESRF, EMBL Grenoble, EMBL Hamburg, Soleil, Global Phasing, Diamond, MRC-LMB. The project agreement should be signed in a DNA 2.0 meeting to be organised in October.

4.3 Licence

Two possible licensing schemes were discussed :

- Open Source : People can take and re-use code provided that they acknowledge the original authors (open source with acknowledgements). This could help with certain kinds of funding.
- Restrictive licence with the possibility to prevent exploitations. This could help with certain kinds of funding.

Conclusion of the licence discussion : We will make a core system which is open source and this core will have all the functionality as described in the scientific case. Additional plugins could be developed under restrictive licence. The intention to develop a plugin under either licence by any collaborating partner but not as part or within the structure of the collaboration should be disclosed to all partners in the collaboration. The precise definition of the functionality of the core will be

clarified as an annexe of the project agreement.

4.4 Architecture of DNA 2.0

The draft DNA 2.0 architecture (Appendix 16) was presented and discussed. There was a discussion whether a work flow tool should be used but no conclusion was made.

The question was raised if DNA 2.0 should contain a GUI, given that some facilities will anyway use their own GUI (e.g. ESRF). We agreed that DNA 2.0 should contain an optional GUI. If this GUI is written in Java it should fit into the Diamond GDA (beamline control software).

There was also a discussion about the programming language for DNA 2.0 - Java or Python (see Appendix 15). As AAlib cannot be rewritten in Java without considerable effort we decided that Python v. 2.5 or higher should be adopted for DNA 2.0.

We agreed that DNA 2.0 should support distributed / grid computing.

The details of how this support is implemented will be decided in forthcoming meetings.

4.5 CCPN framework

Unless CCPN can provide a realistic time frame to provide solutions to shortcomings revealed during the spike we should not consider using the CCPN framework for core DNA 2.0 developments. A list of questions and comments will be prepared by Andrew and Olof and will be sent to the CCPN developers [Report sent to CCPN developers on August 2nd].

The identified shortcomings are:

- The current dependence on ObjectDomain and its lack of availability
- The inability to conduct unit tests on code which use the CCPN auto-generated API
- The CCPN framework imposes too strong dependency in the data model, e.g. memops root object, single composition hierarchy and the complex data type
- The licence : clarifications are needed

Comments raised in the report but not discussed in the meeting will also be fed back.

4.6 Alternative to the CCPN framework

We agreed that a data modelling tool is essential for the development of the DNA 2.0 data model and as we cannot rely on ObjectDomain we discussed several of alternatives: Umbrello, MagicDraw, Rational Rose and Enterprise Architect.

The possibility of auto generation of the API (as provided by the CCPN framework) was considered to be beneficial but not compulsory. While many data modelling tools do provide basic API generation very few allow incorporation of special code in the data model (also provided by the CCPN framework). One example cited was that code provided in the model can be incorporated automatically by MagicDraw (however Java centric).

Data model

We discussed how we might find an objective procedure for evaluating the relative advantages of different data models as the limited time in the spike did not allow for such an evaluation of the two

proposed data models. We decided to choose one or two use cases for studying one data model, see conclusions.

4.7 Development plan

Milestones :

- October 2007 : see conclusion
- January 2008 : data model for two use cases
- June 2008 for a prototype implementing at least one use case, for example data collection allowing for radiation damage (varying exposure time during collection). Should run on a beamline. This initial prototype will not contain a GUI.

5. Conclusions of the workshop

5.1 Project agreement

We concluded that the first step, before starting to develop DNA 2.0, must be to sign a project agreement. The earliest such an agreement could be ready to be signed was agreed to be October 2007 (because of summer holidays and conferences). A "DNA 2.0 Project Agreement Meeting" should be organised at the ESRF in October

where this project agreement should be signed.

The project agreement should contain the following items:

- Project name
- Objectives of the collaboration
- Expectations of the involved partners
- Licence
- Definition of the DNA 2.0 "core"
- Team and management organisation – definitions who does what, time available
- Disclosures, private developments

- The executive committee should decide who should sign the project agreement.
- Alun and Olof are responsible for coordinating this work.

5.2 Use cases

It was also concluded that before any development can start on the DNA 2.0 data model, a number of use cases must be defined. The

use cases should be based on scientific case for DNA 2.0

The following people where suggested to form a "Use Case Working Group" : Elspeth (ESRF),

Johan (York), Gerard (Global Phasing), Andrew (MRC LMB), Alun (Diamond) and Olof (ESRF). This working group should work on the following tasks:

- Define priorities between the use cases – not ruling out possible extensions even if remote
- Definition of terms used in the use cases
- Work flow, i.e. sequence of events
- Storage (persistence) of data

The computing developers should follow the work of the use case working group and examine possible

implications for the architecture, e.g. concurrency, parallel, sequential execution

, constraints, risks and feasibility.

The use cases should contain the following information:

- Starting requirements
- Success and failure scenarios
- Recommendations for validation tests

We suggested that at least two fully described use cases should be developed till the October meeting: the one proposed by Sasha (data collection allowing for radiation damage, varying exposure time during collection)

and the other one to be defined. Olof is responsible for coordinating the work on the use cases.

5.3 Data model spike

Even though the work on the data model cannot start before we have defined the use cases, we decided to make Data Model

spike. The goal of this

spike should be to develop a “broad” data model based on two use cases to be finished for the October meeting.

The data model spike is coordinated jointly by Marie-Françoise and Peter. Olof, Romeu, Pierre and Sandor are members of the spike.

5.4 Methods used for producing implementations of the model

We decided to set up a working group that will look for alternatives to ObjectDomain for as a UML modeling tool. The tool should ideally run on Linux, Windows and MacOSX. The

tool should support API generation in both python and java

. The generated APIs should support concurrence and file system persistence.

This working group is coordinated by Marie-Françoise. Working group members : Olof, Romeu, Pierre, Sandor and Peter.

5.5 Organisation of the October meeting

Olof will organise the October meeting. Open access should be provided to reports and code prior to meeting. Reports should be available two weeks before the meeting

.

5.6 Write up of Spike report – BioXHIT deliverable

This report will be used as a BioXHIT deliverable.

The code generated in the spike should be released together with this report.

Appendix 1: DNA 2.0 spike development plan (presented in the beginning of the spike)

Goal of the spike:

To produce a prototype which should implement one or several use cases (see below). This prototype should be ready by the end of the spike working period. The experience gained during the work of the spike should be written up as a document which should contain recommendations concerning the DNA 2.0 developments. The spike conclusions and the DNA 2.0 recommendations should be presented in a DNA 2.0 workshop to be held either late May or early June.

Spike developers:

The following developers will actively work with the spike, (either) by working on the data model and (/or) working on the prototype:

EMBL Grenoble:	Sandor
EMBL Hamburg:	Gleb, Sasha and Johan Unge
ESRF:	Marie-Françoise, Romeu and Olof
Global phasing:	Peter
MRC LMB:	Harry
Soleil:	Pierre

The following people will not actively participate in the development but they might participate in VC meetings, they are subscribed to the spike mailing list so they will follow the developments very closely:

Diamond:	Alun and Karl
Global Phasing	Gerard
MRC LMB:	Andrew

Use cases:

The spike development will be centred around the strategy calculation. Given (one or several) indexed reference image(s), the prototype should be able to run the program BEST and present the results. The input parameters to the prototype could either be give on the command line of the prototype or as an XML document based on the data model to be developed. The results should be both in form of log messages and XML format. The prototype should be able to perform the following use cases:

1. Simple BEST strategy

The following two use cases will be refined once the definitions have been clarified:

2. Multi-wedge BEST strategy

3. Non-continuous multi-wedge strategy

(Note that we have left out the MAD and kappa strategies as these are more complicated and thus don't fit well in the spike time-frame).

Development details:

The data model will be developed following the guidelines in the BioXDM project.

The prototype will be developed in Python (version 2.5). Ideally the same prototype should be developed in Java in order to make a comparison between the two languages. The python implementation should have priority but if time allows we could try to develop the same prototype in Java. The prototype should run on Linux, Windows and Macintosh.

The prototype will be developed using the AALib framework. The AAlib framework will be presented to all spike developers during the initial phase of the spike. The code written should follow guidelines which we should agree upon during the first phase of the spike; e.g. use of class name spaces, naming of classes, methods and variables etc.

The prototype should have both unit tests and validation tests.

The prototype should be able to store the results in a persistent manner, i.e. the data model should be able to take into account and store the results from many strategy calculations of the same initial reference images.

The prototype should be able to write a log file with log and error messages. The results should also be available as HTML pages.

If an error occurs during the processing the prototype should handle the error in a robust manner and present clear error messages.

The development IDE will be eclipse.

Concerning the project management tools we didn't take any decision during the meeting, instead we agreed on comparing these three solutions:

- Online sourceforge project
- Private sourceforge project - free for up to 15 developers
- Diamond based solution (subversion, trac, wiki etc.)

A final decision will be taken during our next VC (Friday March 9th).

The development team will meet regularly, i.e. at least once a week, using the Marratech VC software (unless any facility cannot use it, e.g. Diamond has some difficulties).

Appendix 2: Presentation of the prototype

Objectives

The goal of the prototype work was to develop code that interfaces with BEST to perform the strategy step of data collection. The input parameters should refer to one or more indexed reference image(s). The parameters could either be given on the command line or as an XML document based on the data model.

The results from the execution should be stored in a persistent manner, i.e. the data model should take into account and store the results from many strategy calculations of the same initial reference images. The log and error messages should be written to a log file, and the results should also be available in HTML format.

If an error occurs during the processing, the prototype should handle the error in a robust manner and present clear error messages.

Use cases

The prototype should be able to perform the following use cases:

1. Simple BEST strategy. (Single wedge with single sub-wedge)
2. Multi-wedge BEST strategy. (Single wedge with multiple sub-wedges)
3. Non-continuous multi-wedge strategy. (Multiple wedges with multiple sub-wedges)

MAD and kappa strategies have been left out as these are more complicated and thus don't fit well in the spike time-frame.

The code

The programming language is Python 2.5 and the code uses the AALib framework.

The prototype consists basically of core modules, plugins and the CCPN-generated API:

Core modules

The core modules are mainly composed of the main application (dnaspikeapplication.py), configuration modules (DSDefinition.py: specific to the application and DSPython.py: specific to python), and an import module of all the objects needed by AALib (DSKernel.py).

Plugins

A plugin is invoked by giving its corresponding command line option when you run dnaspikeapplication.py. E.g.:

```
python ../modules/dnaspikeapplication.py --bestCmd --verbose \
--datasource=../tests/test-Best/dataset96 -b ../libraries/best-3.1 \
-e $HOME/.DSBESTExecution -p $HOME/.DSBESTExecution/DSPersistedData001
```

Here the `--bestCmd` will invoke the DSPluginBestCmd.py plugin. The mapping between command line options and plugins is given in DSPluginCommandLine.py.

CCPN-generated API

This API provides a persistence framework for the prototype. It has getter and setter methods, validation methods to check constraints, finder methods and methods to construct new instances of child classes. The API was generated automatically from the BioXDM-style data model using the CCPN machinery.

File hierarchy

Programming language, libraries and external dependencies

Python

From the Python web site:

> Python is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days. `[...]` Python is distributed under an OSI-approved open source license that makes it free to use, even for commercial products.

AALib

The Asynchronous Action Library project (AALib) is a framework based on pure object-oriented concepts to facilitate asynchronous communication between threads (thread safety) and external

processes, allowing the development of complete applications in different programming languages and for different platforms.

AALib was developed using Python for the basic layer and designed to be natively translated to Java.

AALib is Open Source software and distributed with a licence that allows for the maximum freedom of use.

BEST

From the BEST 3.1 manual:

> BEST is a program for optimal planning of X-ray data collection from protein crystals. The method employed in the program is based on the modelling of statistical characteristics of the data yet to be collected using the information derived from a few initial images. Diffraction anisotropy, crystal radiation damage, anomalous, geometrical restrictions (e.g. spot overlapping) and hardware limitations are taken into account.

BEST produces an XML file that is parsed by the prototype and the results are persisted through a CCPN-generated API; that is, the results are written to an BioXDM-style data model.

CCPN library

The aim of the CCPN project is to perform a service for NMR spectroscopists analogous to that of CCP4 in the X-ray community. The CCPN machinery has been used to generate an API for persisting the results from an execution of the prototype into the BioXDM-style data model. This API depends on the CCPN python library.

Amara

From the Amara web page:

> Amara XML toolkit is an open-source collection of Python tools for XML processing, not just tools that happen to be written in Python, but tools built from the ground up to use Python idioms and take advantage of the many advantages of Python over other programming languages.

In the Spike, Amara is used to parse the XML output from BEST. Amara depends on 4suite:

4suite

4suite is library of integrated tools for XML (and RDF) processing written in Python. The only reason why 4suite was used in the Spike is that Amara depends on it.

Environments

The prototype has been designed to run in Linux, Windows and Mac OSX. However, some minor practical details remain before it will work on Windows and Mac OSX.

Functionality not implemented

Due to severe time constraints, some features were not implemented at this stage:

- The prototype currently only runs on Linux. (Although it's probably not a major task to

make it work on Windows and Mac OSX.)
- It should be possible to give the BEST parameters on the command line or in a configuration file, but currently they have to be hardcoded in plugins/DNASpkie-plugins-v1.0/best-cmd/DSPluginBestCmd.py.
- Input parameters can only be given on the command line, not in an XML file.
- Results are not written to HTML.
- Human-readable output from BEST is not saved.

Limitations due to external dependencies

The format of the BEST XML file makes it difficult to use the standard Python XML parsing module, which is why we chose to use Amara instead. With some modifications to the format, it would be much easier to use the standard Python XML parsing module. It has been noted that from an ideological/political perspective, it would be preferable if DNA 2.0 would work without too many external dependencies.

Prototype feedback

A minor detail: As one would expect, if one forgets to add the CCPN python library to PYTHONPATH, the result data is not persisted. However, there are no error messages to indicate that something went wrong (except the Python error trace in the log file) or why it didn't work.

Appendix 3: BioXDM-style model for DNA spike prototype

Aims

The initial aims of this modelling exercise were:

1. to produce a model in the UML of BEST's input and output data as a collaboration between all interested participants in the DNA spike, and
2. to use the CCPN machinery to generate a Python API from the model that the prototype application would use to handle the input and output data to BEST.

In view of time constraints, the scope was restricted to BEST's output data only.

N.B. The name BioXDM comes from a BioXHIT associated project which is exploring the use of the CCPN approach in "smart data collection" - see https://www.bioxhit.org . Unlike CCPN, BioXDM does not propose to model every detail of data strategy, collection and processing, but restricts itself to the data that is required to be exchanged between the applications involved.

The Modelling Process

The original plan was to enable interested participants to contribute to the data model directly using the ObjectDomain modelling tool. Unfortunately, ObjectDomain became unavailable before the spike began. We decided that for the spike Peter Keller would maintain the model using his existing ObjectDomain licence. The model was developed through video conferencing discussions. Class diagrams of the evolving model were presented at these conferences and also checked in to the spike's Subversion repository.

The participants had varying levels of familiarity with the UML and some explanation of the notation were required during the discussions. It was probably helpful that the participants were involved in the modelling process from the start, rather than being presented with a model that already had been partly developed independently.

The unavailability of ObjectDomain will be addressed in the future by CCPN migrating to a different modelling tool, and CCPN has announced plans to do this. [At the time of writing, ObjectDomain's web site is available once again, but this will not necessarily change CCPN's migration plan.]

Evolution of the model

The starting point for this data model was an examination of the data handled by the BEST strategy program. As the model was developed and understanding of the structure of the data was refined, it became progressively less BEST-specific. Having said that, the development of the model was stopped when it was judged sufficient to support the prototype rather than satisfying any particular criteria of genericity.

The resulting model, with 11 classifiers (9 classes and 2 complex types), is small. However it reflects the outcome of very detailed discussion and input from several people. The major points dealt with during this process are:

1. Accurate separation of BEST's input and output data

   Some data that appears in the BEST output file is in reality input to BEST's strategy calculation. This data is best considered as the result of pre-processing of MOSFLM, Denzo or XDS indexing output.

2. Clarification of the meanings of Wedge and SubWedge and their relationship to strategies in general and BEST strategies in particular.

   Briefly: within a Wedge, adjacent images abut accurately so that partial reflections can be summed. Within a SubWedge, the images have a constant oscillation width and exposure time. Any re-orientation of the sample starts a new Wedge.

3. Distinguishing between (i) the data that define how a strategy is carried out ("actionable items", e.g. detector distance, exposure time) and (ii) data that are derived characteristics of the strategy (e.g. completeness, predicted reflection statistics).
4. Inclusion of classes to represent groups of strategies and manual strategies.
5. The reduction in BEST-specificity mentioned above.

There are some scientific shortcomings within the area of coverage of the model that we decided not to address within the spike. These include:

1. Proper structuring of instrument-level parameters (source, goniometer and detector) and modelling of their relationships
2. Separation of the purely geometric parameters/statistics of data collection (e.g. oscillation width, axis settings, completeness, redundancy) from those that relate to the interaction of the sample with the X-rays (e.g. image exposure time, predicted resolution limits, predicted intensity/sigma).
3. No assessment of compatibility of this model with data collection strategy software other than BEST has been done.

CCPN-specific UML usage

The CCPN methodology requires the data model to have a hierarchical structure, namely each class must have one and only one parent class within its package (except for the "Top Object" of a package, which must be a child of the memops.Implementation.MemopsRoot class, and memops.Implementation.MemopsRoot itself which has no parent class). This is so that there is a single path from memops.Implementation.MemopsRoot to any object in a system, which is a property required by the CCPN framework.

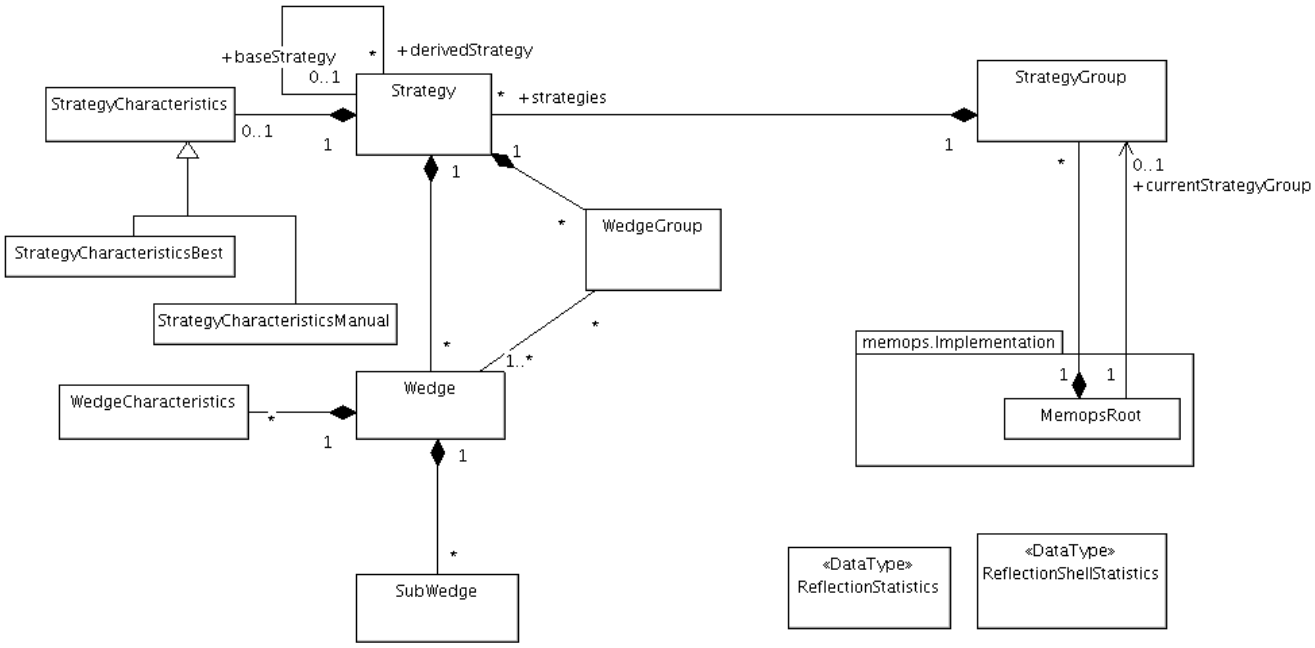
A class' parent class is indicated in the class diagrams by the UML composition association, with the solid diamond at the parent end. (This is a departure from standard UML usage).

This means that a class cannot be modelled with more than one composition association (since it can only have one parent class). This restriction is partially compensated for by CCPN support for complex types that have attributes and methods (although not associations), and can be used to declare the types of attributes of other classes. These are indicated on the class diagrams with the

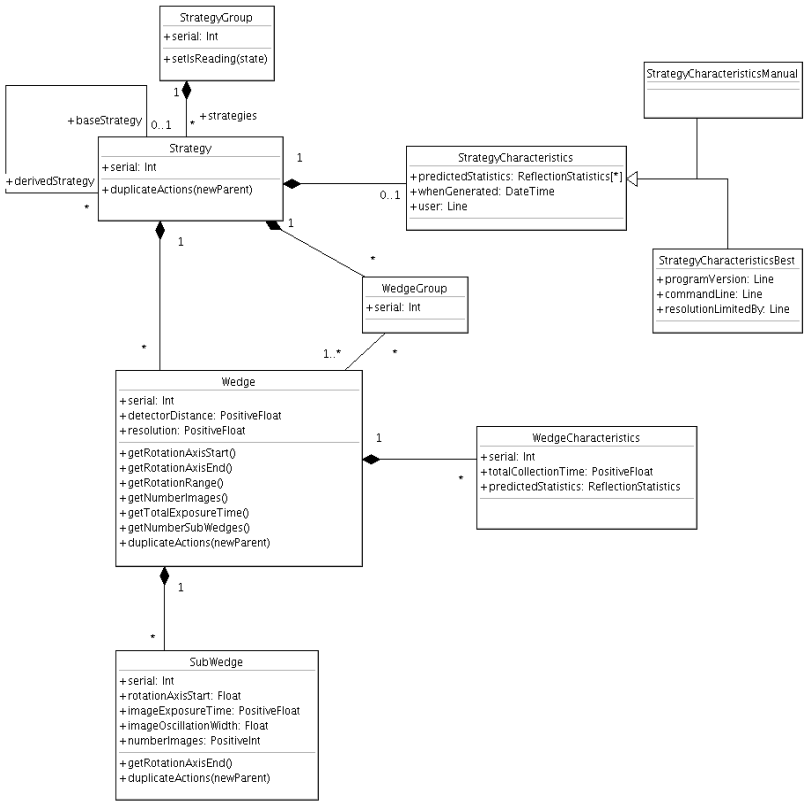
<<DataType>> stereotype. This stereotype is also used for simple data types that are restrictions on primitives (such as PositiveFloat): the two types are distinguished by the fact that simple types have no attributes or methods and only inherit from other simple types. Note that normal classes cannot be used in the CCPN machinery to type attributes: only <<DataType>>'s can.

Class diagrams for data model

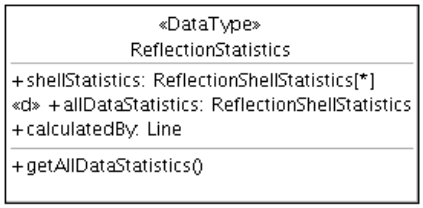
Overview

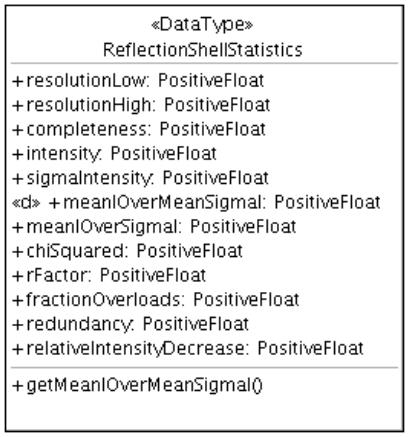


Detailed view of classes



Detailed view of types





BioXDM-style Data Model Feedback:

- The Data Model is closely linked to how the ccpn machinery works:
    - MemopsRoot ccpn root object
    - Single Composition hierarchy
    - The class definition of a complex data type is used to type attributes of other classes. The CCPN framework does not support using a conventional class for attribute typing

    For this reason, inherits from ccpn cons in terms of flexibility and can't be used outside ccpn context.

- Atomicity lack

The BioXDM-style data model is an abstract description of scientific concepts and terminology, in terms of class diagrams in the UML. Some of these terms may not routinely be given precise definitions by the scientists who use them. Implementations of this model should be created in order to support various programming language interfaces to a common data storage; this is an approach to support consistent data exchange between applications. The process of creating the model itself was only loosely coupled with those implementations, since much of the discussion about the model's structure and content was at the level of higher level scientific concepts, not about the details of implementation. We could have coded implementation(s) of the data model by hand in one or more programming languages, but this would have been difficult in the spike's time-frame. This was one reason for using CCPN in the spike: we used it to auto-generate a Python implementation to realise the data model.

Appendix 4: Data Model – “Experiment-StrategyPoints”

Introduction

This data model was developed within the spike and is proposed as an alternative to be analysed and improved by the DNA2 spike working group. The main objective of this proposal is to present the concepts of a generic and simple data model to represent the strategy.

The main “user and software requirements” used to design this proposal are discussed in the document “Diamond-Bioxhit-20060628.ppt” - Pieritz, Svensson and Sean McSweeney (presented at Diamond-UK Meeting in June-2006). The main idea was the design of a new abstraction for DNA2 based on “The Experiment” (resumed here):

- All processing and historical data will be stored in an “Experimental Project Record”;
- Different Beamlines or equipments can be used in the same “Experiment” simultaneously;
- Customizable automation schemes;
- Experiment can be either automated or manual Data Collection of a Single Crystal or Crystal Sets;
- Based on a “Dynamic Data Model”;
- Back Compatibility: it uses the current and future Databases and Hardware.

Objectives of the Data Model

- The main characteristic of this data model is the abstraction of the Strategy as a set of basic Points representing the Status of the scientific experiment/method and its devices;
- To be flexible : It is possible to add new devices and scientific methodologies to each “StrategyPoint” using the same code engine;
- Preserving the current and future scientific abstractions: i.e. : The “Groups” abstraction to conserve Wedges and SubWedges;
- Store all information used and generated by the strategy engines;
- Use only Objects with simple and standard definitions;
- Each Object is independent and self-contained = no hierarchy to generate and manage entities;
- The data persistence is assured by each individual object and is preserved when it is shared with other entities (duplication or other operation);
- Simple code maintenance;
- Unittest capabilities;
- Code reliability;
- No necessary external licenses or agreements.

Overall Concept

- The group of strategies presented in this data model are assembled in the DSProjectStrategy* class (it is a container);
- The DSProjectStrategy has a set of DSStrategy (it is a container) implemented by its child classes: DSStrategyKappa, DSStrategyBest, etc. ;
- Each DSStrategy is defined by a set of DSStrategyPoint(s);
- The DSStrategyPoint represents a container of device status and scientific status. It is not a

geometric representation of any kind;

- The DSState represents the parameters (resolution, angles, etc) of different “ideal” devices (i.e.: DSStateBeam, DSStateDetector, DSStateGoniometer or/and others) and Scientific status (i.e.: DSStateWedges, DSStateSubWedges?? and others); DSStrategyPoints of similar devices/scientific states can be assembled within particular Groups. For example, a wedge is a group of points that have same Detector resolution. This mechanism allows new status devices and scientific abstractions to be added to the current or the future version of an existant DSStrategyPoint container.
- Each DSStrategy can share its DSStrategyPoint(s) set or an individual element with other strategies to compose complex solutions;
- Each DSStrategy class is specialized by its childs implementing a single method to “parse in and out” the information from the external strategy engine (i.e.: DSStrategyBest, DSStrategyManual, DSStrategyKappa, etc.);

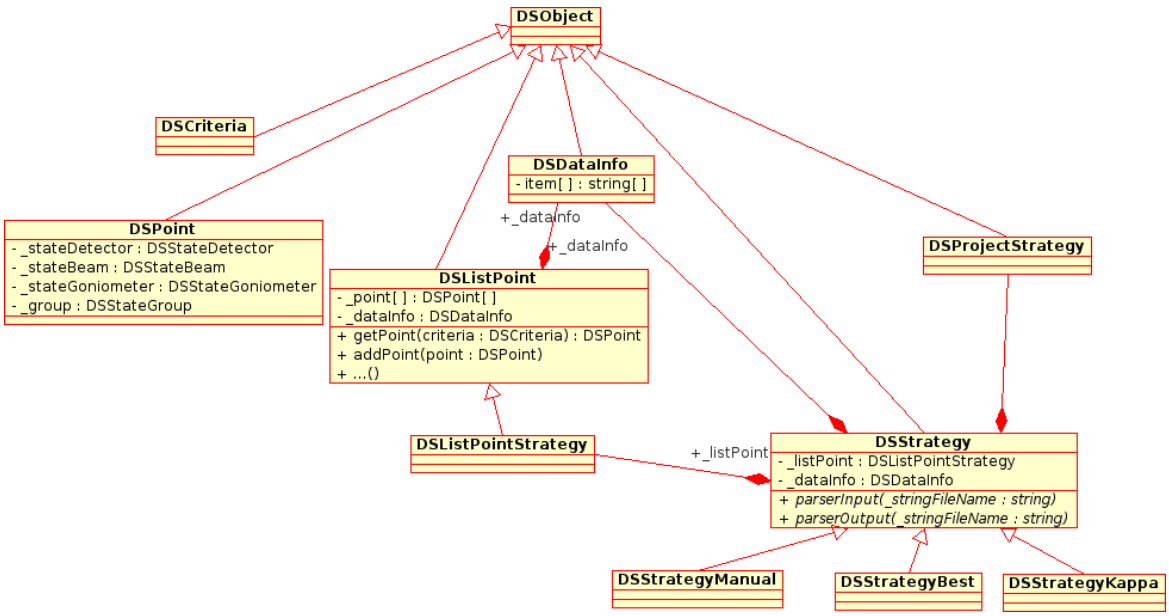
*Note: the class namespace used in this proposal is DS to represent “DNA Spike” followed by the name of the basic class and/or the other child classes.

Basic Design Pattern: Object Factory Pattern

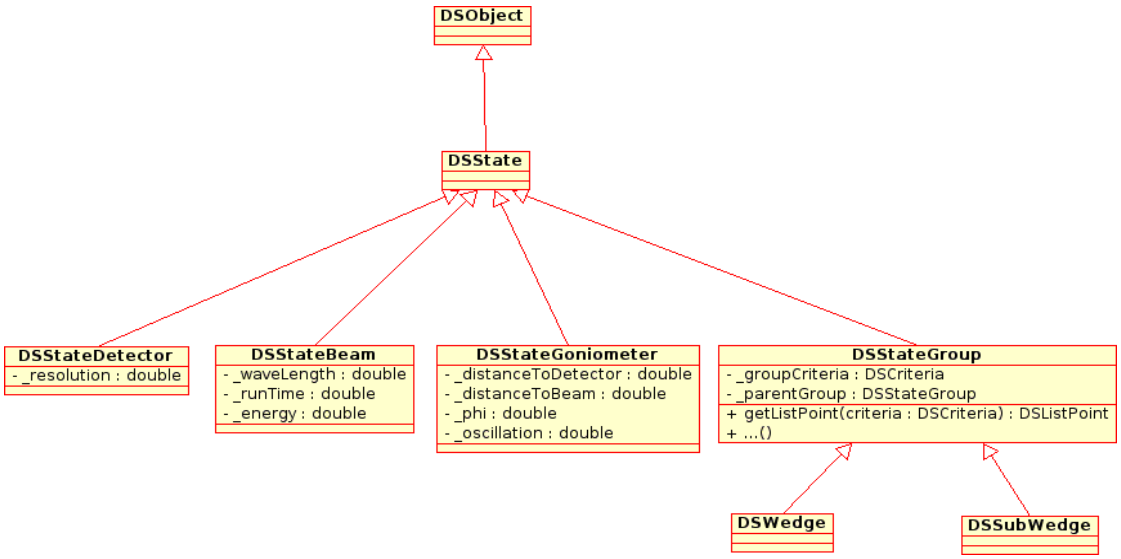
- The main classes DSProjectExperiment, DSStrategy (and its childs) and DSStrategyPoint are generic containers of objects. This flexibility comes from the pure object architecture where all objects comes from an unique basic entity, in this case called the basic “DSObject”. It allows the flexibility for evolution and a single code engine to manipulate any kind of object in the current and future new type of elements or entities.

Class Diagrams

The uml definitions were designed using the open source tool “Umbrello”, available in al modern Linux distributions. These uml specifications are using the standard “xmi” file definition to be shared to all standard case tools. They can be used to generate interfaces using any kind of code language. The diagrams shown here are incomplete and they represent only a draft to share the main concepts and they will be improved.



This uml definition shows the hierarchy of different containers used in the DSProjectStrategy, DSStrategy (and its specialisations) and DSListPointStrategy. The containers are using the “Object Factory Pattern” to implement their manager system of class members. The object relation and the main mechanism of these different entities are discussed briefly in the section “Overall Concept”.



This diagram shows some different status objects to store the information to define the “Strategy Point”. These objects are stored in the DSStrategyPoint container. These States objects represent the information to generate the experimental condition to acquire one data. These state class types must be expanded to address new equipment and new scientific cases.

External Code Dependencies (libraries, framework,): No Dependencies

- The data model doesn’t use any kind of library and it can be implemented in all languages using standard types and convection;
- Easily exported to any kind of external application and framework;

Scientific Cases Covered by this data model

- Single-Wedges;
- Multi-Wedges;
- Sub-Wedges;
- No-Wedges
- Still Images Experiment;
- Variable Oscillation Methods ;
- Kappa Experiment (using a specialized DSStateGoniometerKappa);
- Mad Experiment;
- Inverse Beam Experiment;
- Manual Experiment;
- Automated Experiments (i.e.: robotic arms and devices);
- Pipeline Experiments;
- Incomplete Experiments (i.e.: recovery or recycling of previous experiment);

Hypothetical Future Scientific Cases covered by this data model (with no modification)

- Multi-Beamline Experiment;
- Multi-Detector Experiment;

Conclusions

The model “Experiment-StrategyPoints” is a first draft and it must be completed and improved. The basic approach of Strategypoint presents a tremendous potentiality to be applied over a large number of scientific cases with no changes in the main architecture and code. The code back-compatibility is assured with the possibility to add new strategies, devices and scientific methodologies. The model has no dependencies on other external tools or libraries and licences.

Feedback

This proposal contains no specifics of how it would be used in practice, in particular about the mechanism for implementing the model, how persistence and databases would be handled or what functionality would be provided by the generated interfaces.

The point about sharing models between tools using XMI needs to be treated with caution. There are several different versions of the XMI standard, and vendor interpretations can also vary. Support for preservation of diagram layouts is also poor. The ease and success of this process may depend on the particular tools involved.

Appendix 5: CCPN-generated API

Key features

The Python API included in the spike prototype has been generated automatically from the BioXDM-style model discussed above. The result of the generation is a single file of Python source code for each package in the model, containing one class definition for each class in the model. For the status of API documentation and XML schema information, see below.

Each classifier (class or complex type) in the UML model is implemented by a Python class definition in the API (see wiki:SpikeReport_Prototype_Data_Models_BioXDM_Style_Presentation for a discussion of complex types). The automatically-generated code includes:

1. Getter and setter methods for all attributes and associations (which is why these methods do not feature in the model)
2. Validation code to check constraints on classifiers, attributes and associations. This code is invoked at construction time (classifiers) or setting time (attributes and associations). It can also be invoked manually via a checkValid method on each class
3. "Finder" methods for all associations. When invoked on an object at one end of a link, these methods will return a list of references to objects at the other end of the link satisfying a set of attribute equality conditions. For example:

   myStrategyCharacteristics = myStrategy.findAllStrategyCharacteristics(user="pkeller")

   The cardinality of the association involved here is 0..1 so this is not a good example but hopefully conveys the idea. These finder methods have not been used in the prototype.
4. Methods to construct new instances of associated classes.

A small amount of hand-written code exists in the API to enforce manually-specified constraints and provide specialised methods: this code is maintained inside ObjectDomain, and so is included in the API automatically at code generation time without any further action needing to be taken. The proportion of manually-written code in the prototype's model is probably less than 0.5%.

Current state of the CCPN machinery

The CCPN generation machinery exists in two versions. The new version has more powerful modelling features than the old version (some of which have been added at the request of Gerard Bricogne and Peter Keller). At the time of writing, the porting of functionality from the old version to the new version is work in progress. At the time of the spike, the new version does not yet generate HTML documentation for the API, nor XML schema for the data being persisted. The development of the Java API generation also lags behind that for the Python API.

Persistence

Persistence in the CCPN machinery is handled through "Repositories": each package has its own set of repositories, each of which can have one of three roles:

1. working project data

2. backup
3. data not specific to any project (CCPN have traditionally used
4. this repository for things like standard amino acid residue definitions).

With the XML persistence used in the spike, each repository has a default location on the filesystem. This location can be changed. The mechanism is slightly awkward, although an improvement is being worked on by CCPN: the prototype's code to do this in the _setPersistedDataLocation() method of DSPluginLoadBestOutputAM.py

The persistence is tied to the root object of a working system (an instance of memops.Implementation.MemopsRoot). Every instance of a normal class that has not been deleted is persisted by a single call to the saveModified() method of the root object. (Instances of complex data types behave slightly differently: see below.) In normal use, persistence is handled automatically by the framework, and the developer does not need to become involved in the details.

Managing objects

Creating objects

Objects are created in one of two ways:

1. Instances of normal classes are instantiated as children of their parents. In the prototype, this is done with code such as:

   myChildObj = myParentObj.newChildObj(attr=value, ...)

   In the Python API, attribute names in the constructor have to be one of the attribute names of the class defined in the model (an exception is raised otherwise). As an alternative to a list of name=value pairs, a Python dictionary can be used instead.

2. Instances of complex data types are instantiated as disconnected objects initially. The resulting object can then be used to set an attribute of another object, either in a constructor or an attribute setter. Constructor usage looks like this:

   myComplexObj = ComplexObj(attr=value, ...)
   myChildObj = myParentObj.newChildObj(complexAttr=myComplexObj,...)

   If the complex object is instantiated but not used to set the attribute of another object, it will not be persisted.

Accessing attributes and links

Objects are modified through setter routines that are autogenerated for each attribute or association by the CCPN machinery, for example:

```
myChildObj.setAttr(value)
```

Python also allows direct setting of attributes, such as:

```
myChildObj.attr = value
```

however this form has been avoided in the prototype, since mistyping the attribute name results in a new attribute of that name being created on the object. If this happens, the attribute will not be persisted: the datum being set is volatile and will not be recoverable after the application terminates. (This is a Python feature, not specific to CCPN).

Similar to the setter methods, getter methods are auto generated for each attribute.

Note that attributes and links both have getter and setter methods, so they can be used to navigate between objects as well as accessing attributes. Associations are given default names based on the class name, but these can be overridden in the model.

By default, attributes with cardinalities greater than one are handled as lists, and links with cardinalities greater than one handled as unique sets (although this behaviour can be altered on a case-by-case basis in the model).

Deleting objects

Objects are deleted by a call to the delete() method defined on each object

CCPN Feedbacks

Pros

1. The idea of having a generated API from a Data Model into different languages and persistence formats is good (but not mandatory)

Cons

1. Code generation machinery
    - This is more a comment than a feedback about Code generation machineries : Normally, generated code doesn't have to be manually updated since in case of Data Model modifications, the code should be re-generated from scratch. Therefore, the generated classes should be sub-classed in order to allow manual modifications (methods additions: that are not implemented by the machinery, potential workarounds on generated code bugs.
2. Verbose code
    - Strategy Data Model code is generated in a single python file (~9 classes ~6000 lines)

      : constructors, getter, setter, type checking, navigation.

    - Many code duplications around Type checking and navigation.
    - 2 APIs have been generated (memops.api.Implementation and dnaspike.api.Strategy). These 2 apis are closely linked. AFAIU (As Far As I Understood), basically memops.api.Implementation deals with ccpn main Types the dnaspike.api.Strategy DM objects are an extension from (Dependence from dnaspike.api.Strategy to memops.api.Implementation) and defines the entry point in the Data Model with the link MemopsRoot/StrategyGroup (Dependence from memops.api.Implementation to dnaspike.api.Strategy).
    - some unecessary explicit re-definition of inherited methods and attributes:

      ex: whenGenerated = StrategyCharacterics.whenGenerated

- For the reasons listed above, the code is not very easy to read.

3. Flexibility lack
    - Close dependence on ObjectDomain tool which status is uncertain.
    - Object creation: CCPN objects(*) have to be created top-down from a ccpn root object:

      This is connected with the way objects are persisted and loaded. Ccpn requires a primary data hierarchy to be explicitly defined in the model (Tree hierarchy). This prevents different instances of a particular class belonging to instances of different classes

```
sg = root.newStrategyGroup()
s1 = sg.newStrategy()
w = s1.newWedge()
sw1 = w.newSubWedge(rotationAxisStart=0.0, imageOscillationWidth=1.0,
       imageExposureTime=2.0, numberImages=10)
```

    - For the reason listed above, the ccpn objects can't be Unit tested independently. This implies potential maintenance issues.
4. Maturity lack
    - at least for python, persistence feature was not fully implemented
5. No Unit Tested
6. License of CCPN not clear – is it finalised? Question related to the following URL: http://www.ccpn.ac.uk/ccpn/software/ccpn-license. Does this relate to anything related to DNA 2.0 developments?

(*): "CCPN objects" should be understood as "Objects generated by the CCPN machinery"

First, I have to state that we could not test the CCPN machinery right from the beginning of the spike as a product, because it was not fully functioning. In contrast, we have profited from its ongoing development by being able to finally achieve the goals of the spike, namely to produce a working application using this tool.

We have applied this tool to automatically generate a Python API which could be used for a persistence layer of the spike application. It has been applied on the "BioXDM-style Data Model" resulted by a common effort of all of us. Although when working with this data model we are continuously developing a data model for mx experiments, within the spike we have not applied CCPN to any other data model. (Not even to the "Experimental Data Model" designed and presented during the final phase of working on the spike.)

Regarding the CCPN tool:

PROS

- the idea of automatic generation of APIs in multiple languages (ideally: Python/C/C++/Java)
- free availability: we can collaborate to develop it further on our taste

CONS

- slow development of the desired features as the result of a lack of manpower

- not yet delivered features
    - API in C/C++/Java/Fortran
    - concurrent access control
- inflexibility in constructing objects on-the-fly (only just a few construction mechanisms are supported)
- additional work can be foreseen as the base tool (ObjectDomain) must be replaced

---

A factual point: manually-written methods can be added to existing classes, and there is no need to subclass existing classes to do this. This has been done during the spike, and the API's were regenerated repeatedly with no problems at all (OK - given the ObjectDomain situation you will have to take my [Peter's] word for this, but that is what happened). A manual method can also be defined with the same name as one that would have been automatically-generated, in which case no automatically-generated method is produced: the manual definition is substituted in its place.

CCPN licencing issues: the only CCPN code that is under the GPL is that used for implementation generation. We would not need to distribute that. [If DNA users wanted to do generation themselves, they could acquire the code directly from CCPN - there would be distribution/version issues, but nothing that we couldn't handle IMHO.] The part of CCPN's own model (the memops package) that we used, plus its generated implementation are under the LGPL (which allows redistribution of original and modified versions of the LGPL-ed code, irrespective of the licence that applies to other parts of any application that links to it). Our model plus generated implementations would be under whatever licence we choose to give it.

The point about unit testing is not specific to CCPN, but general to any situation where classes derived from a data model are instantiated. The UML (in common with other paradigms such as Entity-Relationship Modelling) allows the definition of an association between two classes that is mandatory at either or both ends (i.e. has a low cardinality greater than 0, such as 1..* or 1..3). If an object is instantiated without another object at the other end of a mandatory link, the first object is in a state that violates constraints derived from the model. The issue is: under what circumstances should it be possible to instantiate a class in this way, and what operations should it support?

Appendix 6 : AALib – Asynchronous Action Plugin Framework http://aalib.sourceforge.net

Introduction:

Main goal: « Code once and run anywhere ».

The project is a Framework* based in Pure Object Oriented Concepts to develop asynchronous communication between threads (thread safety) and external process, allowing the development of complete applications in different programming languages and platforms for heterogeneous developers. The « Basic Application Framework skeleton » implements the RAID - “Rapid Application Development and Deployment” - to manage asynchronous code.

The project started with the main objective of generating a unique code framework running on all platforms (windows, linux, etc) to control diferent motors and devices asynchronously. It was improved during three years of continuing development and now other domains can be addressed such as software control and grid computing.

Objectives of the Framework:

- Independent of the System architecture and OS;
- Independent of the Programming Language;
- Independent of the Programming IDE;
- Full OOP design – Object Oriented Programming;
- One code running on all platforms with the same features and actions;
- Asynchronous Multi-thread Actions + Thread Safety + Grid Computing;
- Advanced Resources and Generic Application Design (i.e.: RAID);
- Implements the standard internet services and protocols;
- Use only Objects with simple and standard definitions;
- Each Object is independent and self-contained (no hierarchy to generate and manage entities);
- The data persistence is assured by each individual object (duplication or other operation);
- Complete and independent Unit test framework;
- Multi-thread exception handling and log;
- Automatic generation of code documentation in html, pdf, and man;
- Incorporate Debug and Exception control Model;
- Well defined code syntax and rules for the code design;
- Simple code maintenance (all code is identical in all operational system);
- Code reliability (i.e.: runtime flow control robustness);
- No necessary external licenses or agreements;
- Massively tested and in continuous support and improvement;
- Full compatible with all C, C++ and Python standard libraries and frameworks;
- Extensibility tested with some GUI frameworks (i.e.: QT3 and QT4);
- It can be deployed in/with other code languages as Java and C++;

Basic Design Patterns Included in the Framework:

- MVC Design Pattern = Model-View-Controller;

- Object Factory Design Pattern = Plugin Model;
- Callback Design Pattern = Generic code connection and execution;

External Code Dependencies (libraries, framework):

- Standard official Python version: 2.4 or later;
- Tested and used on: Linux, WindowsXP, Solaris, Wincygwin and MacOS X.

User Cases Covered by this Framework:

- Basic Application Framework: A basic application class defines a "skeleton" to RAID ("Rapid Application Development and Deployment"). It manages all logs, resources and actions creating and connecting all code with the heterogeneous operational system. All signals and events are controlled and monitored by the main code inspector. The main architecture to generate and control the multi-thread actions is encapsulated on the framework and its basic classes to simplify the code development;
- Plugin Framework: The plugin factory pattern is extensively used on the basic application skeleton to implement the code generation and object management during the runtime. It creates and manages the mechanism to load objects and data dynamically;
- XML-RPC server plugin architecture: An extension of the Basic Application skeleton implements a complete internet server/client architecture based on the XML-RPC standard ("Remote Procedure Call based on XML file exchange"). It is created and configured by the multi-thread plugin factory implemented by the library. A special mechanism is used to exchange the runtime objects natively and transparently to the user. It is a high level interface to manage the complexity of the object translation to XML definition and its managing;
- Basic Callback System: It allows the connection of any code from any library and framework in synchronous and asynchronous mode. The callback design pattern is implemented in the kernel of the main classes of the framework.
- Data Binding – XML Scheme object construction: The data binding standard based on XML scheme is natively implemented in the framework. It allows the generation and compilation of native code on runtime. It is used on the plugin framework to generate the in/out classes to manage any kind of data from/to xml files;
- Subprocess Management: The framework implements plugin and specialized code to manage external process based on multi-thread actions.

User Case under development:

- Grid agent architecture to run any object in any computer in the network using the XML-RPC AAlib object exchange model;

Projects Using the AALib framework:

- DNABenchmark – Test benchmark manager to evaluate the scientific results – The DNA - http://www.dna.ac.uk - project is a collaboration initially between the ESRF, the CCLRC Daresbury Laboratory and MRC-LMB in Cambridge, with the aim of completely

automating the collection and processing of X-Ray protein crystallography data - Founded by European Community Project: BioXHIT http://www.bioxhit.org

- DRank : Crystal Data Ranking Module in DNA Package – The DNA project - http://www.dna.ac.uk : BioXHIT http://www.bioxhit.org
- Beamfocus: Beamline Assisted Focus Application – Software used to assist the beamline operator to optimally focus X-Rays beams - ESRF European Synchrotron Radiation Facility – Scisoft Group - http://www.esrf.fr/SciSoft
- EDFExplorer – Software used to assist the beamline user to transfer his data in EDF format to an HDF5 container - ESRF European Synchrotron Radiation Facility – Scisoft Group - http://www.esrf.fr/SciSoft

License:

- Based on FreeBSD open source license: free and open source;

Future objectives:

- Generate an extended documentation;
- Create new examples;
- Create tutorials for training;

Conclusions:

The AALib framework is a well tested open source framework to be used on RAID development of asynchronous multi-thread applications. It implements the basic plugin architecture to be used with the special AALib action objects and standard python classes. This mechanism allows the rapid development of dynamic and modular applications to facilitate continuous improvement. Modern software engineering techniques based on unit tests are used to guarantee the quality of each part of the code during the development cycle. The python library version of the AALib is fully compatible with all standard frameworks using C, C++ and Python.

Feedback

Pros

- The code is quite clear and readable (especially with the Unit Tests framework that allows not to code too much type checking in the source)
- Unit Tests
- It is quite easy to create an (simple) application skeleton with AALib

Cons

- Lack of documentation
- Difficulty to enter easily in the framework. e.g; I discovered quite late that a plugin was not only an Action Class but can also be any kind of object that has an xsd or a py file available.
- Interaction between plugins is not very clear (IN/OUT parameters)

Appendix 7 : Unit test and Validation Test

Introduction:

In computer programming, unit testing* is a procedure used to validate that individual units of source code are working properly. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual program, function, procedure etc, while in object-oriented programming, the smallest unit is always a Class; which may be a base/super class, abstract class or derived/child class. Units are distinguished from modules in that modules are typically made up of units.

Ideally, each test case is independent from the others; mock objects and test harnesses can be used to assist testing a module in isolation. Unit testing is typically done by the developers and not by end-users.

The validation test is the test set based on confirmed results or benchmark files. It is used to confirm and compare the final result of the application and also define the domain where the software should be applied.

*Note: – see complete definition at http://en.wikipedia.org/wiki/Unit_test

Objectives of the Unit test and validation:

- Facilitates change - re-factor;
- Simplifies integration;
- Documentation;
- Separation of interface from implementation;
- Comparison;
- Benchmark;

Test procedure used on the spike:

- Some unit test were developed but not for all code;
- No validation test developed due a lack of time;

Conclusions:

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it helps to maintain control of the quality of the code during the software life cycle. The validation test guarantees the final result and defines the domain of the software and it should be used to benchmark the overall results of the application.

Appendix 8 : Spike management tools - Trac

Trac is an open source, minimalist, web-based project management and bug-tracking tool, inspired by CVSTrac. It is developed and maintained by Edgewall Software. See the Trac Wikipedia article for more information.

The version of Trac used in the spike is 0.10.3. The spike Trac server is managed by Diamond and in order to use it one must first have a "federal id" which are not discussed here (see Diamond hosting.)

Positive experiences

- Trac simple and therefore easy to have an overview of all functions.
- Trac is an open source project so it's possible to change it's behaviour if needed. For example, during the spike a translator between federal ids and real names was introduced.
- The connection Trac - subversion ("Browse Source") worked well.
- Trac worked well in general, we didn't spot any bugs during the spike developments.

Negative experiences

- The wiki part is maybe too simple to be used for serious documentation. For example:
  - No support for navigation bars.
- The idea behind the "Roadmap" is a good one but might again be too simple - if there are too many mile-stones it's very difficult to have an overview.
- The "tickets" are also too simplistic:
  - It's not possible to assign a ticket to a subgroup of people.
  - Email notification which didn't work in the beginning cannot be assigned to one or a few persons, instead the notifications were sent to the spike mailing list.

Feedback

Another negative: some other wiki's have the concept of a page being a child of another page, thus providing a hierarchy. This isn't always useful, but here it would have been. We have to remember to put a link at the bottom of every page of the report that says "Back to the page of contents".

Appendix 9 : Subversion

The Subversion revision control system was used throughout the spike. Peter used it mainly through the Subversive plugin for Eclipse, but also explored briefly the following tools:

1. Subclipse (another Eclipse plugin)
2. SmartSVN (Foundation version)
3. kdesvn The KDE subversion client
4. The 'svn' command

Subversion vs. CVS

Subversion has several advantages over CVS, some of the significant ones being:

1. Operations are atomic.

   An operation such as commits of multiple files either wholly succeeds or wholly fails. It will not partially succeed

2. Natively client/server.

   Working directories can be accessed by any (well-written) client, and the developer can change clients without having to check out a new set of working directories. (With CVS, the exact format of some crucial files is not properly documented, so checking out using one client and accessing the working files with another risks data corruption. CVS only supports the use of the 'cvs' command.

3. Can export from a working directory

   `svn export /path/to/working/directory` will create the files that you would get by committing local changes to the repository from the working directory, and then exporting from the repository, but it does this as a purely local operation. This allows a check that the commits that you are about to do are complete.

For other features, see the Subversion home page.

Subversion does not require or support the concept of CVS-like tags and branches: this is because commits are considered to create a new version of the entire repository. The functionality of tags and branches is implemented by creating new directories in the repository, and by convention top-level directories called `trunk`, `tags` and `branches` are used to do this. Depending on need, these can be at the top level of the repository or at the top level of each project within the repository. A decision about this has to be taken early on in the lifetime of the repository. See chapter 4 of Version Control with Subversion for a full discussion of this subject.

Some Subversion clients use this convention to provide CVS-like tag/branch functionality (for example the professional version of SmartSVN and Subclipse]. Tagging and branching with Subversion was not done during the spike.

Choice of client

To some extent this is arbitrary and can be left to individual developer, however for the spike we used Eclipse so it was a choice between the Subversive and Subclipse plugins. Subversive was

better at handling multiple repositories, and allowed a username to be associated with each repository. (Also a password, but it was not obligatory to do this.) For checking commits, the `svn export` command (as described above) was used.

Neither plugin seemed to offer a direct equivalent of `cvs delete`, namely the ability to mark a file for deletion, but only delete the working copy on commit. Both hooked in to the normal file delete operation of Eclipse, but then this actually deleted the working file.

Note that in its default configuration the `svn` command stores the repository password in plaintext in a file in the users' home directory tree ("see http://subversion.tigris.org/faq.html#plaintext-passwords") for more details of this -- the configuration file has to be edited to prevent this from happening. Using publickey authentication with an agent would be a better solution, but this was not tried.

Appendix 10 : Diamond hosting

Positive experiences

o Ready-to-use solution for the spike

Negative experiences

o To perform administration tasks on the DNA project Management tools (Trac and Subversion servers) one has to ask the Diamond sys admin via Karl. A limitation is that in general, a Sys Admin is rarely available, that is especially the case for Diamond because the site is “under construction”. This is a drawback towards public SourceForge-like solutions where administrative tasks can be easily performed by somebody who has the administration rights on the project.

o For security reason, users must have Federal Ids for identification and to use Diamond Trac Server. A Federal Id request must be done directly to Diamond. A drawback is that users ids are set by default to the FED ID (i.e zjd18495) and thus one can't be recognized easily, a mapping must be done to switch specifically a FED Id to a user name. A second drawback is that an external user can't access the Trac server even in a read-only mode unless he has his FED Id set.

o If the Diamond sys admin decides your password isn't up to scratch, your account gets disabled with no notification (other than an e-mail to your Diamond account which you can't read because it's been disabled...). If your account is disabled, you lose access to the SVN etc!

Appendix 11 : Eclipse

Eclipse is an Integrated Development Environment (IDE), originally developed for use with Java programming, but it has been extended with the availability of "plug-ins" for various other languages. Of particular interest to the DNA Spike is the possibility of using it with Python, for which the most suitable plug-in seems to be PyDev.

Wikipedia says the following about IDEs:

"An integrated development environment (IDE), also known as integrated design environment and integrated debugging environment, is a type of computer software that assists computer programmers in developing software.

"IDEs normally consist of a source code editor, a compiler and/or interpreter, build-automation tools, and (usually) a debugger. Sometimes a version control system and various tools to simplify the construction of a GUI are integrated as well. Many modern IDEs also integrate a class browser, an object inspector and a class hierarchy diagram, for use with object oriented software development. "

In other words, an IDE replaces tools that a traditional programmer might use such as editors, command-line compiler, Makefiles, etc. and should give ready access to other tools in order to make development more straightforward.

For a traditional programmer, using an IDE is a real pain in the neck to begin with - rather than starting your favourite editor, typing the code in (formatting it pretty much how you want), then running either invoking an interpreter or running a compiler and the resulting executable, you need to do a lot of book-keeping stuff to even start programming. However, it seems that once it's done, it's done, and you get a lot of tools to help develop and debug your code, and it should lead to more orderly development. It should also help keep developers in register with each other, since the version tracking is much better integrated.

Eclipse plugins

The following plugins were used:

- Subversive: Plugin for Subversion.
- Subclipse: Plugin for Subversion.
- PyDev: Plugin for Python and Jython.

Subversive and Subclipse are "competing" plugins, i.e. they both serve the same purpose: To provide an Subversion interface for Eclipse. Therefore it is only necessary to install one of them. ! Eclipse is distributed with CVS pre-installed, so it is not strictly necessary to install Subversion for development tracking, but since it offers some advantages it seems prudent to do so.

Similarly, there are different plug-ins available for Python.

Installing Eclipse

I've only done this on a couple of Macs, but if other platforms are as easy then it really shouldn't be a problem for anyone.

Eclipse can be downloaded from http://www.eclipse.org/downloads/ as tar.gz or zip files for a

variety of platforms - the download server recognizes the platform running the web browser and chooses the appropriate version.

Since Eclipse was developed primarily as a Java IDE, a Python plugin was required for the Spike. A number are available, PyDev was chosen. Other languages can also be supported via plug-ins, e.g. Fortran (but this is irrelevant to the Spike).

The URL for PyDev can be found on the Eclipse web site - from http://www.eclipseplugincentral.com there's a link to http://pydev.sourceforge.net, where instructions (including the download URL for use in the Eclipse software update manager) for installing PyDev can be found.

The installation itself of the various plug-ins is dead easy, and takes about 10 minutes from start to finish if you have a half decent network connection, even without referring to the documentation. The only problem I found was that the "options" should be de-selected from the expanded list of installation options so that Mylar (whose function is unclear) is not searched for.

Before creating a new Python project in Eclipse, you need to specify a Python interpreter or Jython compiler (Jython compiles Python scripts to Java bytecode).

Subversion

Advantages and disadvantages: Overall, both Subversion plugins seem to work quite well. While Subclipse is better for importing a checked-out project, Subversive is aware of the trunk/branches/tags directory convention (with configurable names for these directories). Subclipse is being developed by the same Open Source community (Tigris.org) as Subversion, which might be seen as an advantage.

Neither Subversive nor Subclipse seem to support "SVN delete" functionality without immediately deleting the local file / directory as well.

Appendix 12 : Marratech

Marratech has been heavily used during the spike as the main communication platform (next to wiki, mail-list, telephone, real meetings etc.). Although, as summarized in the PROS section, the platform really gave a good performance and got a nice acceptance, we have also experienced that it cannot replace the REAL Meetings with face-to-face personal contacts. Since we had Marratech Video Conferences (VC) quite frequently, we had to cut the meetings regularly to keep them (relatively) short. That led unavoidably to not clarified thoughts hanging in the air, miss-understandings appearing on the different sides etc. Although essential issues have been proposed for almost all meetings, a lot of us could not attend all of those. Probably a better differentiation between essential topics/decisions, and irrelevant technical questions could help the people to organize their attendance better. It became also clear, that participating such regular VCs does not mean that a REALLY presence because of interrupting telephone calls, colleagues looking for immediate help assistance, etc. Indeed, it is quite natural since one cannot be separated from the everyday working environment so often just because of a moderate priority collaborative project. Last but not least, although the headset is on (and the webcam as well), if the VC leads to a less relevant field, one can easily find himself working on a different topic on his computer.

Evaluation of the VIDEO CONFERENCING tool: In the beginning of the spike, we have tried to test an alternative tool (EVO), but we could not manage to start it up because of firewall restrictions. Lacking the time for a systematic evaluation of all available solutions, we simply jumped on the Marratech service. A bit more detailed review on potential alternatives is available at eg: http://decibel.fi.muni.cz/download/papers/Report_on_Videocoferencing_Systems.pdf Note that this review suggest the free, open source and cross-platform system of the UCL multimedia package (VIC/RAT/WBD/NTE/SDR): http://www-mice.cs.ucl.ac.uk/multimedia/software/

Pros:

- already available service at ILL/EMBL/ESRF site. There are no special installation and maintenance issues.
- cross-platform solution running on JVM supporting all different kind of computers (problems are listed among contras).
- It was working fine on quite versatile set of our computation resources (WinXP PC with a Creative webcam, Mac station, etc.)
- easy invitation to ad-hoc conferences (we have not really used this feature)
- graphical whiteboard with Cut&Paste (and desktop sharing) capability
- leading discussion on the whiteboards
- integrated chat feature
- during the VC, private chatting and talking possibility with other participants
- acceptable sound quality and video refresh rate (also for non-speaking participants)
- recording the flow of the VC (very nice! a shame that there is no CLEAR ENOUGH feedback that the recording is ON or NOT)

Contras:

- limitation of available seats (ILL offers 20 place at the moment. We could easily overrun the limit of 10 that wa active during the spike.
- since it is not a specific solution for a specific computer, but rather a cross-platform solution,

it does not deal with all the different microphones and camera drivers on different operating systems. It assumes a standard webcam based setup being supported by the JVM used. In cases where the actual JVM does not support the available hardware/OS/driver combination, it fails correctly functioning. During the spike we have experienced a lot of problems because of this: system connection lost / webcams did not work / multimedia drivers became unusable, miss-configured / etc. It is a shame that we do not have a systematic study on the correctly working combinations of different hardware/OS/driver/JVM.

- Note that the current version of Marratech does not work with JVM1.6 - as we have experienced
- OpenSUSE 10.2 and the Philips SPC900NC webcam, the Marratech client made the user's workstation unstable.
- Slow video refresh rate on the small video boxes
- No way of resizing the small video boxes, and open more big windows, etc.
- No way of selecting a given participant to be monitored in the big window (after the first voice the currently speaking participant is monitored automatically).
- no automatic VC wide sound level control which would adjust the site specific microphone settings

Feedback:

The ESRF Computing Services (CS) has started to look into different hosted VC solutions that could replace Marratech. Up to now we have not found any other which could provide any more interesting features. We have assisted in the testing of three different alternatives:

Meeting3D by Texio

This VC solution has a 3D function which resembles Second Life but which seems of no interest for our type of collaborations.

Pros:

- Good audio quality
- Excellent video quality
- The video window can be detached and be "floating"
- Excellent white board
- Excellent sharing of application - very rapid
- Applications can be remotely controlled. E.g. if one participants starts sharing of eclipse, he/she can then give any other participant the right to control eclipse remotely.

Cons:

- Runs only on Windows
- The speaker video is not automatically highlighted
- Local ESRF problem: the participants are systematically disconnected after a few minutes

Adobe Acrobat Connect Professional

Pros:

- Good application sharing

Cons:

- Very bad audio quality
- Only four videos visible at one time
- Windows only *(Comment by Karl: I think participants (not presenters) can use Linux ...)*

Webex

Pros:

- Good application sharing

Cons:

- Very bad audio quality
- Only four videos visible at one time

EVO

Pros:

- Free

Cons:

- It is a shame that we could not test the free EVO system.

Appendix 13 : New name

When the DNA collaboration was started in 2001, the acronym “DNA” was inspired by the “GNU” so called recursive acronym: “GNU is Not Unix”. As we were aiming for automation of structure determination and there was already a project called “Autostruct” we took (unfortunately) the decision to call the project DNA: “DNA is Not Autostruct”.

After having too many times tried to explain this acronym to people not knowing what “GNU” means (and there are many people who doesn't know...) we finally gave up. We couldn't however at this point change the name of the project so given the acronym we started to look for the words which could form this acronym. In 2004 an alternative was finally found: automateD collectioN of datA.

This acronym could finally be explained to everyone however yet a big problem remained: a lot of MX science is about the biological DNA which leads to confusion. Therefore, since we are now starting a new collaboration we should find a new name for the project. This new name should ideally:

- Contain at least some the key words for what we are trying to do: automation, strategy, ranking, (data) collection, etc.
- Be as unique as possible so that anyone can easily find it by doing a “google” search.

An example of such an acronym is “ISPyB” : “Information System for Protein CrystallographY Beamlines”

Here's the list of candidates:

- **STAR** : (automatic **ST**rategy **A**nd **R**anking). Not really a good one because “star” is a very generic word.
- **ASYD** (pronounced "acid") : **A**utomatic **S**trateg**Y** and **D**ata collection
- **RASTAN**: **RA**nking, **ST**rategy & dat**A** collectio**N** (see Rastan Saga)
- **McATRISC / Mc@Risc**: **M**acromolecular **C**rystallography **A**utomation **T**ool for **R**anking, **I**ndexing, **S**trategy calculation **A**nd (data) **C**ollection
- **STAR'AC** : **ST**rategy **A**nd **R**anking for dat**A** **C**ollection (see STARACademy)
- **RASTER** : **R**anking and **A**utomatic **ST**rategy gen**ER**ator
- **RASTAMAN** : **R**anking and **A**utomatic **ST**rategy gener**A**tor (**MAN**ual over-ride possible)
- **EFFECT** : **E**nvironment **F**or **F**acilitating mx **E**xperiments
- **Astradapt**: **A**utomated **ST**rategy, **R**anking **A**nd **DA**ta **P**rocessing **T**ool
- **mxSTRIDE** : mx **ST**rategy, **R**anking and **I**ndexing for **D**ata collection **E**xperiments. Or alternatively, **mxASTRIDE** with **A**=automatic.
- **GYGAND** : strate**GY** and rankin**G** for **A**utomated collectio**N** of **D**ata
- **ADE** : **A**utomation of the **D**iffraction **E**xperiment
- **SPADE** : **S**oftware **P**ackage for **A**utomation of the **D**iffraction **E**xperiment
- **SADE** : **S**oftware for **A**utomation of the **D**iffraction **E**xperiment
- **SADDCAP** : **S**oftware for **A**utomation of **D**iffraction **D**ata **C**ollection **A**nd **P**rocessing
- **ASTRACOP** : **A**utomatic **ST**rategy **R**anking **CO**llection and **P**rocessing
- **AutoXDC : Auto**mated **X**-ray **D**ata **C**ollection
- **SAGE** : Not (yet) an acronym, refers to JD Bernal

Appendix 14 : Impact on DNA 1.X maintenance / development

Progress on the developments for DNA 1.1 has become increasingly slow with the intense activities of DNA 2.0 developments, especially the tight time scales for the SPIKE, high frequency of meetings, other distractions of developers not involved in the SPIKE being the major factor along the overall improvement to DNA. This raises two major problems:

1) completion of DNA 1.1
2) any plans for future DNA versions under the current framework. There are many options that can be taken for the future of DNA in the current framework:

For DNA 1.1

Option 1: discontinue developments.
Option 2: release DNA in its current state and continue with small incremental improvements.
Option 3: complete DNA 1.1. This would require a concerted effort from approximately 7 developers, Olof, Romeu, Graeme, Karl, Harry, Andrew and Gleb.

For DNA 1.2

What is required for DNA 1.2 has never been fully specified and on timescales of developing DNA in the current framework, embarking on this body of work may have limited merit.

(PS this is a draft!)

Feedback

I think it was always clear that we cannot go for a newer version of DNA without delivering a **stable working version**, namely **DNA 1.1**. It must be released for the MX community which is really waiting for it. But I agree that we should draw the line (as it has already been drawn several times) and stop overadorning.

As also agreed, we should then also concentrate on disseminating the technology developed over the past years, and write up the paper about v1.1. Note that I am still happy to contribute with the Kappa Section before heading a long work towards a newer and more **EFFECT**ive DNA.

Appendix 15 : Java versus Python

Python	Java
Source File: It is possible to define a method in a file that isn't linked to a class. A method is itself considered as an object.	**Source File:** The files are strongly structured: 1 class per file. A method is linked to a class.
Object: Everything is Object even a method. Advantage: a method can be considered as a parameter of another method (callback, see section “callback”). Drawback: It is also possible to point a method on another method to make some kind of short cuts. Ex: theMethod = Another Object.theMethod	**Object:** Something that has attributes and methods. See "Source". Notion of final class/method/variable in Java: - a final class can't be sub-classed, ex: String - a final method can't be redefined by a sub-class - a final variable can't be modified (constant)
Concise style: . Ex1: Hello World Program print "Hello, world!" This is a script-oriented programming language. i.e creating a class is not mandatory to print “Hello, world!” on the screen. But this piece of code can be encapsulated in a class that has a main too. . Ex2: Condition * annexe2 The indentation indicates the beginning and the end of the condition . Ex3: opening a file myFile = open(argFilename)	**Verbose Style:** . Ex1: Hello World Program * annexe1 One needs to create a class for that. . Ex2: Condition * annexe3 The “{“ indicates the beginning and the end of the condition . Ex3: opening a file BufferedReader myFile = new BufferedReader(new FileReader(argFilename));
Type definition: **dynamically typed language:** . No parameters type definition. . No returned type (the method returns a value if there is “return” in the body otherwise it returns none – equivalent of null in java). . No Exception defined (no checked Exception) def buildConnectionString(params) The type of a variable is determined the first time a value is assigned to a variable. > Error is raised at runtime	**Type definition:** **statically typed language:** The types are defined in the method definition: parameters, returned value and exception thrown. String buildConnectionString(String params) throws ConnectionException -> Errors are raised at compilation time. The advantage is to have a clear idea of what the method does and which types it handles.

Advantage: . allows to create generic code . Reduces Development Time. Drawback: . If not handled in a Unit Test, type checking can be generate verbose and “not-easy-to-read” code.	
Strongly Typed Language: a = 9 b = "9" c = concatenate(a, b) d = add(a, b) -> Raises an exception (it is not possible to concatenate/add a string with an int.	**Strongly Typed Language:** Idem.
Visibility (encapsulation): Private/public attributes/methods (no protected scope). Private attributes/methods are identified with a double underscore at the beginning of the name. In practical, everything trends to be public in Python (the “private” attributes and methods are identified with only one underscore so that developers are aware that the attributes/methods are “private” and shouldn't be called directly from “outside”). It is easy to update any attributes or call any methods from any other object... this could give consistency issues. Developer should be aware and disciplined.	**Visibility (encapsulation):** Notion of visibility scope of attributes and methods to ensure objects integrity (within a class only: 'private', class and derived: 'protected' or within a package: 'package protected').
Inheritance: Supports multiple inheritance.	**Inheritance:** Does not support multiple inheritance directly. But a class can implement several interfaces. This makes Java “less object-oriented” than Python? (personal thought... this depends on the developer... since that everything is possible with python, it is also possible to create non objet-oriented code). Another way to emulate multiple inheritance with Java is to use delegation with inner classes.
GUI: Qt / designer : very useful to create rapid prototypes.	**GUI:** Swing, awt (GC) Swt (no GC) Eclipse RCP (to be tested)
Code documentation: doc string The doc string is available at runtime as a method attribute (_doc_) >>> print odbchelper.buildConnectionString._doc_ Build a connection string from a dictionary Returns string.	**Code documentation:** JavaDoc JavaDoc is built at built time and is accessible in an html format.
Community: “Weak” community / java. Libraries available are often written using c native lib (that	**Community:** Large community around Java. Large amount of java extensions, libraries.

need to be compiled on every platform... this can be a limitation for the platform-dependency). There are a lot of numeric computations libraries (Numeric, Numarray, Numpy,...) The community is very open (open source, no commercial interest)	There is an intend for java to be open source before the end of the year.
PythonPath: Confusion is very easy since a PythonPath contain directories (only) that can be seen as the home directory of a python file or the home directory of a module (package directory with init.py defined). So if there is a package directory and a python file that have the same name dnaspike.py and dnaspike), the first resource will be taken into account according to the directory order within the PYTHONPATH. Ex: import dnaspike.api.Strategy In this case, python will try to load api.Strategy from the dnaspike.py rather than the 'dnaspike' directory and will fail. This is an important source of error.	**JavaPath:** Java compiled classes are stored either in directories or in jar files so the confusion described for python is not possible.
Platform independence: yes: depends on how the program is coded	**Platform independence:** yes: depends on how the program is coded
Garbage collector: Supports GC	**Garbage collector:** Idem.
Constructor: One _init_ method per class, one writes only a single constructor, with default values for the optional arguments (for that reason, there is no need to implement several constructors): * annexe4 Limitation with Eclipse/Pydev: the parameters of the constructor are not shown when typing the constructor name.	**Constructor:** Several constructors per class to allow default values: * annexe5 All the constructors are listed when invoking the constructor name within eclipse.
Parametric Polymorphism: Python "does not support" parametric polymorphism (method overloads with different parameters (~ Same as the constructor). since the language is dynamically (and implicitly) typed: a name can be bound to objects of any type (or class) without having to explicitly specify the type, and a list holds mixed type	**Parametric Polymorphism:** In java, it is possible to implement several methods with different parameters since java is statically typed. (~ see constructor section)
Callback: It is possible to define function pointers that can be passed as e.g. a parameter to other methods.	**Callback:** Java does not support callbacks but java's support of interfaces provides a mechanism by which we can get the equivalent of callbacks. The trick is to define a simple

	interface that declares the method we wish to be invoked. http://www.javaworld.com/javaworld/javatips/jw-javatip10.html
Quality assurance: **Unit Tests:** Unit tests are important, they are crucial for dynamically typed language like python. Framework: PyUnit **Code Analyzer:** PyChecker, PyLint, PyFlakes **Debugger/Profiler:** Debugger: pdb (command line debugger), eclipse (step by step). Profiler: profile module.	**Quality assurance:** **Unit Tests:** Framework: Junit **Code Analyzer:** Many available plugins for eclipse (SourceGlider, ! Analyst4j, !Byecycle, CheckStyle, ...) **Debugger/Profiler:** Debugger: eclipse (step by step) Profiler: Jconsole(embedded with jdk), !JProbe, OptimizeIt.
Build / Deployment: Due to the weak amount of available libraries written in pure python (often based on c lib, see "community" part), the build and the deployment are not obvious comparing to java (compilation on each platforms, etc...)	**Build / Deployment:** The huge amount of available libraries and building and deployment facilities (ant, maven, jar, war files, ...) makes java application easy to build, deploy, thus to maintain.
:	**:**

Conclusion

There are advantages and drawbacks in both languages... What is commonly admitted is that:

- Java is an appropriate language for creating components
- Python is good to create prototypes (short development time).
- Python is good as a "glue" language: while Java is better characterized as a "low-level" implementation language (in the sense "creating components"). In fact, the two together make an excellent combination. Components can be developed in Java and combined to form applications in Python
- Python is more for local interpreted programs, while Java is for compiled network-based apps
- Huge Projects would definitely want to use Java
- Python is fast to implement, painful to build / deploy (i.e external packages that are often based on c libraries) -> potential maintenance problem?
- Java: "less fast" implementation, application build and deployment are simple (due to the amount of available libraries written in full java, build tools available (ant, maven), easy application deployment.
- Python is well adapted for systems that performs numeric computation and/or system interfaced with c.

*annexe1:

```
public class HelloWorld{

  public static void main (String[] args){
      System.out.println("Hello, world!");
  }
```

```
}
```

*annexe2:

```
if  a > b :
    a = b
    b = c
```

*annexe3:

```
if ( a > b ){
    a = b;
    b = c;
}
```

*annexe4:

```
class Employee():

  def __init__(self,
     employeeName,
     taxDeductions=1,
     maritalStatus="single"):

     self.employeeName    = employeeName
     self.taxDeductions   = taxDeductions
     self.maritalStatus   = maritalStatus
...
```

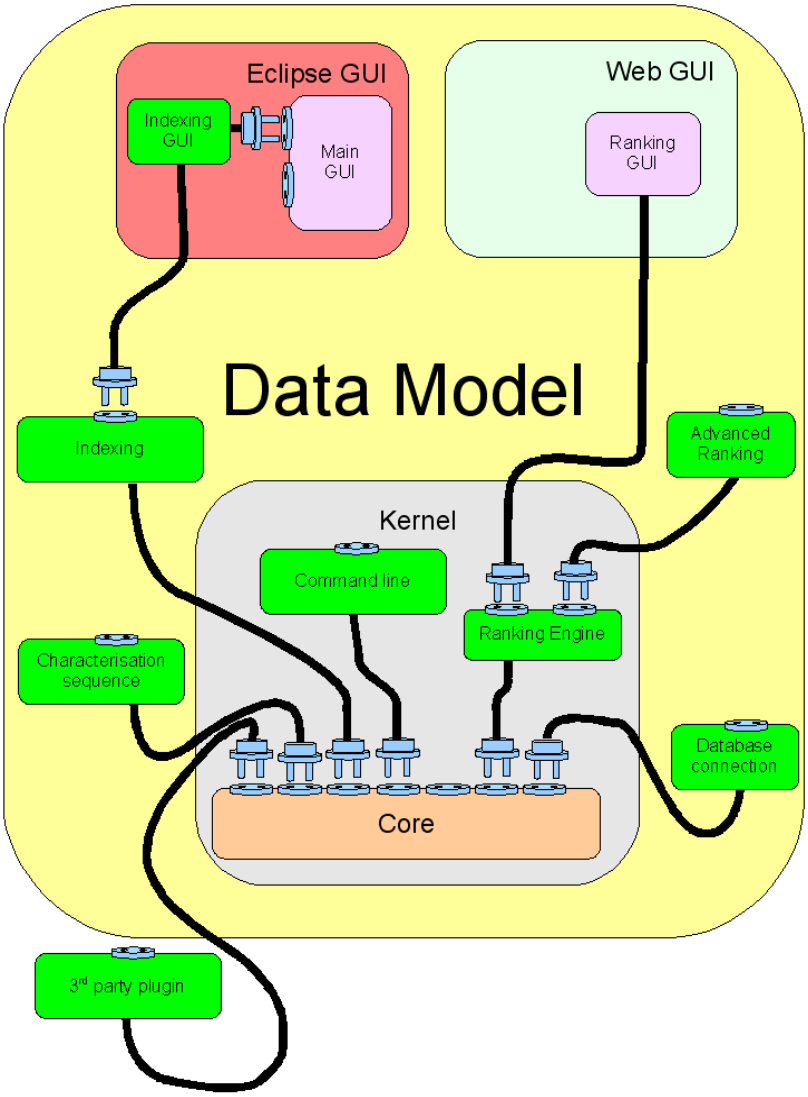
*annexe5:

```
class Employee():
public class Employee{

  public Employee(String EmployeeName,
              int taxDeductions,
              String maritalStatus){

    employeeName    = employeeName;
    taxDeductions   = taxDeductions;
    maritalStatus   = maritalStatus;
 }
  public Employee(String employeeName){

    this( employeeName, 1, ”single”);
}
```

Feedback

About the speed of Java: if raw CPU performance is an issue, Java can be speeded up very effectively by the GCC java compilers or JNative (with some restrictions, e.g. Swing not yet completely implemented). An alternative that we have used at Global Phasing is Excelsior that compiles Java bytecode to native - it makes JMol really slick and responsive..

Appendix 16 : Draft DNA 2.0 Architecture



- The architecture is based on the Model - View - Controller pattern. The kernel is the controller, the model is the DNA 2 data model and the different views can be attached.
- Plugins can be of three different kinds:
  - Completely inside the kernel (generic plugins: command line, sequencer etc.)
  - Specific plugins (indexing, strategy etc)
  - They can be completely external (no examples yet...)

Appendix 17 : Project management tools

The Trac project management tool used during the spike development worked well; however it has

many limitations. Therefore other project management tools should be taken into consideration before taking a final decision which tool to use.

The choice of the project management tools is intimately linked with the choice of license. We have first to decide if we want an open or private tool:

Open project management tools

- Examples : SourceForge and Google code
- These tools are hosted so no server installation / maintenance is needed
- The web site, wiki pages and CVS / Subversion repositories are readable for everyone (but of course only modifiable by project members)
- If mailing lists and forums are provided, they can be managed by all participants
- They can impose restrictions on the license used. For example SourceForge demands that project hosted by SourceForge are completely open source

Private project management tools

- Examples : Trac, private SourceForge etc.
- These tools needs to be installed and maintained
- As they can be hosted only by one facility / institute, it means that only this facility / institute can intervene in case of problems or maintenance. For example, the DNA mailing lists can only be maintained by ESRF staff as the list server located at the ESRF does not allow external maintenance access
- The web site, wiki pages and CVS / Subversion repositories can be made either public or private
- Therefore any license can be used

To summarise, if we want to adopt a restrictive license we don't have a choice, we must use a private project management tool. Similarly, if we take the decision to use a public project management tool (e.g. SourceForge) we must adopt an open licence.

Alternatives to Trac

SourceForge

- A reference in the open source community. Hosts thousands of projects, including many well-known like Subversion and PostrgreSQL.
- A large set of collaboration tools; mailing lists, forums etc.
- Can host mediawiki and buzgilla
- Very restrictive concerning the license, only completely open source projects

Private SourceForge

- Exactly the same features as the public SourceForge
- Free for up to 15 developers
- Must be hosted. Easy installation (WMware image)
- No restriction concerning the license

Google code

- Similar to SourceForge but not as many features
- No restriction concerning the license

Appendix 18 : Development Practices

Code Convention

- Code homogeneity
- Everybody can quickly and easily read the code produced by someone else: No Personal code style

Test-Driven Development

- Quality (early bugs detection)
- Modularity/Flexibility (if a feature is re-implemented, the tests give a quick status of potential regressions -> implementation is under control)
- Code Simplicity (Development in order to make the Unit Test pass. Easier Maintenance)
- Code comprehension (a good mean to know what a piece of code does)
- Focus (Validate the feature requirement without implementing additional unnecessary piece of code)

Code/Test Specification Review

Before implementing the Code/Test, a specification review is recommended (even very short – e.g. a few lines in an email) to make sure that a developer will implement something that will be agreed and understood (in terms of feature, fix, dependence if needed, etc...). This will be used as the code documentation basis.

Code/Test Review

- This implies validation of code convention, tests and documentation (Documentation is included in the product cycle -> a documentation that is not regularly maintained becomes quickly obsolete)
- Communication
- no « black-box » code
- Make sure that the code/feature/fix is coded based on the requirement and accordingly tested.
- Atomic review (One review per feature/fix) for Atomic Commits.

Commits

- Before committing, make sure that the previous steps are achieved
- Build and Tests must PASS in the developer machine
- Atomic Commits : One Feature/Fix per commit (Make reviews, Source Management, investigation in case of problem easier)

Continuous Integration Tool

- Launched daily on ideally each supported platform.
- It includes a sequence of automated steps that check that all is under control (Build, Code Style check, Tests,...)

- Easy Investigation in case of problem (because atomic commits, no huge diff from a stable previous state)

Conclusion

This Is The Theory (inspired form the agile methods in which Flexibility and Adaptation are the basis) In Practice, This Will Work only If the Collaboration is Based on:

- Open mindedness
- Fair-Play
- Quality Concern
- Humbleness (don't hesitate to question about his/her own methods)
- Pragmatism
- Knowledge Sharing

Appendix 19 : Documentation

The software documentation will be fully integrated in the product life cycle. Documentation will have the same priority level as Coding and Unit Testing. So a developer task won't be only Coding but "Code + Unit Test + Documentation"

Different kinds of documentation have to be considered:

Scientific Requirements (Use Cases)

-> Wiki

High Level specification (per feature)

-> OpenOffice + Wiki

Code / Test Specifications

Will be validated during the specification review and integrated in the code. The specification review validation step will be used as the basis for the PyDoc/JavaDoc that will be reviewed during the code review.

Manual

-> OpenOffice (will be checked-in in the source code.) -> online manual which could be opened from a future GUI in a web browser. For all GUI's developed we should also provide "wink videos" à la ISPyB:

http://www.esrf.fr/UsersAndScience/Experiments/MX/How_to_use_our_beamlines/howtoVideos/IspybGeneral