Journal of
Applied
Crystallography
ISSN 0021-8898

Received 24 December 2005
Accepted 27 February 2006

A program for automated optimization of initial crystallization conditions

Tahia Ben Haj Abdellatif* and Frank Kozielski*

Institut de Biologie Structurale, Laboratoire des Moteurs Moléculaires, 41 rue Jules Horowitz, 38027 Grenoble, France. Correspondence e-mail: tahia.ben-haj-abdellatif@ibs.fr, Frank.Kozielski@ibs.fr

© 2006 International Union of Crystallography
Printed in Great Britain – all rights reserved

Protein crystallization is a difficult and time-consuming task, because to obtain a crystal, optimization steps are required almost systematically. A tool that simplifies the optimization of crystallization conditions, and that can be used by any crystallographer to design a crystallization plate and to visualize its content, has become a paramount necessity. A free and open-source application has been developed to automate this task. It is based on a graphical user interface (GUI) that allows a personalized crystallization plate to be designed. All data used and generated are saved in XML documents, which allow reuse of the information. The steps involved in preparing a crystallization plate and the functions of the GUI designed to perform these steps are described. Plans for future development are presented. The program was written in Java. The application and its documentation are available under CeCILL license, which is a Free Software license agreement.

1. Introduction

Over the past few years enormous effort has been directed towards the automation of the numerous steps between gene expression and protein structure determination. The cloning of genes into expression vectors, expression and purification of proteins, crystallization, data collection, and structure determination have been more or less automated. The tedious and repetitive process of protein crystallization has been considerably improved by the use of robots that require smaller and smaller drop sizes, down to 50 nl (*cf.* 1 µl drops in manual crystallization). Thus the amount of protein solution required is considerably reduced, cutting costs for protein production and speeding up the crystallization process. In terms of speed, the robotic process is at least one to two orders of magnitude faster than the manual preparation of crystallization drops, since robots with eight parallel-working needles, or even heads with 96 needles working in parallel, are now standard.

The general 'initial screening' procedure consists of mixing protein solutions with different homemade or commercially available crystallization solutions, usually in 96-well crystallization plates, using pipetting robots, now available from manufacturers such as Tecan, Hamilton and Cartesian. The plates are then sealed and stored, and the crystallization drops are screened regularly with an optical microscope (either manually or in fully automated storage cabinets) for the presence of crystals or at least conditions that might, in the eyes of an experienced person, lead eventually to X-ray-suitable crystals.

The 'art' of crystallization now consists of separating the conditions that will not produce protein crystals from those that potentially might give protein crystals. The decision often depends on parameters based on previous experience of the experimentalist.

Surprisingly, the next step in obtaining crystals suitable for X-ray analysis, namely the refinement of the initial crystallization conditions, is still performed manually. This involves varying the different parameters that may influence protein crystallization, such as temperature, drop size, concentration of precipitant solution, buffer, pH of the solutions and salt concentration. The experimentalist has to prepare the solutions manually from different stock solutions, which also have to be made. As a consequence, in many structural biology laboratories each crystallographer 'builds up' several hundreds of different solutions and stock solutions individually, leading to large numbers of solutions for each experimentalist and an enormous investment of time. Generally, the user does not have a graphical view of the plates and cannot explicitly reuse the data.

The program *Rhombix* (Thermo Electron Corporation, https://www.thermo.com/com/cda/product/detail/1,1055,10122769,00.html) controls the automated production of solutions for protein crystallography. However, it is proprietary, expensive and only usable in combination with certain higher-end crystallization robots. There are others tools, such as *Crystal Monitor* (Emerald Biostructures, http://www.emeraldbiosystems.com/OnlineStore/pages.php?pageid=14), which only runs under Windows NT/2000, *Rock Maker* (Formulatrix, http://formulatrix.com/prod-rm.html) and *Pyramid* (Astex Therapeutics, http://www.astex-therapeutics.com). These expensive programs are not suitable tools to fulfill the need in public research to speed up crystallography.

The goal of our project was to automate the optimization of initial crystallization conditions by developing a program that is able to handle this process using crystallization robots. Our software automates the preparation of protein crystallization solutions.

(i) It offers a user-friendly graphical user interface (GUI) to design a personalized crystallization request.

(ii) It calculates the volumes for all crystallization conditions.

(iii) It stores the demand into structured documents, allowing their reuse, modification and exchange.

(iv) It will automate the generation of the programs that pilot the robot.

(v) It is based on standard, free and open-source solutions.

(vi) It can be used in and added to any robot software, because it is robot-independent.

2. Materials and methods

The application described below is computer-platform-independent and consequently can be installed on any machine. Only standard and open-source tools are used. The different decisions and tools are described in detail below.

2.1. Database format: XML

In order to be compatible with other biological and crystallization projects, we decided to structure our data with XML (http://www.w3.org/XML/) as the document format. XML is a simple, very flexible text format that is playing an increasingly important role in the exchange of a wide variety of data. It is also ideal for the long-term storage of structured data. The Protein Data Bank (PDB; http://www.wwpdb.org/) provides a representation of PDB data in XML format called PDBML (Henrick & Berman, 2004). TargetDB (Berman, 2004), which is a worldwide target search for structural genomics, provides status and information in XML. For describing crystallization screens, the Protein Information Management System project (http://www.ebi.ac.uk/msd-srv/docs/pims2.html) also uses XML. Many other initiatives are based on XML format. By design, XML separates the content from the presentation. XML itself is only responsible for representing data in a text format that is readable by all applications. Related technologies such as XSLT (http://www.w3.org/TR/xslt) are used to specify how the data in an XML document should be styled and laid out in presentation media such as HTML or PDF. Transforming an XML document to an HTML or PDF document is automatic when using an application that is based on XSLT scripts.

2.2. XML Schema

To specify the structure of each data set in the application, we need a set of tools for defining acceptable XML document structures and content. There have been many schema language proposals, such as DTD (http://www.w3.org/tr/2000/rec-xml-20001006.html) or XML Schema (http://www.w3.org/XML/Schema.html), which are both official W3C recommendations (http://www.w3c.org). It is obvious that XML Schema is more expressive than XML DTD but, because we need to express uniqueness constraints and use an object-oriented-type system for declarations (with inheritance, abstract types and finals), XML Schema is the appropriate solution. In addition, it is usable by a wide variety of applications that employ XML.

2.3. Programming language

The program has been written in Java, enabling it to be executed on multiple computer platforms. The current version was developed with Java 2 SDK 1.4, and to create the GUI, we chose Java SWING components. A wide spectrum of Java techniques and tools ensure ease of manipulating XML documents. SAX or DOM are simple parsers, which allow developers to pick out selected parts of an XML document and load the data into Java objects for later processing. Programming with these technologies is rather tedious and involves much repetitive code. Other technologies, such as JAXB, Castor and XMLBeans (http://xmlbeans.apache.org/), which are the most commonly used frameworks, enable the conversion of XML input into Java objects and Java objects into XML. We chose XMLBeans because we found it had advantages over JAXB or Castor.

(i) It fully supports XML Schema and the corresponding Java classes provide constructs for all of the major functionalities of XML Schema.

(ii) It preserves the full XML infoset.

(iii) It provides parse methods to parse an XML document.

(iv) It generates Java interfaces.

XMLBeans makes loading and saving files easy and allows us to avoid writing more Java codes and classes. Fig. 1 shows how this tool is used.

2.4. General architecture

Many institutes and laboratories use a Tecan Genesis pipetting robot (Tecan Genesis Automation, http://www.fioreauto.com/tgcs.htm) for protein crystallization. It is piloted by *GEMINI* (http://www.tecan.com/handler.asp?id=1468) scripts that are written and optimized by the experimentalist. *GEMINI* is a Tecan product that offers a control interface to help users to write their programs. *GEMINI* scripts specify the pipetting commands (*e.g. Aspirate*) and programming commands (*e.g. Start Timer*). The experimentalist has to concentrate not only on the crystallization experiment but also on how to fill the plate, how to create the worktable *etc.* The usual architecture for piloting a Tecan robot is presented in Fig. 2(*a*).

The new scheme of a crystallization request is presented in Fig. 2(*b*). At each step, such as for example '*User Identity*', '*Solution Manager*' or '*Distribution*', users can load XML files from their local disk. These files have to be well formed and valid. In addition, the user can save all inputs entered *via* the interface. The information is saved in XML documents.

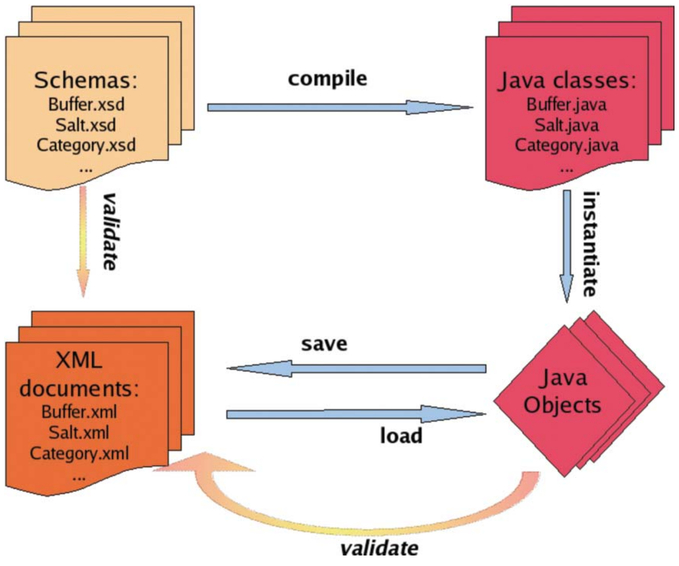


Figure 1
Using XMLBeans to load, save and validate data. XMLBeans is an XML–Java binding, which generates Java classes and interfaces from XML schemas. The generated Java classes are used to parse or generate an XML document that conforms to the schema: the application implements the Java classes and generates a content tree of data objects representing the structure and content of a source XML document. From the Java objects, we can load, modify, validate and save XML data. Loading: before loading a file, the program checks if the file is valid. It parses the XML data to a Java instance that can be manipulated. Saving: when the user enters values *via* the interface, they can be saved into an XML file. The program checks if all the mandatory fields are completed and are correct. XMLBeans creates a Java instance and then serializes it to an XML file.

3. Results

3.1. GUI functionalities

Below is a short description of the GUI and its functions. More complete documentation can be found in the manual that accompanies the program. Figures showing the GUI and its functions have also been deposited.[1] The graphical interface ensures that a complete crystallization request is written. The user is guided through the steps required to create and personalize a plate. These steps appear in the following order [steps (4)–(6) are the principal steps].

(1) *User Identity*. This requests basic user information, such as names, contact addresses, the name of the laboratory or institution and the type of project (public/industry), which is useful at later stages, such as in financial accounting. This information can be saved or loaded.

(2) *Protein Description*. When proceeding to the next window, the user is asked to provide information that is useful at later stages for statistical reasons; these data are also used to keep track of important information about the protein, its characterization and the biological risk of the sample to be crystallized, and to register all the necessary conditions for an exact reproduction of the experiment.

(3) *Plate Selection*. This allows ready-to-use commercial standard crystallization kits from different companies to be selected and/or prepared from stock solutions. In addition, these kits can be modified. Alternatively, *Create Plate* allows a completely new and unique plate to be created. This function is especially useful when optimizing potential crystallization conditions. A valid file containing a plate description can also be loaded and used. In this step, the user selects one or more plates.

(4) *New Plate Description*. Values are requested for plate types or tubes, the plate format, which can be 24-, 72- or 96-well, and the volume of the reservoir solution, which can be freely adapted to the plate format (*e.g.* between a minimum of 30 µl for 96-well plates up to a maximum of 1.5 ml for 24-well plates or 10 ml for Falcon tubes).

(5) *New Plate Solution Manager*. This constitutes the list of solutions that will be used to fill the new plate at step (6). The solutions can be either created or selected from the database of preloaded solutions. In its current state, the database contains the complete list of stock solutions available from Hampton Research, but the user can incorporate stock solutions from other companies as well. More details on the stock solutions (company, creation date of solution *etc.*) are given when clicking on the *Details* button.

(6) *New Plate Distribution Step* (Fig. 3). This is the most important window, allowing the user to compose, fill and modify the crystallization plate. All steps are reversible. The well selection can be from top to bottom or bottom to top, and from right to left or left to right. The user can specify the value of the distribution scope and the distribution pattern. The distribution scope determines which wells will contain the selected solution. The options are as follows.

(i) *one well*. The distribution concerns only one selected well.

(ii) *one line*. The user can select the entire line or select only some wells in a line.

(iii) *one column*. The user can select the entire column or select only some wells in a column.

(iv) *all lines*. The same distribution will be applied identically to each line of the whole plate or only in the selected region.

(v) *all columns*. The same distribution will be applied identically in each column of the whole plate or only in the selected region.

[1] Supplementary figures for this paper are available for the IUCr electronic archives (Reference: DD5030). Services for accessing these data are described at the back of the journal.

(vi) *plate* (*pH*, *concentration*). This concerns the distribution of a buffer. Suppose that a buffer named bicine should be used in the pH range 7.6–9.0. We can decide to distribute this buffer by varying its concentration and its pH. By using this scope, the pH variation takes effect in lines; the concentration variation takes effect in columns.

(vii) *plate* (*concentration*, *pH*). This is the same as function (vi); the difference is that the pH variation takes effect in columns and the concentration variation takes effect in lines.

The distribution pattern determines concentration changes (and/or pH for buffers) of each solution across the plate. The possible options are as follows.

(i) *constant*. The distribution is constant in the selected wells.

(ii) *min/step*. The distribution is in steps from a minimum starting value.

(ii) *min/max*. The distribution is in steps from a minimum starting value to a maximum ending value. The step value is calculated automatically.

The software offers certain features that ensure a better visual presentation for crystallization conditions, such as two different views for the creation of a new plate to be interchanged at the top-left corner of the plate.

(i) *liquids view*. This is the default view. The user can see all the solutions that have been distributed (Fig. 3).

(ii) *pH view*. The user can visualize the pH value in each well.

When solutions are distributed, their volumes are automatically calculated for each well. At any time, the user can see details about the contents of the well, such as the concentration, pH and volume of the solution, and water volume, either in table form (bottom of Fig. 3), or in a more visual representation using different colors for each well (Fig. 3, crystallization plate).

(7) *New Plate Summary* (Fig. 4). This window summarizes the main information for the crystallization plate (*e.g.* plate type, volume of reservoir). The different stock solutions used to fill the new crystal-

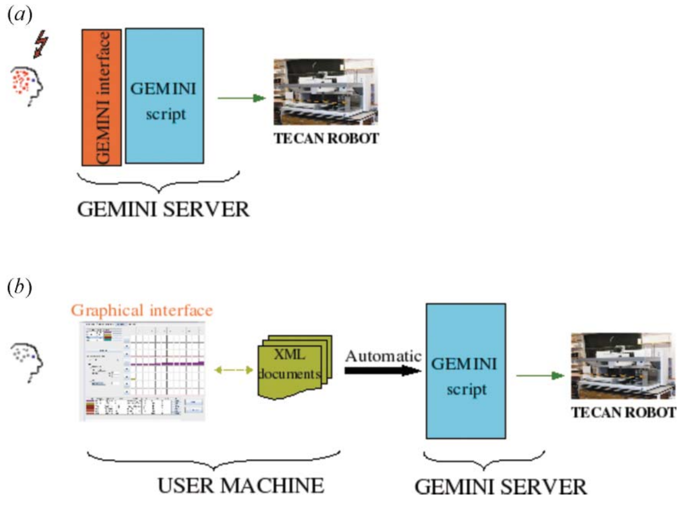
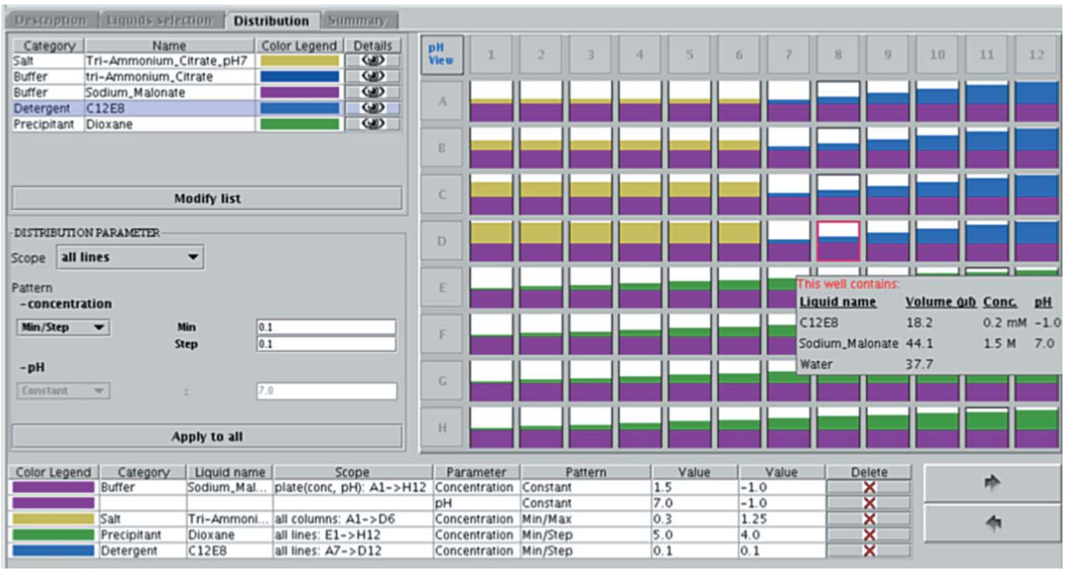


Figure 2
(*a*) The usual scheme of a crystallization experiment. The experimentalist uses the *GEMINI* interface to write the script that will be executed to pilot the robot. Because *GEMINI* normally executes a script in a linear sequence, the user has to write it in the exact order that it is to be executed. (*b*) The new scheme of a crystallization experiment. By using the new GUI, the experimentalist can concentrate only on the crystallization experiment. The way in which the plate is filled is completely hidden, because the script that should be executed to pilot the robot will be generated automatically. Users can specify demands on their own workstations and reuse the data, which can be saved and loaded from the desktop.



Category	Name	Color Legend	Details
Salt	Tri-Ammonium_Citrate_pH7	🟨	👁
Buffer	tri-Ammonium_Citrate	🟦	👁
Buffer	Sodium_Malonate	🟪	👁
Detergent	C12E8	🟦	👁
Precipitant	Dioxane	🟩	👁

DISTRIBUTION PARAMETER

Scope: all lines

Pattern

- concentration: Min/Step; Min 0.1; Step 0.1
- pH: Constant : 7.0

This well contains:

Liquid name	Volume (µl)	Conc.	pH
C12E8	18.2	0.2 mM	-1.0
Sodium_Malonate	44.1	1.5 M	7.0
Water	37.7		

Color Legend	Category	Liquid name	Scope	Parameter	Pattern	Value	Value	Delete
🟪	Buffer	Sodium_Mal...	plate(conc, pH): A1->H12	Concentration	Constant	1.5	-1.0	✗
🟪				pH	Constant	7.0	-1.0	✗
🟨	Salt	Tri-Ammoni...	all columns: A1->D6	Concentration	Min/Max	0.3	1.25	✗
🟩	Precipitant	Dioxane	all lines: E1->H12	Concentration	Min/Step	5.0	4.0	✗
🟦	Detergent	C12E8	all lines: A7->D12	Concentration	Min/Step	0.1	0.1	✗

Figure 3
New Plate Distribution Step. The solutions that have already been selected and chosen are distributed in the plate. Several functions can be used and combined to dispense the solutions. A table of distribution actions summarizes each step. There are two different views of a plate: *liquids view* and *pH view*. The default is *liquids view*, which shows the different solutions in the wells. Each solution has a color that can be personalized.

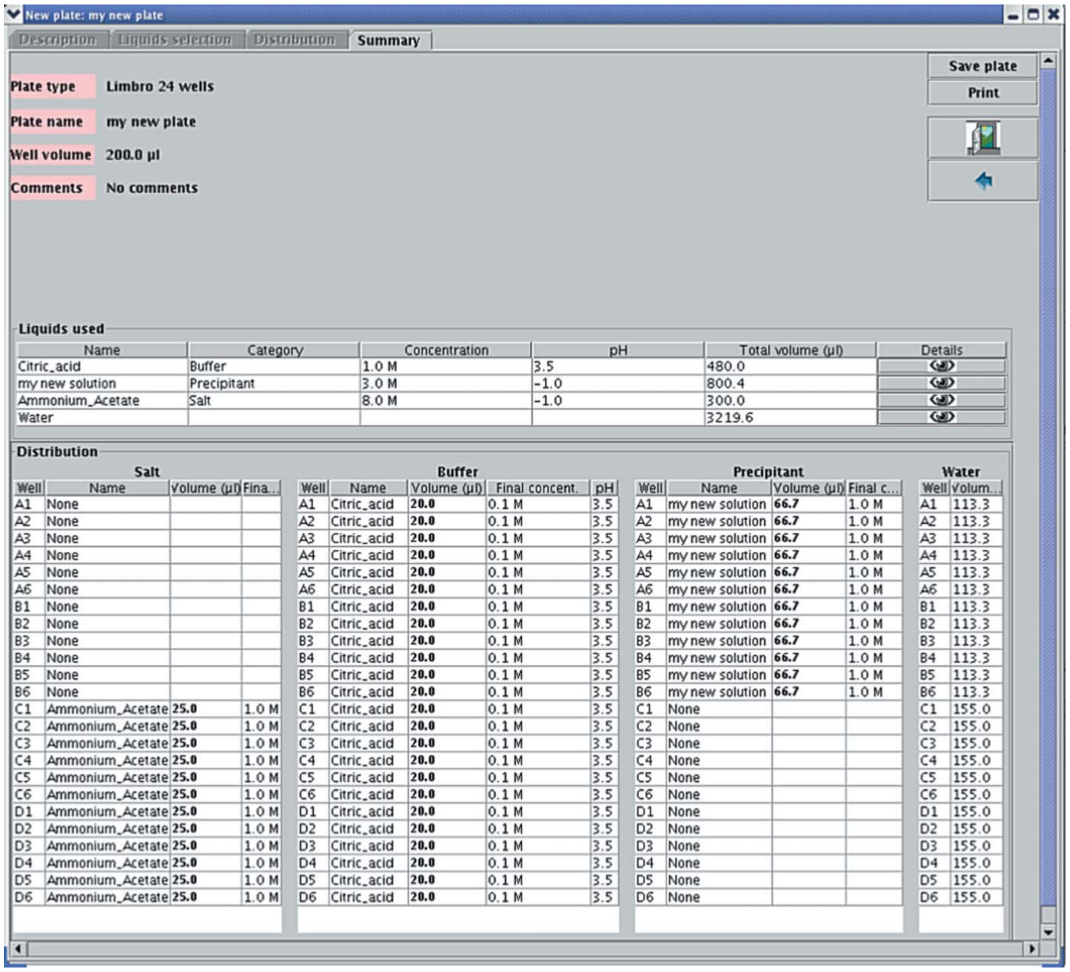


Plate type	Limbro 24 wells
Plate name	my new plate
Well volume	200.0 µl
Comments	No comments

Liquids used

Name	Category	Concentration	pH	Total volume (µl)	Details
Citric_acid	Buffer	1.0 M	3.5	480.0	👁
my new solution	Precipitant	3.0 M	-1.0	800.4	👁
Ammonium_Acetate	Salt	8.0 M	-1.0	300.0	👁
Water				3219.6	👁

Distribution

Salt				Buffer					Precipitant				Water	
Well	Name	Volume (µl)	Fina...	Well	Name	Volume (µl)	Final concent.	pH	Well	Name	Volume (µl)	Final c...	Well	Volum...
A1	None			A1	Citric_acid	**20.0**	0.1 M	3.5	A1	my new solution	**66.7**	1.0 M	A1	113.3
A2	None			A2	Citric_acid	**20.0**	0.1 M	3.5	A2	my new solution	**66.7**	1.0 M	A2	113.3
A3	None			A3	Citric_acid	**20.0**	0.1 M	3.5	A3	my new solution	**66.7**	1.0 M	A3	113.3
A4	None			A4	Citric_acid	**20.0**	0.1 M	3.5	A4	my new solution	**66.7**	1.0 M	A4	113.3
A5	None			A5	Citric_acid	**20.0**	0.1 M	3.5	A5	my new solution	**66.7**	1.0 M	A5	113.3
A6	None			A6	Citric_acid	**20.0**	0.1 M	3.5	A6	my new solution	**66.7**	1.0 M	A6	113.3
B1	None			B1	Citric_acid	**20.0**	0.1 M	3.5	B1	my new solution	**66.7**	1.0 M	B1	113.3
B2	None			B2	Citric_acid	**20.0**	0.1 M	3.5	B2	my new solution	**66.7**	1.0 M	B2	113.3
B3	None			B3	Citric_acid	**20.0**	0.1 M	3.5	B3	my new solution	**66.7**	1.0 M	B3	113.3
B4	None			B4	Citric_acid	**20.0**	0.1 M	3.5	B4	my new solution	**66.7**	1.0 M	B4	113.3
B5	None			B5	Citric_acid	**20.0**	0.1 M	3.5	B5	my new solution	**66.7**	1.0 M	B5	113.3
B6	None			B6	Citric_acid	**20.0**	0.1 M	3.5	B6	my new solution	**66.7**	1.0 M	B6	113.3
C1	Ammonium_Acetate	**25.0**	1.0 M	C1	Citric_acid	**20.0**	0.1 M	3.5	C1	None			C1	155.0
C2	Ammonium_Acetate	**25.0**	1.0 M	C2	Citric_acid	**20.0**	0.1 M	3.5	C2	None			C2	155.0
C3	Ammonium_Acetate	**25.0**	1.0 M	C3	Citric_acid	**20.0**	0.1 M	3.5	C3	None			C3	155.0
C4	Ammonium_Acetate	**25.0**	1.0 M	C4	Citric_acid	**20.0**	0.1 M	3.5	C4	None			C4	155.0
C5	Ammonium_Acetate	**25.0**	1.0 M	C5	Citric_acid	**20.0**	0.1 M	3.5	C5	None			C5	155.0
C6	Ammonium_Acetate	**25.0**	1.0 M	C6	Citric_acid	**20.0**	0.1 M	3.5	C6	None			C6	155.0
D1	Ammonium_Acetate	**25.0**	1.0 M	D1	Citric_acid	**20.0**	0.1 M	3.5	D1	None			D1	155.0
D2	Ammonium_Acetate	**25.0**	1.0 M	D2	Citric_acid	**20.0**	0.1 M	3.5	D2	None			D2	155.0
D3	Ammonium_Acetate	**25.0**	1.0 M	D3	Citric_acid	**20.0**	0.1 M	3.5	D3	None			D3	155.0
D4	Ammonium_Acetate	**25.0**	1.0 M	D4	Citric_acid	**20.0**	0.1 M	3.5	D4	None			D4	155.0
D5	Ammonium_Acetate	**25.0**	1.0 M	D5	Citric_acid	**20.0**	0.1 M	3.5	D5	None			D5	155.0
D6	Ammonium_Acetate	**25.0**	1.0 M	D6	Citric_acid	**20.0**	0.1 M	3.5	D6	None			D6	155.0

Figure 4
New Plate Summary. The figure shows as an example the creation of a 24-well plate. The window summarizes the most important details and descriptions of the plate creation. It contains a table of the stock solutions used and the contents of each well, including the volume, the concentration, the pH and the category of each solution of the well. The data can be saved or printed.

lization plate are listed, together with the total volume required for the entire plate. In the lower part of the window, the wells are systematically listed, including the different components (*e.g.* salts, precipitants, buffers, water) required to define the precipitation solutions and, most importantly, the volumes to be distributed in each well. The information can be printed and/or stored in the database and if necessary reloaded for later use.

At all steps, obligatory fields to be filled in the windows are marked with a red star. The arrow buttons are used to move to the next window or the previous one. The application also provides several error-handling mechanisms, such as

(i) *WellVolumeException*: the well capacity does not match the plate type, or the volume is negative or zero;

(ii) *MandatoryFieldsException*: mandatory fields such as user name, well volume *etc.* are not filled in;

(iii) *BufferDistributionException*: dispensing more than one buffer in a well is not allowed; in other contexts, this exception can be ignored;

(iv) *OverFlowException*: adding a solution generates the overflowing of one or more wells in the plate selection;

(v) *ConcentrationValueException*: the concentration value is negative or zero.

The errors are reported to the user in popup windows.

3.2. Data

The user can save all the information as XML files. Each file has a specific structure defined in an XML Schema. The files concern the following.

(1) **User identity**. In order to simplify the input, the user's identity data can be saved into a file that can be loaded whenever needed.

(2) **Solution**. This is the liquid needed for a crystallization request (*e.g.* ammonium formate). We classify these into categories, such as buffer, salt, detergent, precipitant, additive and solvent. Since we can add more categories and because solutions do not have exactly the same specifications, we created an abstract type that is used by restriction to define each category. The solution schema is

```
<xs:complexType abstract="true" name="Solution">
  <xs:annotation>
    <xs:documentation>It could be a Buffer, Salt, Solvent, Detergent, Additive...</xs:documentation>
   </xs:annotation>
   <xs:sequence>
     <xs:element maxOccurs="1" minOccurs="1" name="name" type="xs:string"/>
     <xs:element maxOccurs="1" minOccurs="1" name="concentration" type="ibs:Concentration"/>
     <xs:element maxOccurs="1" minOccurs="0" name="pH" type="xs:PH"/>
    <xs:element maxOccurs="1" minOccurs="0" name="volume" type="ibs:Volume"/>
   <xs:element maxOccurs="unbounded" minOccurs="1" name="origin" type="xs:string"/>
  ...
    </xs:sequence>
</xs:complexType>
```

The following is an example of the *Buffer* category structure that uses *Solution* type by restriction:

```
<xs:complexType name="Buffer">
  <xs:annotation>
    <xs:documentation>Buffer used in the Crystallography plate.</xs:documentation>
   </xs:annotation>
 <xs:complexContent>
  <xs:restriction base="ibs:Solution">
    <xs:sequence>
      <xs:element maxOccurs="1" minOccurs="1" name="name" type="xs:string"/>
      <xs:element maxOccurs="1" minOccurs="1" name="concentration" type="ibs:Concentration"/>
      <xs:element maxOccurs="1" minOccurs="1" name="pH" type="xs:PH"/>
     <xs:element maxOccurs="1" minOccurs="0" name="volume" type="ibs:Volume"/>
     <xs:element maxOccurs="unbounded" minOccurs="1" name="origin" type="xs:string"/>
...
    </xs:sequence>
  </xs:restriction>
</xs:complexContent>
 </xs:complexType>
```

The difference between *Buffer* type and *Solution* type is that the *pH* element should be present (the *minOccurs* value is set to 1), otherwise it is not a buffer. For the other liquids, such as additive, detergent and precipitant, the *pH* element cannot exist. We can imagine writing the *Solution* type in another way by adding an attribute *category* that specifies the category of the solution and keeping the *pH* element optional. We obtain the following non-abstract *Solution* type:

```
<xs:complexType name="Solution">
     <xs:annotation>
     <xs:documentation>It could be a Buffer, Salt, Solvent, Detergent, Additive...</xs:documentation>
     </xs:annotation>
     <xs:sequence>
             <xs:element maxOccurs="1" minOccurs="1" name="name" type="xs:string"/>
             <xs:element maxOccurs="1" minOccurs="1" name="concentration" type="ibs:Concentration"/>
      <xs:element maxOccurs="1" minOccurs="0" name="pH" type="xs:PH"/>
     <xs:element maxOccurs="1" minOccurs="0" name="volume" type="ibs:Volume"/>
      <xs:element maxOccurs="unbounded" minOccurs="1" name="origin" type="xs:string"/>
      ...
  </xs:sequence>
  <xs:attribute name="category" type="ibs:Category"/>
</xs:complexType>
```

We do not need to create other types, but we obtain a schema that cannot validate a buffer element. We should add a program that checks the following condition: if category is equal to buffer, the *pH* element must exist; if not the *pH* element must not exist. If we try to use this schema with a validating XML editor (such as XMLmind or Pedro), the validation will succeed but will be incomplete.

(3) **List of solutions**. The user can load or save a list of solutions. This is simpler than loading or saving solutions one by one.

(4) **Well**. A well is the reservoir into which solutions will be distributed. It contains *solution* elements and has attributes such as ‘row’ and ‘column’ that correspond to its location in the plate:

```
<xs:complexType name="Well">
     <xs:sequence>
             <xs:element name="maximumVolume" type="ibs:Volume" maxOccurs="1" minOccurs="0"/>
             <xs:element name="volume" type="ibs:Volume" maxOccurs="1" minOccurs="0"/>
             <xs:element name="salt" type="ibs:Salt" maxOccurs="unbounded" minOccurs="0"/>
     <xs:element name="solvent" type="ibs:Solvent" maxOccurs="unbounded" minOccurs="0"/>
     <xs:element name="buffer" type="ibs:Buffer" maxOccurs="1" minOccurs="0"/>
     <xs:element name="additive" type="ibs:Additive" maxOccurs="unbounded" minOccurs="0"/>
     <xs:element name="precipitant" type="ibs:Precipitant" maxOccurs="unbounded" minOccurs="0"/>
     <xs:element name="detergent" type="ibs:Detergent" maxOccurs="unbounded" minOccurs="0"/>
     </xs:sequence>
     <xs:attribute name="row" type="ibs:RowPosition" use="required"/>
     <xs:attribute name="column" type="ibs:ColumnPosition" use="required"/>
</xs:complexType>
```

(5) **Distribution**. This is an element that concerns only the software. It contains the information about how the plate has been filled. It is mainly needed when loading a plate and it is necessary for the automatic generation of Tecan *GEMINI* scripts:

```
<xs:complexType name="Distribution">
<xs:sequence>
<xs:choice maxOccurs="1" minOccurs="1">
<xs:element maxOccurs="1" minOccurs="1" name="salt" type="ibs:Salt"/>
<xs:element maxOccurs="1" minOccurs="1" name="solvent" type="ibs:Solvent"/>
<xs:element maxOccurs="1" minOccurs="1" name="buffer" type="ibs:Buffer"/>
<xs:element maxOccurs="1" minOccurs="1" name="additive" type="ibs:Additive"/>
<xs:element maxOccurs="1" minOccurs="1" name="precipitant" type="ibs:Precipitant"/>
<xs:element maxOccurs="1" minOccurs="1" name="detergent" type="ibs:Detergent"/>
</xs:choice>
<xs:element maxOccurs="1" minOccurs="1" name="scope" type="ibs:Scope"/>
<xs:element maxOccurs="1" minOccurs="1" name="pattern" type="ibs:Pattern"/>
</xs:sequence>
</xs:complexType>
```

A *distribution* element contains the following.

(i) A *solution* element, which designs the distributed solution.

(ii) The *scope* element, which follows this structure:

```
<xs:complexType name="Scope">
    <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="1" name="firstWellPostion">
            <xs:complexType>
                <xs:attribute name="rowPosition" type="ibs:RowPosition" use="required"/>
                <xs:attribute name="columnPosition" type="ibs:ColumnPosition" use="required"/>
            </xs:complexType>
        </xs:element>
        <xs:element maxOccurs="1" minOccurs="1" name="lastWellPostion">
            <xs:complexType>
                <xs:attribute name="row" type="ibs:RowPosition" use="required"/>
                <xs:attribute name="column" type="ibs:ColumnPosition" use="required"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
    <xs:attribute name="name" type="ibs:ScopeName"></xs:attribute>
</xs:complexType>
```

The *scope* element presents different logical and structured ways to fill a plate, making the creation of a plate easier. The type *ScopeName* presents the name of these different possibilities:

```
<xs:simpleType name="ScopeName">
    <xs:restriction base="xs:string">
            <xs:enumeration value="one well"/>
            <xs:enumeration value="one column"/>
    <xs:enumeration value="all columns"/>
    <xs:enumeration value="one line"/>
    <xs:enumeration value="all lines"/>
    <xs:enumeration value="plate(pH, concentration)"/>
    <xs:enumeration value="plate(concentration, pH)"/>
  </xs:restriction>
</xs:simpleType>
```

(iii) The *pattern* element determines the concentration and the pH changes of the solutions across wells. It can have the values presented by the *PatternName* type:

```
<xs:simpleType name="PatternName">
  <xs:restriction base="xs:string">
    <xs:enumeration value="constant"/>
          <xs:enumeration value="min/step"/>
    <xs:enumeration value="min/max"/>
  </xs:restriction>
</xs:simpleType>
```

To specify whether the distribution concerns concentration or pH variations, *PatternParam* makes the distinction

```
<xs:simpleType name="PatternParam">
    <xs:restriction base="xs:string">
            <xs:enumeration value="concentration"/>
            <xs:enumeration value="pH"/>
    </xs:restriction>
</xs:simpleType>
```

A *Pattern* element should conform to the following structure:

```
<xs:complexType name="Pattern">
    <xs:choice minOccurs="1" maxOccurs="1">
    <xs:sequence>
                <xs:element name="min" type="xs:float"/>
                <xs:choice>
                        <xs:element name="max" type="xs:float"/>
                        <xs:element name="step" type="xs:float"/>
                </xs:choice>
        </xs:sequence>
        <xs:element name="constant" type="xs:float"/>
    </xs:choice>
    <xs:attribute name="param" type="ibs:PatternParam"/>
    <xs:attribute name="name" type="ibs:PatternName"/>
</xs:complexType>
```

(6) **Plate**. Once the distribution phase is finished, the user can save the plate into a file that can be reloaded and modified. The plate schema is as follows:

```
<xs:complexType name="Plate">
  <xs:sequence>
    <xs:element maxOccurs="1" minOccurs="1" name="name" type="xs:string"></xs:element>
    <xs:element maxOccurs="1" minOccurs="1" name="wellVolume" type="ibs:Capacity"></xs:element>
    <xs:element maxOccurs="1" minOccurs="1" name="type" type="ibs:PlateType"></xs:element>
      <xs:sequence>
       <xs:element name="salt" type="ibs:Salt" maxOccurs="unbounded" minOccurs="0"/>
      <xs:element name="solvent" type="ibs:Solvent" maxOccurs="unbounded" minOccurs="0"/>
      <xs:element name="buffer" type="ibs:Buffer" maxOccurs="unbounded" minOccurs="0"/>
      <xs:element name="additive" type="ibs:Additive" maxOccurs="unbounded" minOccurs="0"/>
      <xs:element name="precipitant" type="ibs:Precipitant" maxOccurs="unbounded" minOccurs="0"/>
      <xs:element name="detergent" type="ibs:Detergent" maxOccurs="unbounded" minOccurs="0"/>
     </xs:sequence>
    <xs:element name="well" type="ibs:Well" maxOccurs="96" minOccurs="0">
     <xs:key name="wellPosition">
     <xs:selector xpath=".//well"/>
            <xs:field xpath="./@row"/>
            <xs:field xpath="./@column "/>
     </xs:key>
    </xs:element>
     <xs:element maxOccurs="unbounded" minOccurs="0" name="distribution" type="ibs:Distribution"/>
  </xs:sequence>
</xs:complexType>
```

A plate is composed of a list of *solution*, *well* and *distribution* elements. The solutions are those used in this plate. We should notice that in a plate each well is unique. This specification is described by the key element *wellPosition*. The column and row values determine the unique position of a well. We can attribute characteristics to a plate such as the name, capacity and type. The plate type is the following:

```
<xs:simpleType name="PlateType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Vapor Batch Douglas I. 96 wells"></xs:enumeration>
            <xs:enumeration value="Greiner 96 wells - flat bottom"></xs:enumeration>
            <xs:enumeration value="Greiner 96 wells - round bottom"></xs:enumeration>
            <xs:enumeration value="microbatch 96 wells"></xs:enumeration>
            <xs:enumeration value="microbatch 72 wells"></xs:enumeration>
            <xs:enumeration value="Limbro 24 wells"></xs:enumeration>
            <xs:enumeration value="Falcon-15ml 96 wells"></xs:enumeration>
    </xs:restriction>
</xs:simpleType>
```

(7) **List of plates**. A demand can be composed of many plates. The user can choose more than one plate to crystallize a protein. The structure of this demand will be a sequence of *plate* elements.

The XML Schemas are written in a modular way, so that different laboratories can extend or modify different types and enumeration lists such as *PlateType* and solution categories.

4. Discussion

We have developed a program for protein crystallization that has several potential applications. First of all, it can be used by a single experimentalist working in a small crystallization laboratory, who simply wants to keep track of data and to organize crystallization work. Secondly, the program is well suited to be used in a crystallization facility accessible to a large number of scientists from different public and/or private institutions, who want to optimize their initial crystallization conditions. New stock solutions can be added both to the stock solution database and physically to the collection of crystallization solutions, and be readily available for later users, avoiding repetition in the preparation of identical solutions by several persons. Overall, the use of this program will help to organize a crystallization facility, speed up the design of new crystallization plates, reduce the price per plate by avoiding repetitive preparation of stock solutions, and keep track of data that are often lost or hard to access once scientists work on other projects or leave the institution.

During the development of this program we noticed several potential improvements that will be implemented in the future:

The first version of the software uses file system data. This approach is simple and does not depend on other program packages or libraries. Nevertheless, it will be of interest to develop an interface that allows the use of an open-source XML database management system, which could be relational like mySQL (http://www.mysql.com), PostgreSQL (http://www.postgresql.fr) or Oracle (http://www.oracle.com), or native like dbXML (http://www.dbxml.com) or Berkeley DB XML (http://www.sleepycat.com/products/xml.shtml).

As Tecan robots are used in many crystallization laboratories and institutes, our next step will be to develop a program that automatically translates the XML data containing the information of the designed plate into *GEMINI* scripts that pilot the Tecan robot. XSLT technology will be used.

Because we used XML technology, we can easily add different tools for statistical investigations or for searching for information in the data (*e.g.* to search for all the plates used to crystallize the protein Eg5, or the name of the experimentalist who used the solution 0.1 *M* MES pH 6).

5. Availability and documentation

Information concerning the product is available online at http://www.ibs.fr/content/ibs/presentation/lab/LMM/robot_Xtallo/.

This project is financed through the European Integrated Project BIOXHIT (http://icarus.embl-hamburg.de/bioxhit/bioxhit.jsp). We thank Delphine Blot, Elizabeth Hewat, Fred Metoz and Jean-Luc Parouty for helpful discussions. Special thanks go to Sean McSweeny (ESRF, Grenoble) who supports the project.

References

Berman, H. M. (2004). TargetDB. http://targetdb.pdb.org/.

Westbrook, J., Ito, N., Nakamura, H., Henrick, K. & Berman, H. M. (2004). *PDBML: The Representation of Archival Macromolecular Structure Data in XML.* http://bioinformatics.oxfordjournals.org/cgi/content/abstract/bti082.