eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

ZTF-FCT
Zientzia eta Teknologia Fakultatea
Facultad de Ciencia y Tecnología
Euskadi 2006

Gradu Amaierako Lana / Trabajo Fin de Grado
Fisikako Gradua / Grado en Física

Herramientas para la mejora del Bilbao Crystallographic Server

Egilea/Autor/a:
Claudia Ezquerro
Zuzendaria/Director/a:
Gotzon Madariaga

Leioa, 2016ko irailarenen 1a /Leioa, 1 de septiembre de 2016

Resumen

El Bilbao Crystallographic Server (BCS) aloja una base de datos de estructuras moduladas (B-IncStrDB) que requiere como procedimiento de entrada la existencia de un fichero en el formato estándar CIF. Asimismo el servidor incluye un conjunto de programas que permiten el empleo de archivos CIF como archivo de entrada. Sin embargo, la ausencia de un validador global y de un proceso de extracción parcial único de la información de los archivos CIF produce errores, reduciendo la eficacia de estas herramientas. Por esta razón, el objeto del presente trabajo es la creación de una serie de programas escritos en lenguaje Python que utilizan el módulo "iotbx.cif", cuya integración en el BCS solventaría los problemas existentes actualmente. Además, se ha creado un diccionario local llamado "jana.dic" , que incluye entradas provenientes del programa JANA (el programa más utilizado para el refinamiento de estructuras moduladas) aún no contempladas en los diccionarios CIF oficiales, porque puede tener una gran utilidad en el B-IncStrDB al permitir una validación más completa y cómoda de los ficheros de entrada.

Abstract

The Bilbao Crystallografic Server (BCS) hosts a database of modulated structures (B-IncStrDB) that requires as input procedure the existence of a file in the standard format CIF. The server also includes a set of programs that allow the use of CIF files as input file. However, the absence of a global validator and the lack of an unique process of partial extraction of the information contained in the CIF files produces errors, reducing the effectiveness of these tools. For this reason, the object of this paper is the creation of a set of programs written in Python language which use the "iotbx.cif" module, whose integration into the BCS would solve the problems existed currently. Furthermore, a dictionary called "jana.dic" has been created, which includes entries from the JANA program (the program most commonly used for refinement of modulated structures) still not included in the official CIF dictionaries, because it can have great utility in the B-IncStrDB by allowing a more complete and comfortable validation of input files.

Índice

Nomenclatura

Acrónimos

ASCII	American Standard Code for Information Interchange
B-IncStrDB	Bilbao Incommensurate Crystal Structure Database
BCS	Bilbao Crystallographic Server
cctbx	Computational Crystallography Toolbox
CIF	Crystallographic Information File (Framework)
iotbx	Input/Output toolbox
STAR	Self-Defining Text Archive and Retrieval

1. Introducción y Objetivos

El Bilbao Crystallographic Server fue creado en 1997 como un proyecto de los Departamentos de Física de la Materia Condensada y Física Aplicada II de la Universidad del País Vasco. Este portal aloja bases de datos cristalográficas y programas del campo de la teoría de grupos, de la cristalografía estructural y de las aplicaciones de la física del estado sólido, todos ellos accesibles de manera gratuita a través de la URL: http://www.cryst.ehu.es.

La web está construida sobre un núcleo que aloja las bases de datos y contiene diferentes capas. Estas capas están divididas según el campo de aplicación de los programas que contienen, y el diseño de los programas permite que se retroalimenten, es decir, que empleen los resultados de otros programas. La estructura de las capas que recogen los diferentes programas está descrita en [9]. Además de las bases de datos invariables que recoge el BCS, éste también aloja una base de datos de estructuras moduladas (B-IncStrDB), en la que los usuarios pueden consultar la información cristalográfica de las estructuras recogidas o aportar nuevas entradas.

La web está en continuo desarrollo y mejora, por ello este Trabajo de Fin de Grado tiene el objetivo de contribuir en pequeña medida al desarrollo de los programas que trabajan con archivos CIF, así como a la mejora del B-IncStrDB para el que también son una parte esencial.

Actualmente, las herramientas del BCS que trabajan con archivos CIF no realizan un proceso de validación global previo a cualquier cálculo, lo que provoca errores y una reducción de su eficacia. Además, el hecho de que no se produzca una extracción parcial de la información, conduce a que los posibles errores en partes del CIF que no son requeridas para el programa con el que estemos trabajando, nos impidan que esta herramienta se ejecute.

Por lo tanto, el objetivo de este trabajo es la creación de las herramientas necesarias, para que cuando estas sean integradas e implementadas en el Bilbao Crystallografic Server, no se produzcan lo errores mencionados en los programas que emplean archivos CIF de entrada ni en el B-IncStrDB. Para que estos errores no sucedan, es necesario que estos programas realicen una validación robusta, exhaustiva y la misma para todos los componentes del BCS que trabajen con archivos CIF, además de una extracción parcial de la información contenida en los mismos. El lenguaje empleado para crear estos códigos será Python y el módulo “iotbx.cif” [6] y [7] servirá como pilar fundamental para su diseño.

Es importante recalcar, que una parte fundamental para la obtención de códigos realmente útiles que pretenden ser integrados posteriormente con los códigos existentes de los programas, es que estos cubran todas las necesidades de los mismos. Por ello, aunque se cree un código global para todos los programas, se deben particularizar las acciones llevadas a cabo para lograr cubrir las necesidades de cada uno de los programas que se pretenden mejorar. Por este motivo, una parte fundamental del proceso será el análisis riguroso de los inputs que necesita cada programa del BCS.

2. Desarrollo

El primer paso para poder desarrollar las herramientas necesarias para mejorar el BCS, era conocer y asimilar los fundamentos teóricos básicos y estudiar los dos entornos en los que se emplearán los códigos, para determinar las necesidades que se deben cubrir mediante las herramientas creadas para cada uno de ellos. Por ello, el desarrollo está dividido en dos partes. En la primera parte se presentan los conceptos básicos descritos, y en la segunda, se realiza una descripción de las partes del módulo "iotbx.cif" empleadas para la creación de los códigos, además de un análisis exhaustivo de los códigos creados y del diccionario "jana.dic".

2.1. Conceptos fundamentales

En esta parte se realiza una descripción de los conceptos básicos cuyo conocimiento previo es necesario para la creación de las herramientas que mejoren el BCS.

2.1.1. CIF

La información cristalográfica se trasmite y almacena a través del archivo llamado CIF, acrónimo de Crystallographic Information File. Este archivo fue establecido por la Unión Internacional de Cristalografía como el mejor medio para conseguir la trasmisión y el almacenamiento de la información cristalográfica. De este modo, se facilita el intercambio de información entre laboratorios, investigadores y/o publicaciones, satisfaciendo la necesidad de establecer un archivo de trasmisión global.

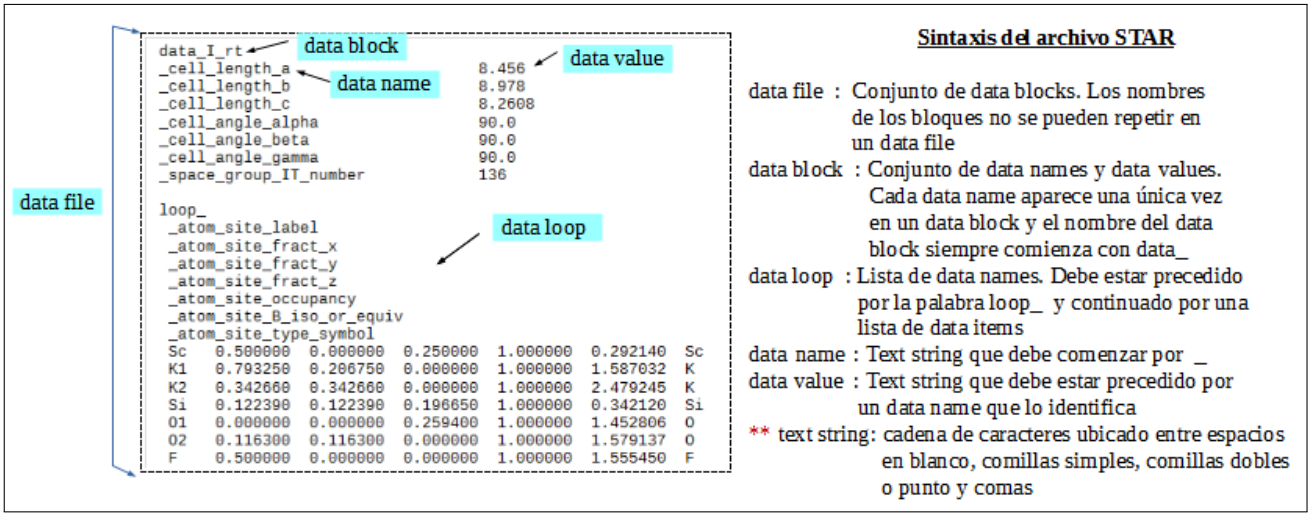

Figura 2.1: Sintaxis y terminología del archivo STAR mostrado a traves de un ejemplo

El archivo CIF está basado en el archivo STAR, acrónimo de Self-Defining Text Archive and Retrieval. El archivo STAR se caracteriza por estar escrito en texto ASCII (por lo tanto los archivos CIF que incluyan caracteres no ASCII no serán validos) y por contener un conjunto de "data blocks". Cada "data block" contiene un conjunto de "data values". Cada "data value" está determinado por un único "data name" que debe aparecer previamente, y los "data values" pueden aparecer en una lista mediante la estructura de ciclo identificada por la aparición de la palabra "loop_" antes del "data loop". Para conocer mejor la terminología y la sintaxis del archivo STAR puede consultarse la Figura 2.1.

El archivo CIF posee una serie de restricciones respecto a la sintaxis del archivo STAR que pueden ser consultadas en [4] y [5]. Por lo tanto, podríamos considerar el archivo CIF como un subgrupo del archivo STAR, ya que se imponen una serie de restricciones que compensan la perdida de generalidad, puesto que permiten la simplificación del software requerido para la creación y utilización del archivo CIF.

2.1.2. Diccionarios

Los diccionarios CIF nos proporcionan una clasificación formal de los conceptos cristalográficos ("data names") mediante la definición estricta de cada uno de ellos, como por ejemplo: el tipo de dato, las relaciones que cumple, si puede aparecer en una lista, etc. De este modo estos diccionarios nos permiten realizar una validación de los archivos CIF desde el punto de vista sintáctico, pero también desde la perspectiva del léxico y de la validez de los valores ("data values") contenidos. Para conocer la estructura de un diccionario CIF puede consultarse la Figura 2.2 .

cif_core.dic ×

```
data_cell_angle_
    loop_ _name                 '_cell_angle_alpha'
                                '_cell_angle_beta'
                                '_cell_angle_gamma'
    _category                     cell
    _type                         numb
    _type_conditions              esd
    _enumeration_range            0.0:180.0
    _enumeration_default          90.0
    _units                        deg
    _units_detail               'degrees'
    _definition
;               Unit-cell angles of the reported structure in degrees.
                The values of _refln_index_h, *_k, *_l must correspond to the
                cell defined by these values and _cell_length_a, *_b and *_c.
                The values of _diffrn_refln_index_h, *_k, *_l may not correspond
                to these values if a cell transformation took place following
                the measurement of the diffraction intensities. See also
                _diffrn_reflns_transf_matrix_.
;

data_cell_formula_units_Z
    _name                       '_cell_formula_units_Z'
    _category                     cell
    _type                         numb
    _enumeration_range            1:
    _definition
;               The number of the formula units in the unit cell as specified
                by _chemical_formula_structural, _chemical_formula_moiety or
                _chemical_formula_sum.
;
```

Figura 2.2: Extracto del diccionario "cif_core.dic"

El primer diccionario que se creó fue el llamado "cif_core.dic" y a partir de ese momento se fueron creando nuevos diccionarios complementarios que recogían los "data names" específicos necesarios para estandarizar estrictamente la descripción estructural de un material.

Los diccionarios se definen mediante un lenguaje de definición de diccionarios (DDL) basado en la sintaxis CIF. Existen dos versiones de lenguaje de definición del diccionario cristalográfico DDL (DDL1 y DDL2), que difieren en los atributos que se graban y en la expresión de las relaciones entre los diferentes elementos de datos. De hecho, el lenguaje DDL2 fue creado para abordar dos cuestiones que surgieron durante el desarrollo de un diccionario CIF para la terminología de la cristalografía macromolecular: la necesidad de describir con precisión la naturaleza jerárquica de la estructura macromolecular y las características estructurales asociadas, y para poder codificar las definiciones del diccionario de una de manera que permita una validación mediante software más detallada. Para profundizar en las características de cada uno de los lenguajes DDL consúltese la referencia [4].

Dictionary name	Description
Core CIF dictionary (cif_core.dic)	Fue el primer diccionario creado. Cubre las necesidades de los parámetros correspondientes a pequeñas moléculas cristalinas simples y cristales inorgánicos
Restraints dictionary (cif_core_restraints.dic)	Incluye parámetros que describen las restricciones aplicadas en el refinamiento de mínimos cuadrados de estructuras cristalinas
Electron density CIF dictionary (cif rho.dic)	Incluye parámetros que describen los resultados de los estudios de densidad electrónica
Powder CIF dictionary (cif_pd.dic)	Incluye parámetros que están relacionados con la difracción de polvo
Modulated structures CIF dictionary (cif ms.dic)	Incluye parámetros que describen las estructuras moduladas
Twinning dictionary (cif_twinning.dic)	Incluye parámetros utilizados para describir la estructura de un cristal polidominio

Figura 2.3: Descripción de los diccionarios DDL1 existentes a día de hoy

En el caso de los programas que trabajan con archivos CIF del BCS, teniendo en cuenta los datos cristalográficos que emplean (de los cuales se hará un análisis en la próxima sección), el diccionario "cif_core.dic" recopila todos estos "data names" y no se necesita ningún diccionario adicional. Sin embargo, en el caso del B-IncStrDB los archivos CIF de las estructuras moduladas que vamos a emplear recogen un abanico más amplio de "data names", lo que obliga a la necesidad de emplear varios diccionarios para realizar una validación lo más precisa posible. Teniendo en cuenta

los "data names" que se emplean para describir estas estructuras, se realizará una validación que incluya todos los diccionarios DDL1 existentes actualmente, junto con el diccionario "jana.dic" mencionado previamente. Para conocer el campo de la cristalografía que abarca cada uno de estos diccionarios puede consultarse el resumen incluido en la Figura 2.3.

Para finalizar, todos los diccionarios que existen actualmente y en todas las versiones creadas se pueden descargar directamente en ftp://ftp.iucr.org/pub/cifdics/ Además la página web de la Unión Internacional de Cristalografía también ofrece todos estos diccionarios en 3 modos, para facilitar su lectura: ASCII, HTML y PDF accesibles en http://www.iucr.org/resources/cif/dictionaries

2.1.3. Programas del BCS

El BCS ofrece un conjunto de programas útiles en diversos campos relacionados con la Cristalografía y algunos de ellos ofrecen el uso del archivo CIF como input.

El BCS incluye 14 programas (ver Tabla 2.1) que trabajan con archivos CIF, y además de esta opción de input, también permiten que la información de la estructura que queramos emplear sea dada mediante el formato de texto BCS. Este formato, es el formato estándar utilizado por todas las herramientas del servidor. Contiene el grupo espacial, los parámetros de red y las posiciones atómicas correspondientes a la unidad asimétrica. Se supone que esta información está dada en la configuración estándar. Para visualizar las dos opciones que ofrecen los programas puede consultarse la Figura 2.4 y la Figura 2.5.

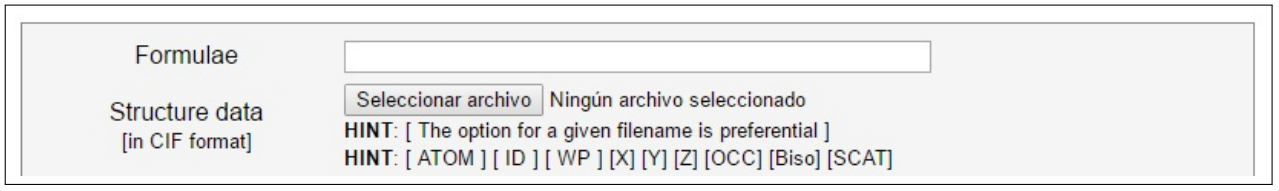

Figura 2.4: Primera opción de input que ofrecen los programas analizados del BCS. El usuario debe elegir el o los archivos CIF (dependiendo del programa) que quiera emplear y que deben estar alojados en su ordenador

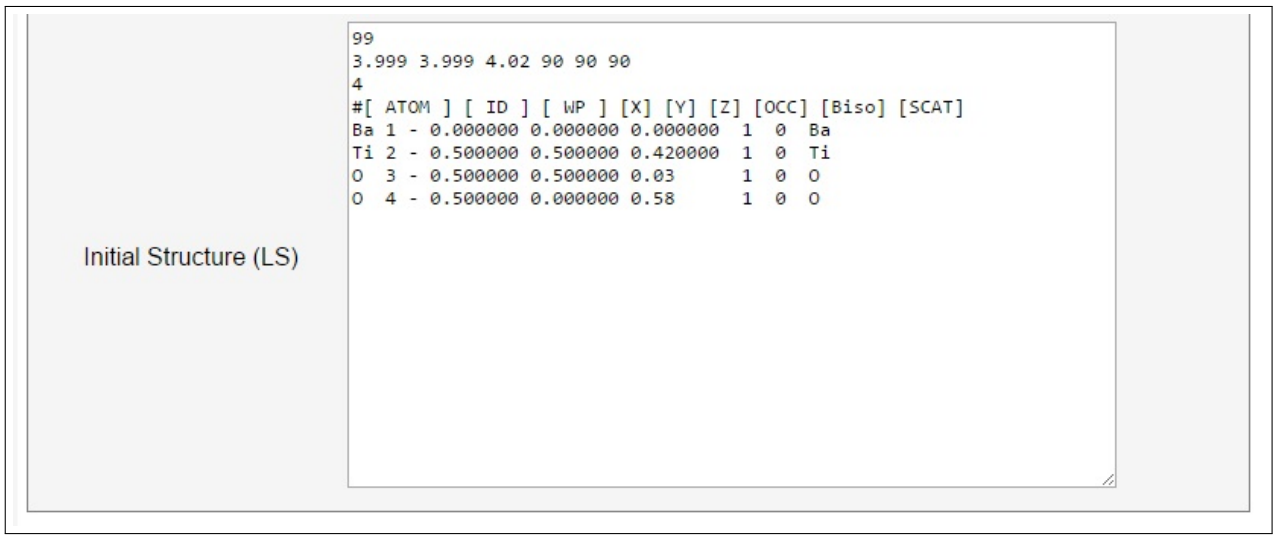

Figura 2.5: Segunda opción de input que ofrecen los programas analizados del BCS. El usuario debe rellenar el cuadro que nos ofrece esta página, cambiando los datos que están indicados como ejemplo por los de la estructura que desee analizar.
Definición de la terminología empleada en el formato BCS:
atom type/ATOM= Símbolo químico del átomo, **ID**=Número de posición del átomo en la lista, **Numb.**=Identificador numérico de un determinado tipo de átomo, **WP**= Posiciones de Wyckoff, **X,Y,Z**= Posiciones atómicas en coordenadas fraccionarias respecto a las longitudes de los ejes de la celda unidad, **OCC**=Ocupación (Fracción de un tipo de átomo presente en esa posición), **Biso**=Desplazamiento isotrópico atómico promedio , **SCAT**= Etiquetas para el cálculo del factor de scattering (en general el símbolo químico)

Algunos programas emplean un único CIF y otros emplean dos. Dentro de estos últimos, algunos comparan dos estructuras y otros emplean una estructura de baja y alta simetría. Para conocer cuales son las necesidades correspondientes a cada uno de los programas respecto a los archivos CIF de entrada puede consultarse la Tabla 2.1. Siguiendo lo recogido por esta tabla , los nuevos códigos crearán el archivo CIF que incluya los "data names" necesarios para el funcionamiento de cada programa así como un archivo de texto que incluya la información de la estructura en formato BCS emulando lo que existe actualmente. Para crear los archivos CIF necesitaremos una serie de parámetros, y para el fichero de texto otros y algunos serán requeridos por ambos. Sin embargo, no todos los parámetros del formato BCS son necesarios para el funcionamiento de los programas. En concreto las posiciones de Wyckoff (WP), BISO y las etiquetas para el cálculo del factor de scattering (SCAT) no son necesarias. En el caso de no conocer las posiciones de Wyckoff basta con incluir – en su lugar, y en el caso de BISO y de las etiquetas para el cálculo del factor de scattering, dejar un espacio en blanco.

Como se pretende crear una serie de herramientas que imiten con la máxima precisión posible la situación inicial de los programas, el archivo de texto creado clonará el formato de archivo BCS y por ello empleará el mismo criterio que el formato BCS con los parámetros citados. De igual modo, aunque los comentarios no

intervienen en la ejecución de los programas, se incluirán comentarios en aquellos programas en los que actualmente en su ejemplo vengan incluidos comentarios. Para visualizar de una forma esquemática y más sencilla los parámetros requeridos por cada programa y el destino de uso de los mismos se ha creado la Tabla 2.1 que los recoge.

	1 Cif	High & Low sym. Cifs	2 struct. Cifs	Lattice param. (a,b,c, α,β,γ)	# of indep. atoms	ITA	Atom type	x,y,z	Comm.	Atom label	ID	Numb.	Occ	WP	Biso	Scat
Amplimodes		✓		✓	✓	✓	✓	✓	✓	✓		✓		✓		
Pseudo	✓			✓	✓	✓	✓	✓		✓	✓			✓		
Dope	✓			✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓
Wpassign	✓			✓	✓	✓	✓	✓	✓	✓	✓			✓		
Transtru	✓			✓	✓	✓	✓	✓	✓	✓	✓			✓		
Setstru	✓			✓	✓	✓	✓	✓	✓	✓		✓		✓		
Equivstru	✓			✓	✓	✓	✓	✓	✓	✓	✓			✓		
Strconvert	✓			✓	✓	✓	✓	✓	✓	✓		✓		✓		
Visualize	✓			✓	✓	✓	✓	✓	✓	✓	✓			✓		
Compstru			✓	✓	✓	✓	✓	✓		✓		✓		✓		
Structure relations		✓		✓	✓	✓	✓	✓	✓	✓		✓		✓		
Sam	✓			✓	✓	✓	✓	✓	✓	✓		✓		✓		
Twin domains		✓		✓	✓	✓	✓	✓		✓		✓		✓		
Raman correlation space		✓		✓	✓	✓	✓	✓		✓		✓		✓		

Tabla 2.1: Recopilación de los parámetros requeridos por cada uno de los programas para la creación del archivo CIF y del archivo de texto. Código de colores empleado:
■ Número de cifs requeridos y características de los mismos
■ Parámetros incluidos tanto en el cif reducido como en el txt de la estructura inicial
■ Parámetros incluidos sólo en el cif reducido
■ Parámetros incluidos sólo en el txt de la estructura inicial

** Descripción de la términología empleada en la tabla: **ITA**=Número del grupo espacial de las Tablas Internacionales de Cristalografía, **Atom type**= Símbolo químico del átomo, **x,y,z**= Posiciones atómicas en coordenadas fraccionarias respecto a las longitudes de los ejes de la celda unidad, **Comm.**=comentarios, **Atom label**= Etiqueta de la posición atómica , **ID**=Número de posición del átomo en la lista , **Numb.**=Identificador numérico de un determinado tipo de átomo, **Occ**=Ocupación (Fracción de un tipo de átomo presente en esa posición) , **WP**= Posiciones de Wyckoff , **Biso**=Desplazamiento isotrópico atómico promedio , **Scat**= Etiquetas para el cálculo del factor de scattering (en general el símbolo químico)

Actualmente tal y como están construidos los programas del BCS que trabajan con archivos CIF, en el caso de que haya más de un "data block" siempre cogen el último. Esto es algo que claramente hay que solucionar, puesto que reduce su eficacia y desaprovecha las características que ofrecen estos programas. Además, en el caso de que el CIF contenga errores, no nos indica cuales son exactamente lo que dificulta la corrección de estos por el usuario.

2.1.4. B-IncStrDB

El BCS aloja el Bilbao Incommensurate Crystal Structure Database (B-IncStrDB), una base de datos de estructuras moduladas, que además es el repositorio oficial de la Unión Internacional de Cristalografía para este tipo de estructuras. Para acceder a ella podemos hacerlo a través de la propia web del BCS o directamente mediante la URL: http://webbdcrista1.ehu.es/incstrdb/.

Dentro de las estructuras moduladas, las estructuras inconmensurables son aperiódicas, en las que los átomos están sometidos a una modulación que impide que pueda definirse una Red de Bravais. Para su descripción se emplea una estructura periódica de referencia, el/los vector/es de onda de modulación y un conjunto de funciones de modulación atómica. En general, para establecer la simetría se emplean los grupos superespaciales de acuerdo al formalismo del superespacio [12]. Este es el motivo por el que se necesita la creación de una base de datos específica para ellas, ya que no es posible incluirlas en las bases de datos de estructuras convencionales. La B-IncStrDB contiene no sólo estructuras moduladas sino también los llamados compuestos de intercalación.

Esta base de datos ofrece dos posibilidades de uso, la de usuario y la de contribuyente. La base de datos recoge actualmente 138 estructuras y podemos acceder a ellas de acuerdo a la fecha de publicación, o sirviéndonos de la herramienta de búsqueda que dispone de diferentes parámetros de búsqueda. Para cada una de las estructuras podemos encontrar su información cristalográfica en formato HTML o en un archivo CIF descargable. En cuanto al uso como contribuyente, actualmente existen dos opciones para añadir nuevas entradas: enviar la información mediante el formato CIF via e-mail, o en el caso de las estructuras resueltas mediante el programa JANA [10], utilizando el output del programa e incluyendo algunos datos adicionales establecidos en el formulario que permite el añadido de nuevas entradas. A cada una de estas estructuras se le asigna un ID y una contraseña que posibilite la modificación de los parámetros por el autor de la entrada.

Se ha detectado al validar algunas entradas de la base de datos mediante la herramienta exhaustiva que se emplea en los códigos de este trabajo, que éstas poseen errores. La propia base de datos ofrece un validador, pero al validar entradas de la base de datos en las que mediante el método exhaustivo se han detectado errores esta herramienta nos dice que son válidas. Por ello, este es un error que hay que subsanar. A modo de ejemplo se ha validado la estructura con ID: 5452E0wh9M que está alojada en la base de datos utilizando el validador que ofrece el B-IncStrDB, y lo obtenido se puede ver el la Figura 2.6 . Por otro lado, se ha validado esta estructura empleando el programa de validación exhaustivo que se ha creado para ser implementado en la base de datos, y el informe de errores obtenido se puede ver en la Figura 2.7.

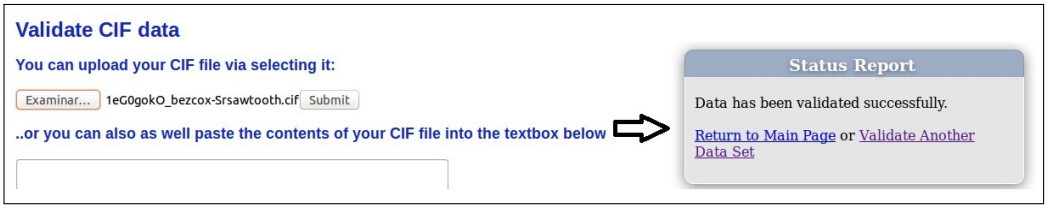

Figura 2.6: Resultado obtenido al validar la estructura de ID: 5452E0wh9M con la herramienta de validación que ofrece el B-IncStrDB

Figura 2.7: Resultado obtenido al validar la estructura de ID: 5452E0wh9M con la herramienta de validación creada en el presente trabajo

Los programas necesarios para que el proceso de incorporación de nuevas estructuras sea mucho más eficicaz son tres. Un programa que valide el CIF empleando la misma herramienta que los programas del BCS, pero empleando todos los diccionarios DDL1 y además empleando el diccionario "jana.dic" que deberá ser creado. Otro algoritmo que una vez se constate que el CIF es válido extraiga todos los "data blocks" y un último programa que permita buscar el "data value" de un "data name" dado este "data name" y el "data block" del CIF que queramos emplear.

2.2. Diseño de los programas y del diccionario "jana.dic"

2.2.1. El módulo "iotbx.cif"

En una biblioteca cristalográfica es indispensable que se ofrezcan herramientas que permitan la lectura, manipulación y validación de los archivos CIF, puesto que este es el archivo oficial estipulado por la Unión Internacional de Cristalografía para la transmisión y recopilación de la información cristalográfica. La biblioteca cristalográfica **cctbx** [11] está escrita combinando los lenguajes Python y C++ , lo que le otorga la flexibilidad de usar un lenguaje interpretado (Python) y beneficios en el rendimiento al utilizar un lenguaje compilado (C++). Dando cuenta de esta necesidad, R.J. Gildea [6] y [7] desarrollo en su tesis doctoral un nuevo módulo que al implementarse en el **cctbx** cubriese las carencias existentes respecto a la falta de aplicaciones que permitan la utilización de archivos CIF. Este módulo es "iotbx.cif ". Se trata de una nuevo programa del CIF dentro del módulo "iotbx" del **cctbx**. La biblioteca **cctbx** puede ser descargada en http://cctbx.sourceforge.net.

A la hora de describir los programas es importante recordar la terminología que se emplea para ciertas entidades en el lenguaje Python.

- Cada uno de los archivos .py se denomina **módulo**.
- Los módulos pueden formar parte de un **paquete** y un paquete se define como una carpeta que contiene archivos .py, es decir, módulos. No obstante, para que una carpeta sea considerada paquete requiere de la existencia de un archivo de inicio al que se le da el nombre __init__.py. Este archivo puede estar vacio.
- Dentro de los módulos podemos encontrar **funciones**, que es el modo de unir de manera compacta expresiones y sentencias que realicen una serie de acciones, pero que solo se ejecutarán en el caso de que sean llamadas.
- A aquellos módulos cuyo propósito es ser ejecutados se le suele denominar **script** o **programa**. Por ello, en la descripción de las herramientas se va a emplear el término programa para aquellos módulos que han sido creados para ser ejecutados, y módulo, para los que van a ser utilizados por los programas pero que no van a ser ejecutados en solitario.

Para el diseño de los programas (módulos), es necesario conocer previamente los algoritmos que hay que emplear para conseguir algunos propósitos con los archivos CIF. Por este motivo, a continuación se va a realizar una descripción de las herramientas requeridas y el modo de empleo de las mismas en el contexto del "iotbx.cif" para cada uno de los objetivos. Hay que recordar que todos los nombres dados de archivos, "data names"... , deben estar entre comillas. Por ello, por ejemplo en el caso de tener que indicar el nombre del archivo CIF tendremos que escribir "myfile.cif" en lugar de cifname, o si no, asignar previamente una equivalencia entre cifname y "myfile.cif" (la sentencia cifname="myfile.cif").

Validación de un CIF

Dentro de "iotbx.cif" existe un módulo llamado validation, que ofrece una función llamada "smart_load_dictionary" que permite la validación de un CIF. Esta función ofrece diferentes opciones de uso, y para los códigos creados vamos a emplear dos de ellas. Por un lado, vamos a emplear la que realiza la validación a partir del nombre del diccionario que queramos emplear (el modo name), que debe estar alojado en la carpeta dictionaries que contiene "iotbx.cif" y debe estar comprimido. Por otro lado, existe la opción de dar el file path del diccionario que queramos emplear (el modo file_path), que requerirá que exista un diccionario comprimido cuyo file path sea el indicado en la función. En el caso del programa creado para mejorar los programas del BCS llamado "programs", como los "data names" que aparecen están todos recogidos en "cif_core.dic", emplearemos directamente el modo name por comodidad, pero en los casos en los que se emplea "jana.dic" al no ser un diccionario oficial se ha preferido situar un diccionario total en cada una de las carpetas y utilizar el modo file_path. Para la comprensión de cómo se ha creado este diccionario total, puede consultarse el código del programa "user_dic_validation.py" en el que una parte del mismo es la creación de este diccionario (sección 2.2.2.3). Es importante remarcar, que tal y como está diseñado "smart_load_dictionary" no permite que se mezclen diccionarios DDL1 y DDL2, pero como para describir las estructuras recogidas en el B-IncStrDB es suficiente con los diccionarios DDL1, esto no supone ningún problema.

```
import iotbx.cif
from iotbx.cif import validation
cif_model=iotbx.cif.reader(file_path=cifname).model()
cifdic=validation.smart_load_dictionary(name=cifname)
error=cif_model.validate(cifdic, show_warnings=True,out=filetxt)

# En el caso de que necesitemos emplear el file path emplear
#cifdic=validation.smart_load_
#dictionary(file_path=mycif_filepath)
```

Esta herramienta es realmente completa en el caso de que nuestro diccionario sea uno de los oficiales y de que tengamos conexión a Internet, ya que en el caso de que el diccionario que tenemos guardado no sea la última versión, realiza la validación con la última versión consultándola a través de Internet.

Construcción de un CIF

"iotbx.cif" nos ofrece una serie de herramientas que permiten la construcción de un CIF. El código modelo que podríamos emplear es el siguiente:

```
import iotbx.cif
from iotbx.cif import model

# Creación de un ejemplo de model.cif, equivalente a un archivo CIF
# completo

cif = model.cif()

# Creación de un ejemplo de model.block, equivalente a un "data block"

cif_block = model.block()

#Incorporación de los "data names" que necesitemos al bloque

cif_block["_cell_length_a"] = a

#Finalmente debemos añadir cif_block al objeto CIF dando el nombre del
#"data block" que queramos emplear
cif[blockname]=cif_block
```

Para imprimir el CIF que hemos construido en un archivo CIF podemos utilizar el método show(), que ofrece un gran refinamiento del output. Las sentencias que debemos escribir son las siguientes:

```
filecif=open("mycif.cif","w")
cif.show(out=filecif)
filecif.close
```

Extracción de la información contenida en un CIF

El método más sencillo para extraer el valor de un "data name" es el que busca el "data name" directamente en el CIF dado. Para ello podemos emplear dos algoritmos diferentes, que aunque puedan parecer similares poseen una diferencia crucial. Tal y como está diseñado el "iotbx.cif", cuando empleamos el primer método y el "data name" no está en el "data block" indicado de nuestro CIF esta función produce un error y por ello si no queremos que la ejecución se detenga, debemos manipular este error. Sin embargo, con el segundo método (en el cual se emplea el módulo "builders" que está situado dentro del "iotbx.cif") en el caso de que el "data name" no esté, le asignará el valor None y de este modo no se producirá ningún error. Los dos códigos son los siguientes:

```
import iotbx.cif

# Lectura del cif

cif_model=iotbx.cif.reader(file_path=cifname).model()

# Identificación del bloque que queremos emplear

cif_block=cif_model[blockname]

# Búsqueda del valor del "data name" que queramos

datavalue=cif_block[dataname]
```

```
import iotbx.cif
import builders

# Lectura del CIF

cif_model=iotbx.cif.reader(file_path=cifname).model()

# Identificación del bloque que queremos emplear

cif_block=cif_model[blockname]

# Búsqueda del valor del "data name" que queramos

build=builders.builder_base()
build.cif_block=cif_block
a=build.get_cif_item(dataname)
```

Por otro lado, existe un método más compacto para extraer algunos parámetros cristalinos. La biblioteca **cctbx** contiene dos objetos cristalográficos centrales: "xray.structure" y "miller.array". La clase "xray.structure" comprime los objetos necesarios para el cálculo de los factores de estructura. Además, tal y como han sido desarrolladas las herramientas, ofrecen la posibilidad de exportar el objeto "xray.structure" al formato CIF, y por ello, podemos extraer las estructuras cristalinas empleando:

```
import iotbx.cif

structure = iotbx.cif.reader(file_path=cifmod).build_crystal_structures()[dataname]
```

Tal y como está construido "build_crystal_structures" obtiene la información de la estructura cristalina a partir de toda la información contenida en nuestro CIF. Por ello, algunos "data names" pueden ser obtenidos por diferentes caminos. Por esta razón, la obtención de los valores de "data names" a partir de este método es mucho más eficaz que el anterior, y esta es la razón por la que los códigos escritos emplean este método para calcular el valor de los "data names" requeridos siempre que sea posible. Algunos de los "data names" que emplean los programas no están recogidos en este objeto y por ello deberemos utilizar alguno de los dos primeros códigos en los que únicamente se busca el valor del "data name" dado en el CIF de forma estricta y sin utilizar varias fuentes.

El objeto "build_crystal_structures" únicamente se puede construir en el caso de que el CIF incluya una serie de "data names" y esto será tenido en cuenta en el programa creado para la mejora de los programas BCS ("programs") tal y como se explica en el desarrollo de los códigos (consúltese sección 2.2.2.1).

Los "data names" necesarios son: "_cell_length_a","_cell_length_b","_cell_length_c", "_cell_angle_alpha","_cell_angle_beta", "_cell_angle_gamma", "_space_group_symop_operation_xyz","_atom_site_label", "_atom_site_type_symbol", "_atom_site_fract_x","_atom_site_fract_y","_atom_site_fract_z"

El objeto "build_crystal_structures" incluye una serie de objetos dentro de los cuales están contenidos los valores de los "data names". Por ello, para poder utilizar esta herramienta es necesario conocer dónde está ubicado el "data name" que necesitamos. Una parte esencial previa a la creación de los códigos ha sido la búsqueda de esta ubicación mediante el análisis del módulo "builders" y el módulo "__init__.py" del módulo "iotbx.cif ". Mediante el siguiente código se indica cuales son las funciones requeridas para obtener los "data names" necesarios para el programa "programs" creado para mejorar los programas del BCS.

```
import iotbx.cif
from iotbx.cif import builders
structure = iotbx.cif.reader(file_path=cifmod).build_crystal_structures()[dataname]
unit_cell = structure.unit_cell()
params = unit_cell.parameters()
space_group_type=structure.space_group_info().type()
scatterers=structure.scatterers()

#Unite cell parameters

a=params[0]
b=params[1]
c=params[2]
alpha=params[3]
beta=params[4]
gamma=params[5]

#Space group

ITA=space_group_type.number()

#Atom site parameters

for i_seq,sc in enumerate(scatterers):
site = [sc.site[i] for i in range(3)]
atom label=sc.label
x=site[0]
y=site[1]
z=site[2]
OCC=sc.occupancy
UISO=sc.u_iso_or_equiv(unit_cell)

#Calculamos UISO y a partir de este BISO, ya que en "build_crystal_structures"
# BISO no está definido
```

Creación de ciclos (loops) e incorporación a nuestro CIF

Una parte fundamental cuando se desea crear un CIF es la creación de loops. El código que debemos emplear para crear un loop e incorporarlo a nuestro "data block" es el siguiente:

```
#Creación del encabezado. Debemos incluir los "data names" de los "data items"
# que contiene nuestro loop en una lista a la que se ha llamado parameterlist. En el
# caso de que exista un único "data name" debemos escribir
# la lista: (parametername,) en caso contrario no lo entenderá

loop=model.loop(header=(parameterlist))

#Asignación de los valores a cada una de las filas. En el caso general
# en el que hay varias filas deberemos utilizar un ciclo de Python

loop.add_row(valuelist)

#Incorporación del loop en el cif block

cif_block.add_loop(loop)
```

2.2.2. Descripción de los códigos creados

Todos los programas y el diccionario "jana.dic" creados están situados dentro de la carpeta TFG que ha sido comprimida para subirla a ADDI para tener una mayor comodidad. Al mismo tiempo, dentro de esta carpeta existen 3 carpetas que organizan los programas según el destino de los mismos. La carpeta llamada Cryst_EHU_Programs contiene los programas necesarios para la mejora de los programas alojados en el BCS. La denominada B-IncStrDB reúne los códigos que deberían ser empleados en el B-IncStrDB para que este servidor sea mucho mas efectivo, y para finalizar, se han creado dos programas adicionales que podrían ser útiles tanto para incorporarlos a la base de datos como para el uso personal de aquellas personas que trabajen con archivos CIF y están situados en la carpeta Validation_variations . El diccionario "jana.dic" no está dentro de ninguna subcarpeta. Para una sencilla visualización global de la estructura de todos los programas principales y de su ubicación se ha creado un mapa conceptual (Figura 2.8) que recoge esta información. Para ejecutar los programas desde la terminal debemos emplear la sentencia *iotbx.python programname.py* y en todas las carpetas ha sido añadido un archivo CIF llamado "myfile.cif" como modelo para poder observar lo que sucede cuando las herramientas son ejecutadas.

2.2.2.1. Cryst_EHU_Programs

En esta carpeta encontramos todos los componentes que utiliza el programa que se va a ejecutar "programs.py" con el fin de mejorar el BCS. Para comprender mejor la estructura que sigue este programa se ha creado un diagrama de flujo (Figura 2.9).

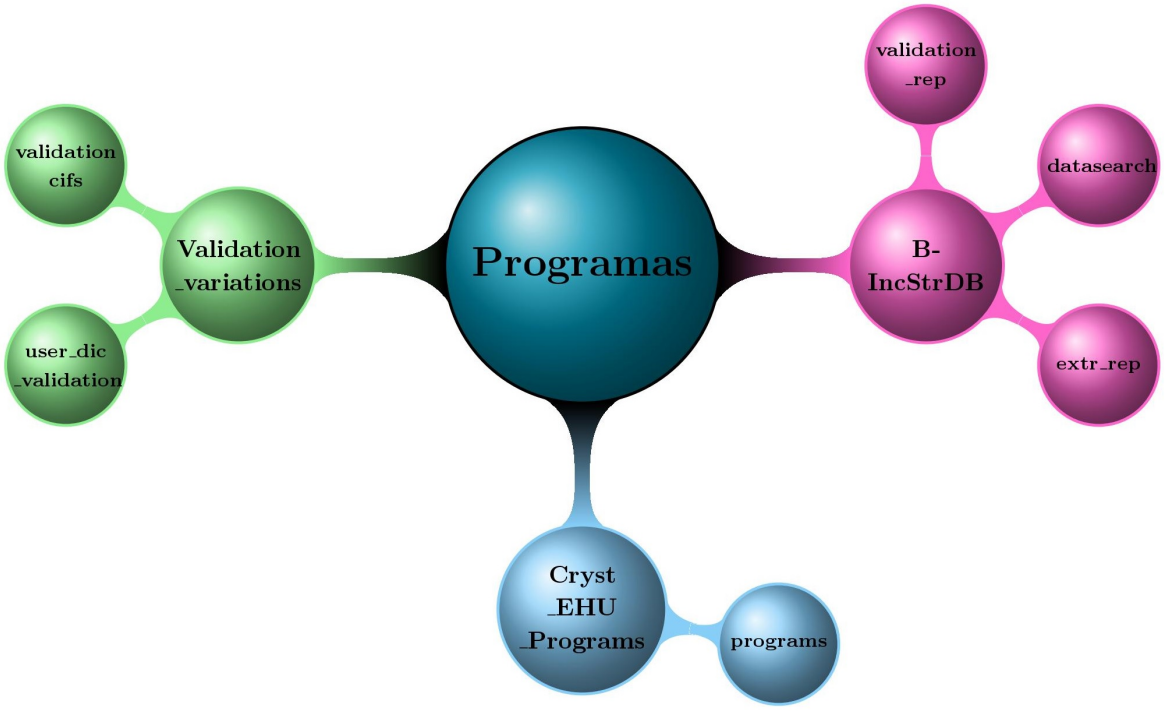

Figura 2.8: Mapa conceptual de los códigos creados y de su ubicación

* "programs.py"

- Objetivo del programa:

El objetivo de este programa es la creación de una herramienta que permita que en los programas del BCS en los que se permite el empleo de archivos CIF como archivo de entrada se realice una validación previa y una extracción parcial de la información que reduzca los errores producidos actualmente. Para el desarrollo de este programa se ha empleado toda la información del estudio previo de los programas recogida en el apartado 2.1.3 y los algoritmos descritos en el apartados 2.2.1. Los inputs de este programa serán el nombre del programa y el/los archivos CIF que queramos emplear y los outputs el/los archivos CIF que deberían entrar como input de los programas del BCS y el/los archivos de texto que clonan la otra forma de entrada de la información en estos programas del BCS.

- Descripción del programa:

La estructura empleada es la siguiente:

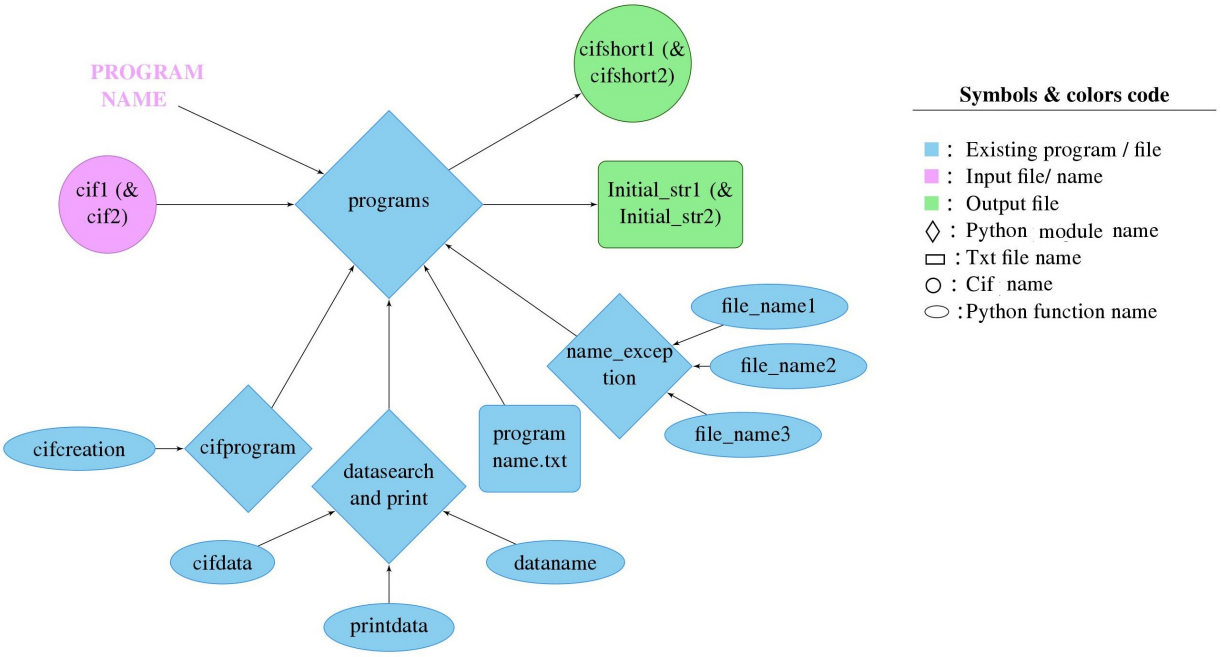

Figura 2.9: Esquema del programa "programs" creado para la mejora de los programas que trabajan con archivos CIF en el BCS.
** En el caso de que en el proceso se detecte que el CIF contiene algún error el programa se detendrá y se creará un archivo de texto llamado "errors" que recopilará los errores detectados

{"a":0, "b":0, "c":0, "alpha":0, "beta":0 ,"gamma":0,"ITA":0,"atom_label":0,"indep_atoms":0, "comments":0,"atom_type":0,"ID":0,"NUMBER":1,"OCC":0,"WP":1,"X":0,"Y":0,"Z":0,"BISO":0, "SCAT":0}

Figura 2.10: Contenido del archivo de texto "dope.txt". En su interior encontramos lo que en lenguaje Python se denomina como diccionario, es decir, a cada uno de los componentes se le asigna un valor. De este modo, cuando al componente se le ha asignado el valor **0** sabemos que debe ser incluido y cuando se le ha asignado el valor **1** indica que no estará incluido. Para conocer el destino de estos parámetros puede consultarse la Tabla 2.1 en la que se recoge esta información.

El programa principal, el script que se va a ejecutar, es "programs.py" y este emplea una serie de módulos que también han tenido que ser creados ("cifprogram", "name_exception", "datasearchandprint"). Además para recoger la información que necesita cada programa de forma compacta y poder ser modificado en el caso de ser necesario, se ha creado un archivo de texto para cada programa que recoge una clasificación de los datos necesarios y de los que no. Cuando accedemos a cada uno de estos archivos encontramos lo que en lenguaje Python se conoce como un diccionario, en el que a cada una de las entradas (la información que podemos necesitar o no dependiendo del programa) se le asigna uno de los dos posibles valores. "0" en el caso de que necesitemos ese parámetro y "1" en el caso contrario. Esto ofrece importantes ventajas respecto a otros posibles diseños de este programa: que puede ser modificado con facilidad (por ejemplo si se desea que un programa utilice más

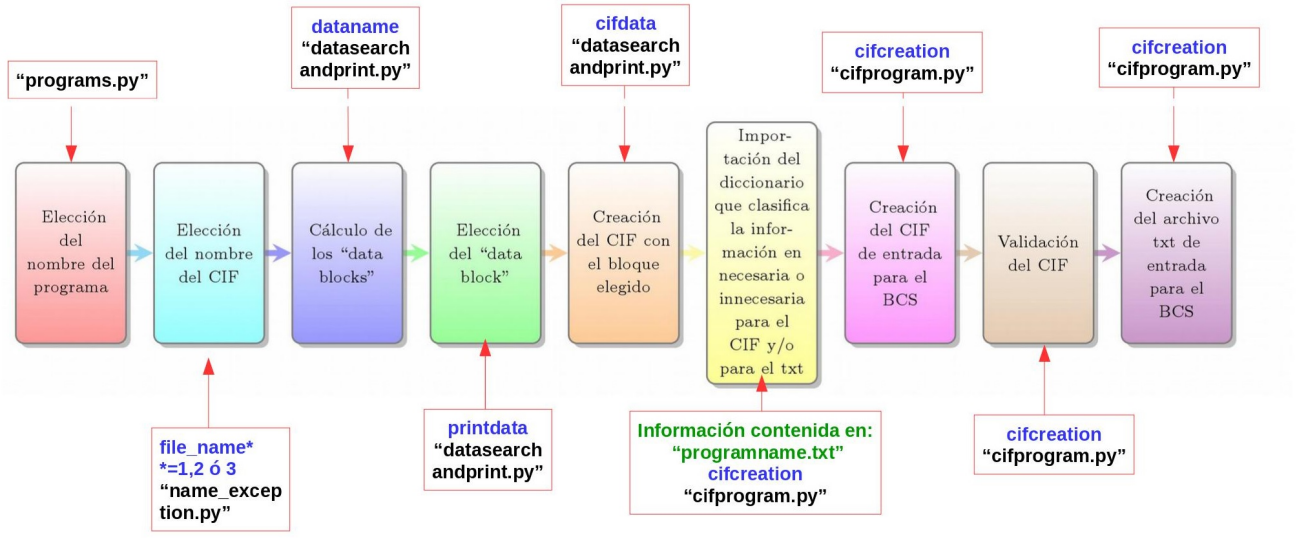

Figura 2.11: Resumen de los principales procesos llevados a cabo por el programa "programs" y el lugar en el que sucede cada uno de ellos.
* Código de color:
■ Función python
■ Módulo python
■ Archivo txt

parámetros que los actuales) y que en el caso de incorporar otro programa al BCS que emplee archivos CIF lo que tendremos que hacer será crear un archivo de texto nuevo con el nombre del programa y siguiendo la estructura descrita, además de realizar un pequeño añadido al programa principal que se explicará más adelante. Para poder observar el contenido de uno de estos archivos de texto y comprender mejor la estructura empleada puede consultarse la Figura 2.10.

Durante la creación del programa se han tenido en cuenta los posibles errores y por ello, en el momento que el programa detecta que el archivo CIF contiene algún error que impide su empleo, la ejecución se detiene y toda la información sobre el error detectado se recoge en un archivo de texto llamado "errors".

Para poder comprender mejor los principales procesos llevados a cabo durante la ejecución y seguir con mayor facilidad la descripción del código se ha creado un esquema (Figura 2.11). En lo que sigue, la descripción del programa "programs" se va a realizar siguiendo el orden de la ejecución, para poder comprender el funcionamiento y la utilidad de cada uno de los componentes de la manera más sencilla posible. El proceso llevado a cabo es el siguiente:

1) **Importación de los módulos** que se van a emplear : los módulos que se han creado específicamente para este programa y el módulo "sys" que nos va a permitir terminar la ejecución de nuestro programa siempre que lo necesitemos.

2) **Definición de la clase Logger**: esta clase va a ser empleada en muchos de los programas y módulos diseñados ya que permite que cualquier mensaje que se imprime en la pantalla sea escrito también en el archivo que se desee. De esta manera, cuando por ejemplo algunas herramientas de "iotbx.cif" detectan errores y los imprimen en la pantalla, a través de esta herramienta podemos manejar los errores e imprimirlos en el archivo que deseemos. En este programa todos los errores son recopilados en el archivo de texto "errors.txt".

3) **Petición del nombre del programa del BCS** que se quiere emplear a través de la terminal (primer input). Una vez almacenado, según el programa que queramos emplear necesitaremos 1 ó 2 archivos CIF como input y estos puede que deban ser de estructuras de alta y baja simetría o no requerir esa condición. Por esta razón, se dividen los programas según a cual de los 3 casos correspondan (En este punto es en el que deberemos hacer una modificación si queremos añadir un nuevo programa, deberíamos incluir el nombre del programa en la parte del condicional que le correspoda según sus necesidadades respecto al número de archivos CIF y de la relación entre estos). Para conocer cuales son las necesidades respecto al archivo CIF de entrada puede consultarse la Tabla 2.1.

4) **Petición del nombre del archivo CIF** que se va a emplear a través de la terminal (segundo output) mediante la función **"file_name* "** situada en el módulo "name_exception" (* será: 1 =2 archivos CIF de alta y baja simetría, 2=2 archivos CIF o 3 =1 archivo CIF según la distinción de necesidades de archivos CIF): el mensaje que se va a imprimir variará según la función empleada para que de este modo describa apropiadamente las necesidades del CIF requerido.

5) **Comprobación de que el archivo se puede abrir y de que que se trata de un archivo CIF** mediante la función **"file_name* "**: en caso de que alguna de las dos condiciones no se cumpla se llama a la clase Logger para que imprima un informe del error producido y se emplea la función sys.exit() para finalizar con la ejecución del programa. En el caso de que no existan errores esta función devolverá el nombre del archivo CIF como output.

6) **Llamada a la función "dataname"** ubicada en el módulo "datasearchandprint": a través de esta función vamos a obtener todos los "data blocks" que contienen la suficiente información cristalográfica para que los programas funcionen. Este es el proceso llevado a cabo por esta función:

- Lectura del contenido de cada línea del archivo CIF mediante un código sencillo, búsqueda de "data blocks" (palabras que posean el prefijo "data_") en cada línea e incorporación del nombre de los "data blocks" encontrados a una lista.

- Lectura de la información contenida en el archivo CIF que queremos emplear manejando los posibles errores de sintaxis mediante la clase Logger y deteniendo el proceso mediante la función sys.exit() en este último caso.
- Comprobación para cada "data block" obtenido que contiene los "data names" mínimos que permitan la construcción del objeto "build_cristal_structures" (para conocer los "data names" mínimos requeridos para el cálculo consúltese la sección 2.2.1 (Extracción de la información contenida en un CIF, pag.15)). Todos aquellos "data blocks" válidos son devueltos como output de la función mediante una lista.

* En el caso de que no exista ningún "data block" válido el proceso termina y para manejar el error se emplean de nuevo la clase Logger y sys.exit().

7) **Elección del nombre del "data block" que el usuario quiere emplear** a través de la terminal (en el caso de que el archivo CIF contenga más de un "data block" válido). Se hace una llamada a la función **"printdata"** ubicada en el módulo "datasearchandprint" cuyo input es la lista de "data blocks" válidos.

8) **Llamada a la función "cifdata"** ubicada en el módulo " datasearchandprint" : esta función construye un CIF mediante el método descrito en la sección 2.2.1 que incluye toda la información del "data block" elegido y la imprime en el archivo "cif*mod.cif" (* 1 ó 2 según si es el primer o el segundo CIF que hemos construido).

9) **Llamada a la función "cifcreation"** incluida en el módulo "cifprogram": esta función permite la creación del archivo CIF y del archivo de texto deseados. El proceso seguido por esta función es el siguiente:

- Construcción del equivalente a un archivo CIF completo y del objeto "build_crystal _structures".
- Determinación de los parámetros que necesita nuestro archivo CIF y el archivo de texto. Para ello, exportamos desde el archivo de texto que lleva el nombre del programa que queremos emplear el diccionario (en lenguaje Python) que contiene identificadores de los parámetros necesarios y de los que no.
- Construcción de todos los objetos que nos ofrece "build_crystal_structures" necesarios para el cálculo de los "data names" de los programas del BCS.
- Incorporación de los "data names" que necesita nuestro programa (de acuerdo a lo recogido en el diccionario Python y a la Tabla 2.1) al CIF que hemos creado: en primer lugar se calculan los "data names" que no aparecen en un ciclo, a continuación se crea un loop con los "data names" de la categoría "_atom_site" que necesite el programa y para finalizar se añade al CIF.

* Es importante remarcar algunos datos sobre el cálculo de algunos "data names":

- En el caso del "_atom_site_label" se han eliminado los caracteres que no correspondan al tipo de átomo o a su número, puesto que en el caso contrario pueden producirse errores en el BCS.
- Para calcular BISO tal y como se explico en la sección anterior se calcula UISO y a partir de este se realiza el calculo de BISO teniendo en cuenta que:

$$BISO = 8 \cdot \pi^2 \cdot UISO$$

- En el caso de que nuestro programa requiera el valor llamado SCAT debemos tener en cuenta que generalmente es igual a "_atom_site_type" y que en el caso del CIF debemos incluir además "_atom_site" debido a la relación de dependencia entre ambos. Los dos han sido calculados a través de "_atom_label".

*Para conocer que parámetros son imprescindibles para el archivo CIF, cuales para el archivo de texto y cuales para ambos puede consultarse la Tabla 2.1, ya que ésta especifica entre los componentes del diccionario que vamos a necesitar cual será el destino de los mismos.

10) **Impresión del CIF** creado en el archivo CIF llamado "cifshort*.cif" (* 1 ó 2 según si es el primer o el segundo CIF que hemos construido) **y validación del CIF**: en el caso de que se detecte algún error o aviso durante la validación se imprimirá en el archivo de texto "errors.txt" y se termina la ejecución.

11) **Construcción del archivo de texto**:

- Para el cálculo de las posiciones de Wyckoff se emplea el método directo combinando "_atom_site_site_symmetry_multiplicity" y "_atom_site_Wyckoff_symbol".
- Para construir el archivo de texto con un formato similar al que encontramos en el BCS, se ha empleado la función tabulate ubicada en el módulo tabulate . Esta función construye una tabla tabulada y ofrece la posibilidad de incluir un encabezado si incluimos los componentes de éste en una lista llamada headers.

* Tal y como se ha mencionado al comienzo, en el caso de que el programa trabaje con 2 archivos CIF el proceso se repite de nuevo para el nuevo CIF dado.

El hecho de que en el BCS no esté indicado en ningún lugar los "data names" que necesita el archivo CIF de entrada de cada uno de los programas ha dificultado la determinación de los mismos. En todos los casos excepto en uno ha sido posible finalmente hacerlo teniendo en cuenta la información que se requiere en cada uno de ellos para el formato BCS y realizando pruebas. Sin embargo, en el caso de DOPE no se ha logrado crear un CIF que contenga los "data names" suficientes para que el programa funcione. Al introducir el CIF creado, nos conduce a una pantalla en la que se debe elegir el supergrupo en el cual se va a buscar la pseudosimetría, pero al hacerlo nos conduce a una pantalla en la que no aparece nada.

Por lo tanto este sería un aspecto a mejorar del código que unicamente requeriría el conocimiento de los "data names" con los que trabaja DOPE que actualmente no están indicados en ningún lugar de la página.

2.2.2.2. B-IncStrDB

Dentro de esta carpeta podemos encontrar tres programas que cubren las tres necesidades que posee el repositorio: una validación exhaustiva del archivo CIF dado, la extracción de los nombres de bloques de "data" contenidos en el CIF y la extracción de la información cristalográfica que necesitemos.

*** "datasearch.py"**

- Objetivo del programa:

Este programa extrae y devuelve como output los "data blocks" contenidos en el CIF cuyo nombre es dado como input.

- Descripción del programa:

Para extraer los nombres de los "data blocks" se emplea el mismo proceso que en la funcion "dataname" descrita en el sección 2.2.2.1: lectura del contenido de cada línea del archivo CIF mediante un código sencillo, búsqueda de "data blocks" (palabras que posean el prefijo "data_") en cada línea e incorporación del nombre de los"data blocks" encontrados a una lista. Finalmente se devuelve esta lista como output.

*** "validation_rep.py"**

- Objetivo del programa:

Este programa permite la validación de un CIF cuyo nombre es dado como input, imprimiendo un archivo de texto que recoge los errores detectados en el caso de que existan y devolviendo la lista de nombres de bloques de "data" que incluye el CIF, únicamente si es constatado que el CIF es válido

- Descripción del programa:

El proceso llevado a cabo por este programa es el siguiente:

1) **Definición de la clase Logger** que permite imprimir en el archivo de texto "errors.txt" el informe del error producido, **e importación de los módulos** requeridos.

2) **Comprobación de que el archivo se puede abrir y de que se trata de un archivo CIF**: en caso de que alguna de las dos condiciones no se cumpla se llama a la clase Logger para que esta imprima un informe del error producido y se emplea la función sys.exit() para finalizar con la ejecución del programa.

3) **Lectura del archivo CIF** que queremos emplear manejando los posibles errores de sintaxis mediante la clase Logger de tal manera que los imprima en el archivo de errores también (en el caso de que se produzcan se detiene la ejecución mediante sys.exit()).

4) **Validación del CIF** empleando todos los diccionarios DDL1 y el diccionario "jana.dic": se utiliza el modo file_path de validación y el diccionario concatenado que agrupa el contenido de todos estos diccionarios (para conocer el proceso de creación de este diccionario consúltese la descripción del programa "user_dic_validation.py" en la sección 2.2.2.3) ubicado en esta misma carpeta. En el caso de que se detecten errores se imprimirán en el archivo "errors.txt".

5) **Devolución de los "data blocks"** como output mediante la llamada al módulo "datasearch" en el caso de que no existan errores **o devolución del valor 1** en el caso de que se detecten errores.

*** "extr_rep.py"**

- Objetivo del programa:

Mediante este programa, se logra la extracción parcial de la información contenida en un CIF. Previo a este programa se debe haber ejecutado el programa de validación, ya que en este programa no se contempla la posibilidad de que existan errores. Dado el nombre del CIF que se quiere emplear, el nombre del "data block" sobre el que se desea hacer la consulta y el "data name" cuyo valor y/o existencia en ese bloque se necesita conocer como input, el programa devuelve como output un valor identificativo de la existencia del "data name" (0: si está incluido en ese bloque, 1: no lo está) y el valor del mismo o en el caso de que no esté el valor -1.

- Descripción del programa:

1) **Importación de los módulos** necesarios **y definición de los dos outputs** que vamos a devolver suponiendo que el "'data name" buscado no existe.

2) **Lectura del archivo CIF** que queremos utilizar **y búsqueda del valor del "data name"** mediante el método directo.

3) **Devolución de los outputs**: en caso de que el "data name" dado no esté dentro del bloque indicado se devuelven como output los valores 1 y -1, y en el caso de que si esté se devuelve 0 y el "data value" como output.

2.2.2.3. Validation_variations

Dentro de esta carpeta se han incluido dos programas adicionales que pueden ser muy útiles para aquellas personas que trabajan con archivos CIF o para implementarlos dentro del B-IncStrDB.

*** "user_dic_validation.py"**

- Objetivo del programa:

Este programa permite la validación de un archivo CIF dado como imput empleando todos los diccionarios DDL1 y el diccionario "jana.dic". Dentro del código se realiza la concatenación de diccionarios y la compresión de este diccionario total. Los nombres de los diccionarios están situados en una lista lo que permite la modificación de los diccionarios que se quieren emplear de forma sencilla. Se supone que el diccionario "jana.dic" está situado en la carpeta dictionaries de "iotbx.cif". Si preferimos situar "jana.dic" en otra ubicación deberíamos usar en la función "smart_load_dictionary" la opción file_path.

- Descripción del programa:

1) **Importación de los módulos** necesarios incluido el módulo llamado "gzip" que nos permitirá realizar la compresión de nuestro diccionario total.

2) **Definición de la clase Logger** que permite imprimir en el archivo de texto "errors.txt" el informe del error producido.

3) **Definición de la función "cif_dic_from_str"** que permitirá la concatenación de diccionarios. Esta función posee dos parámetros de entrada: name1, que es una cadena de caracteres que incluye el contenido de los diccionarios concatenados previamente y name2, que es el nombre del diccionario que queramos emplear. Dentro de esta función a name1 se le añade la cadena de caracteres de lo contenido por el diccionario cuyo nombre es name2. Para finalizar se devuelve la concatenación como output.

4) **Definición de la función "validate"** a la que deberemos llamar para que se ejecute el programa. El proceso llevado a cabo por esta función es el siguiente:

- **Comprobación de que el archivo se puede abrir y de que se trata de un archivo CIF**: en caso de que alguna de las dos condiciones no se cumpla se llama a la clase Logger para que esta imprima un informe del error producido y se emplea la función sys.exit() para finalizar con la ejecución del programa.

- **Creación de la lista que incluye los diccionarios** que se desean emplear. Como se pretende que ésta pueda ser modificada, se consideran todas las opciones posibles incluso que la lista esté vacía (en este caso se emplea el diccionario "cif_core.dic").
- **Llamada recursiva a la función "cif_dic_from_str"** para la concatenación de diccionarios: name1 es el contenido que ya ha sido concatenado en forma de string y name2 el nombre del diccionario cuyo contenido queremos añadir.
- **Impresión del contenido** de nuestro diccionario total en un archivo ".dic" que se ubicará en esta misma carpeta y **compresión del diccionario**.
- **Lectura del archivo CIF** que queremos emplear manejando los posibles errores de sintaxis de tal manera que los imprima en el archivo de errores también mediante la clase Logger (en el caso de que se produzcan se detiene la ejecución mediante sys.exit())
- **Validación del CIF** empleando nuestro diccionario concatenado: utilizamos la forma file_path indicando la ubicación del diccionario que hemos creado.

*** "validationcifs.py"**

- Objetivo del programa:

Este programa permite la validación de un conjunto de archivos CIF de forma automática guardados en la carpeta denominada cif situada dentro de la carpeta Validation_variations, al igual que este programa. Por cada archivo CIF se creara una carpeta con el nombre "nombre del cif" dentro de una carpeta llamada cif_errors y está contendrá un archivo de texto llamado "nombre del cif_errors" que recogerá los errores de validación detectados.

- Descripción del programa:

1) **Importación de los módulos** necesarios incluido el módulo llamado "os" que nos permitirá realizar una exploración de los archivos contenidos en la carpeta cif.

2) **Búsqueda de los nombres de los archivos CIF** contenidos por la carpeta cif **e incorporación de los nombres a una lista**.

3) **Creación de un ciclo** que se realice para cada archivo CIF. Dentro de este ciclo se realizarán las siguientes acciones:

- **Definición del file path** del archivo de texto que contiene los errores **y creación de la carpeta que contiene los errores del CIF** llamada "nombre del cif" ubicada dentro de la carpeta "cif_errors" (que también será creada)
- **Creación del archivo de texto** que contendrá el informe de errores **y lectura del archivo CIF** que queremos emplear manejando los posibles errores de sintaxis de tal manera que los imprima en el archivo de errores también.
- **Validación del CIF** empleando todos los diccionarios DDL1 y el diccionario "jana.dic". Para lograrlo, se utiliza el modo file_path de validación y el diccionario concatenado que agrupa el contenido de todos estos diccionarios (para conocer el proceso de creación de este diccionario consúltese la descripción del programa "user_dic_validation.py" en la sección 2.2.2.3) ubicado en esta misma carpeta. Los errores son recogidos en el archivo de texto cuyo file path se ha definido al comienzo del ciclo.

2.2.2.4. "jana.dic"

Entre los "data names" que incluyen las estructuras moduladas del B-IncStrDB encontramos algunos caracterizados porque empiezan por el prefijo jana. Estos son algunos de los obtenidos en el output del programa JANA [10] que permite dado el diagrama de difracción de una estructura refinarla, es decir, encontrar los parámetros atómicos para los que el diagrama de difracción calculado se aproxime lo máximo posible al observado.

Sin embargo, pese a su gran utilidad, estos parámetros no están recogidos en ninguno de los diccionarios actuales, lo que impide una validación exhaustiva de ellos. Lo único que conseguiremos al validar un CIF que contiene estos "data names", es que el validador nos indique que estos parámetros no existen.

Por ello, para crear un validador lo más exhaustivo y robusto posible, se ha creado un diccionario llamado "jana.dic". Este diccionario está incluido también en la carpeta TFG junto a las demás herramientas creadas. Para la creación de este diccionario se han analizado todos los archivos CIF de las estructuras alojadas en el B-IncStrDB y se han buscado todos aquellos "data names" que comienzan por el prefijo jana. El motivo de usar este método, es que tal y como se ha explicado anteriormente, no están definidos en ningún lugar hasta ahora. Para crear el diccionario se ha empleado el lenguaje DDL1. El diccionario, tal y como sucede en los diccionarios oficiales, está dividido en las correspondientes categorías de los "data names", lo que le otorga la característica jerarquía que poseen los diccionarios.

Los atributos DDL1 que se han incluido son:

- **Atributos de identificación**: el atributo de identificación es name_ y da el nombre del parámetro
- **Atributos de clasificación**: entre estos atributos se han empleado _list que indica si un parámetro puede aparecer en un ciclo de lista, _type que posee las opciones numb (para números), char (para cadenas de caracteres) y null (para describir un diccionario) y además se ha incluido _type_conditions en aquellos parámetros cuyo valor es susceptible de poseer una desviación estándar.
- **Atributos de relación**: entre estos atributos se han empleado _category que indica la categoría a la que corresponde el parámetro y _list_reference que identifica para aquellos parámetros que aparecen en listas el parámetro clave, es decir el que debe aparecer obligatoriamente en esta lista junto a este parámetro.

En el caso de necesitar añadir parámetros adicionales este diccionario es fácilmente extensible añadiendo los nuevos componentes siguiendo la misma estructura, es decir, dotando al parámetro que queramos incluir de los atributos descritos en esta sección.

3. Conclusiones

Este trabajo de fin de grado me ha acercado a la rama de la cristalografía además de enseñarme a valorar la importancia del empleo de la programación para el desarrollo de la misma. El proceso de profundización sobre los fundamentos de los archivos CIF, sobre los diccionarios y sobre una parte del módulo "iotbx.cif" previo a la creación de los códigos, me ha permitido adquirir conocimientos de los mismos.

Por otra parte, el trabajo de elaboración de los códigos me ha permitido desarrollar y aumentar mis conocimientos de programación, en concreto, los del lenguaje Python que ha sido el empleado para la elaboración de los códigos cuya creación era el principal objetivo de mi TFG.

La integración de los programas creados en el BCS permitiría que la validación en el servidor sea global, es decir, que se utilice el mismo método en todos los componentes del mismo que utilizan archivos CIF, aumentando considerablemente su eficacia. Además, se realizaría una extracción parcial de la información requerida por cada uno de los programas, lo que evitaría errores que aparecen actualmente. Los códigos creados para el B-IncStrDB permitirán que la incorporación de nuevas entradas a esta base de datos sea más sencilla y se comprobará que todos los datos de las estructuras incluidos sean válidos puesto que previamente a la incorporación de una nueva entrada se realizará un análisis exhaustivo de la validez de la misma.

En el caso particular del programa "programs" a partir del nombre del programa que queramos emplear y de los cifs que quiera emplear el usuario, el programa extrae los diferentes "data blocks" contenidos en el mismo, y en el caso de que haya más de uno, el usuario elegirá el que quiera emplear. Una vez conocido este dato, el programa creará un CIF con los "data names" requeridos para que el programa funcione, así como, un archivo de texto con la estructura inicial en formato BCS (el otro modo de trabajo de los programas). En el caso de que en el proceso se detecte algún error en el CIF de entrada, el programa finalizaría, generando un informe de los errores detectados recogido en el archivo de texto "errors". Actualmente, en el caso de que haya varios "data blocks" sólo se recoge la información contenida en el último, además al no realizarse una validación, en el caso de que el CIF contenga errores en partes del CIF que no nos interesen el programa nos dará error pero no sabremos exactamente cuales son los errores que contiene el CIF.

Es importante remarcar que en ningún lugar del BCS aparecen estipulados los "data names" que necesita el archivo CIF de entrada de cada uno de los programas. En todos los casos excepto en uno ha sido posible finalmente determinarlo, teniendo

en cuenta la información que se requiere en cada uno de ellos para el formato BCS y realizando pruebas. En el caso de DOPE no se ha logrado crear un CIF que contenga los "data names" suficientes para que el programa funcione, ya que aparece una página en blanco, tal y como se ha explicado previamente.

Para el caso concreto del B-IncStrDB, se han creado tres programas que cubren las carencias existentes actualmente. El primer programa valida los cifs empleando todos los diccionarios DDL1 oficiales y el diccionario "jana.dic" que he creado (la validación se realiza mediante la misma herramienta que se emplea en los programas, globalizando de este modo el proceso en todo el BCS), y crea un archivo de texto llamado "errors" en el caso que se detecten errores. El segundo, extrae los "data blocks" contenidos en el CIF dado y se ejecutará sólo si el CIF es válido. El último programa, buscará el "data name" que nosotros le indiquemos para el "data block" de nuestro cif que queramos emplear.

Además de los programas que se requerían he creado dos programas adicionales que podrían ser útiles para el uso personal de aquellos que trabajan con archivos CIF o incluso para incorporarlos al B-IncStrDB. Uno de ellos permite la validación de todos los archivos CIF que se deseen de manera automática con el único requerimiento de que sean guardados en la carpeta "cif" que esta ubicada junto a este programa. Al ejecutarse se creará una carpeta para cada CIF que incluirá un archivo de texto que recoge los errores detectados. Por otro lado, también he incluido un programa que valida el CIF dado por el usuario con los diccionarios que éste desee. Para finalizar he creado un diccionario llamado "jana.dic" que no existía hasta ahora y que puede ser muy útil, ya que permitirá validar algunos los parámetros obtenidos como output del programa JANA que actualmente no están recogidos en ninguno de los diccionarios existentes.

A modo de último apunte me gustaría destacar que ha sido un proyecto muy positivo. A traves de él, de un modo práctico he aumentado notablemente mis conocimientos sobre cristalografía, programación en lenguaje Python, así como mi capacidad para crear mis propios códigos analizando la situación inicial y lo que se quiere obtener. Además, el hecho de que la integración en el BCS de estas herramientas logre mejorar éste en una pequeña medida y que tenga una clara utilidad es algo muy satisfactorio.

Bibliografía

[1] Lutz M. and Ascher D. *Learning Python*. Sebastopol, CA: O'Reilly, 2004.

[2] Lutz M. *Programming Python* . Sebastopol, CA: O'Reilly, 2001.

[3] Ceder N. R. *The quick Python book* . Greenwich : Manning, 2010.

[4] Hall S. and McMahon B. *International Tables for Crystallography, Vol. G.* New York : Springer, 2005.

[5] Hall S. R., Allen F. H. and Brown I. D. The Crystallographic Information File (CIF): a New Standard Archive File for Crystallography.*Acta Cryst.* Vol. A47: 655-685, 1991.

[6] Gildea R.J. , Bourhis L.J., Dolomanov O.V., Grosse-Kunstleve R.W., Puschmann H., Adams P.D. and Howard J.A. iotbx.cif: a comprehensive CIF toolbox. *Journal of Applied Crystallography* Vol. 44: 1259–1263, 2011.

[7] Gildea R. J. Application of Modern Techniques in Crystallographic Software Development. Department of Chemistry Durham University. 2011.

[8] Aroyo M. I., Perez-Mato J. M., Capillas C., Kroumova E., Ivantchev S., Kirov A., Madariaga G. and Wondratschek H. Bilbao Crystallographic Server: I. Databases and crystallographic computing programs . *Z.Kristallogr.* Vol. 221: 15-17 , 2006.

[9] Tasci E.S., de la Flor G., Orobengoa D. ,Capillas C., Perez-Mato J.M. and . Aroyo M.I An introduction to the tools hosted in the Bilbao Crystallographic Server. *EPJ Web of Conferences* Vol. 22, id.00009 , 2012.

[10] Petricek, V., Dusek, M. and Palatinus, L. *Z. Kristallogr.* Vol. 229(5): 345-352, 2014

[11] Grosse-Kunstleve R.W., Sauter N. K., Moriarty N. W. and Adams P. D. *J. Appl. Cryst.* Vol. 35: 126–136, 2002

[12] Janssen T., Janner A., Looijenga-Vos A. and de Wolff P. M. *Incommensurate and commensurate modulated structures. International Tables for Crystallography, Vol. C, ch. 9.8.* Dordrecht : Kluwer Academic Publishers, 2004.