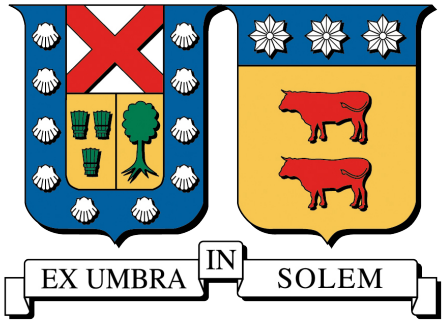
2018

AUTOMATIZACIÓN DEL PROCESO DE INCORPORACIÓN DE DATOS GENERADOS POR ASTRÓNOMOS AL OBSERVATORIO VIRTUAL

SARD CRISTI, PATRICIO HERNAN

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



AUTOMATIZACIÓN DEL PROCESO DE INCORPORACIÓN DE DATOS GENERADOS POR ASTRÓNOMOS AL OBSERVATORIO VIRTUAL

PATRICIO HERNÁN SARD CRISTI

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

PROFESOR GUÍA	:	SR. MAURICIO ARAYA
PROFESOR CORREFERENTE	:	SR. MAURICIO SOLAR

JULIO 2018

A mi familia . . .

Resumen

En el presente trabajo se propone una interfaz y un protocolo que satisfacen la necesidad que tienen los astrónomos de publicar los resultados de sus investigaciones. Para lograr este objetivo se elaboró una propuesta del proceso de ingestión de datos de alto nivel al Observatorio Virtual Chileno (ChiVO), con el fin de que esta entidad publique los datos ingresados, cumpliendo con los estándares internacionales que se demandan.

La elaboración de esta propuesta consiste en la definición de los requerimientos básicos para el desarrollo de esta interfaz, definiendo los tipos de usuarios del sistema y las características de los datos aceptados para su ingestión a ChiVO, además de la definición del proceso que genera automáticamente, los archivos de configuración para realizar la publicación de los datos al Observatorio Virtual.

Para ratificar la propuesta se desarrolló un prototipo funcional considerando un sub conjunto de los requerimientos planteados, asimismo el prototipo se usó como base para la estimación de los costos de desarrollo para la interfaz completa.

Palabras Clave: Observatorio Virtual, Publicación de Datos, Observaciones Astronómicas.

Abstract

In the present work an interface and a protocol that satisfy the need that the astronomers have to publish the results of their investigations are proposed. To achieve this goal, the process of ingestion of high-level data to the Chilean Virtual Observatory (ChiVO) was formulated in order for this entity to publish the entered data, complying with the international standards that are demanded.

The elaboration of this proposal consists in the definition of the basic requirements for the development of this interface, defining the types of users of the system and the characteristics that the data must have to be accepted in ChiVO, as well as the definition of the process that generate automatically, the configuration files to make the publication of the data to the Virtual Observatory.

To ratify the proposal, a functional prototype was developed considering a subset of the proposed requirements, and the prototype was used as a basis for estimating the development costs for the complete interface.

Glosario

- **ADQL**: Astronomical Data Query Language. Lenguaje de consulta basado en la tercera versión de SQL (SQL92), desarrollado para soportar operaciones comunes en el ámbito de la astronomía.
- **BPMN**: Business Process Model and Notation. Método usado para modelar procesos de negocio de forma gráfica.
- **Catálogo**: Conjunto de registros de objetos astronómicos que comparten características comunes y generalmente son resultado de un *survey* astronómico.
- **ChiVO**: Observatorio Virtual Chileno. Uno de los tantos proyectos de Observatorios Virtuales nacionales existentes en el mundo.
- **FITS**: Flexible Image Transport System. Un formato de archivo usado para representar imágenes u otros tipos de datos, además este formato también guarda metadatos en su cabecera.
- **IVOA**: Alianza del Observatorio Virtual Internacional. Una organización que intenta facilitar el descubrimiento y acceso a los datos astronómicos, que se encuentran en diferentes centros y bases de datos en el mundo, para lo cual define varios estándares y protocolos.
- **ObsCore**: Observation Data Model Core. Estándar de la IVOA que define los componentes principales del modelo de datos observacionales, necesarios para el descubrimiento de datos astronómicos en el Observatorio Virtual.
- **SCS**: Simple Cone Search. Protocolo definido por la IVOA que permite el descubrimiento y acceso a registros cuya posición se encuentra en el cono definido de la consulta.
- **SIAP**: Simple Image Access Protocol. Protocolo definido por la IVOA que permite el descubrimiento y acceso a datos de imágenes multidimensionales.

- **SSAP**: Simple Spectral Access Protocol. Protocolo definido por la IVOA que permite el descubrimiento y acceso a datos de espectros unidimensionales.
- **TAP**: Table Access Protocol. Protocolo definido por la IVOA que permite el descubrimiento y acceso a datos tabulares.
- **OV**: Observatorio Virtual. Ecosistema formado por el conjunto de datos (repartidos en varios centros de datos, bases de datos, archivos, etc.) y diferentes herramientas para acceder a estos, capaces de trabajar interoperablemente. El Observatorio Virtual está conformado por varias iniciativas nacionales e internacionales.
- **Survey**: En el contexto de la astronomía corresponde a una observación astronómica que barre una zona del espacio, concentrando la atención en una región más que a un objeto astronómico en particular.
- **XML**: Extensible Markup Language. Lenguaje adoptado como estándar para la representación de datos e intercambio de información.

Índice de Contenidos

Capítulo 1 : Introducción

La acelerada revolución digital de los últimos 60 años y la siempre creciente presencia de los sistemas computacionales, han propiciado importantes cambios de paradigma en el quehacer humano. Un ejemplo de esto son las observaciones astronómicas, en donde cada vez es menos común el concepto del astrónomo *in situ*. Por otro lado, el procesamiento y flujo de grandes volúmenes de datos astronómicos, a través de la Internet, ha ido en aumento exponencial durante los últimos años. Estos datos, almacenados en varios centros de datos distribuidos por todo el mundo, en combinación con todo un ecosistema de programas y herramientas desarrolladas específicamente para su acceso, visualización y análisis, conforman lo que es conocido como el Observatorio Virtual (OV), una iniciativa internacional que busca promover el acceso y la investigación científica de forma pública, tanto para astrónomos como para las personas con interés en la astronomía.

Actualmente existen alrededor de 20 proyectos nacionales e internacionales que conforman el Observatorio Virtual, por lo que para mantener la consistencia, coherencia e interoperabilidad entre todos ellos es necesario que obedezcan estándares y protocolos en común. En este contexto es donde nace la Alianza Internacional de Observatorios Virtuales (IVOA), cuyo esfuerzo va dirigido precisamente a la definición de estándares y protocolos para el almacenamiento, descubrimiento y acceso de datos astronómicos. De esta forma, cuando un astrónomo quiera consultar los datos de algún objeto astronómico, por medio de un programa compatible con el Observatorio Virtual, este podrá encontrar, acceder y procesar los datos de una única forma, sin importar de que instalación u observatorio provengan.

Además, la creciente cantidad de proyectos astronómicos y la constante investigación que realizan los astrónomos, producirán en los próximos años una avalancha de datos de alto nivel, poniendo a prueba los campos de la astronomía y *Big Data*. Muchas veces la publicación de estos datos son un requerimiento en las investigaciones científicas y para esta tarea sería ideal usar la infraestructura del OV, ya que publicando datos astronómicos en él se estaría contribuyendo a esta iniciativa de carácter internacional. Sin embargo, no siempre los astrónomos tienen la *expertise* necesaria para realizar esta tarea y se ven con la obligación de solicitar a alguna otra entidad con experiencia en el proceso que realicen esta labor.

En particular, en el Observatorio Virtual Chileno (Solar et al., 2015), se realiza el proceso de la publicación de los datos con la ayuda de DaCHS (Demleitner et al., 2014), una *suite* orientada a implementar los protocolos de la IVOA para el acceso a sus datos. Aunque el uso de esta herramienta requiere vasta experiencia y conocimiento, además puede consumir una cantidad significativa de tiempo.

Por la razones anteriormente mencionadas es que en este proyecto se define una interfaz, en conjunto de un protocolo, con el fin de automatizar el proceso de ingestión de los datos de alto nivel generados por la comunidad de astrónomos en el Observatorio Virtual Chileno (ChiVO), y su posterior publicación por medio de DaCHS.

1.1. Objetivos

1.1.1. Objetivo General

Mejorar y automatizar el proceso de ingestión de datos astronómicos de alto nivel al Observatorio Virtual Chileno.

1.1.2. Objetivos Específicos

- Estudiar la documentación oficial de DaCHS y el proceso actual que se lleva a cabo en ChiVO, para la publicación de datos asociados al proyecto ALMA.
- Leer la documentación oficial de la IVOA, con énfasis en los modelos de datos de los objetos astronómicos.
- Definir y proponer una interfaz, en conjunto de un protocolo, para la ingestión de datos de alto nivel al Observatorio Virtual, y que además, automatice este proceso.
- Realizar un estudio de los costos para desarrollar la interfaz.
- Desarrollar e implementar un prototipo funcional de la interfaz, considerando sólo un subconjunto de los casos expuestos en la propuesta.

1.2. Estructura del Documento

En el capítulo 1 se describe el problema, su contexto y los objetivos de esta memoria. En el capítulo 2 se define el estado del arte de los diferentes tipos de datos usados en el ámbito de la astronomía, protocolos usados en el Observatorio Virtual (OV), herramientas relacionadas a la memoria, etc. En el capítulo 3 se muestran todos los detalles de la interfaz propuesta, explicando los pasos necesarios desde el momento en que un astrónomo ingresa al sistema, hasta que sus datos astronómicos son publicados en el Observatorio Virtual. En el capítulo 4 se detalla todo lo relacionado al estudio de la factibilidad de la propuesta, como la validación de la misma, esto incluye: el desarrollo del prototipo, requerimientos adicionales obtenidos después de realizar una validación con una astrónoma, la manera en que el sistema completo debe ser validado de forma sistemática y la estimación de los costos para el desarrollo completo de la interfaz definida en el capítulo 3. Finalmente, en el capítulo 5 se exponen todas las conclusiones del proceso completo de la memoria, que consideran tanto la factibilidad del proyecto, como los resultados obtenidos.

Capítulo 2 : Estado del Arte

En este capítulo, se describen los conceptos relacionados al desarrollo de la memoria. Partiendo desde lo general, como los tipos de datos comúnmente usados en el ámbito de la astronomía, hasta lo particular, en donde se describe el servicio web llamado VizeR y en el que actualmente se puede realizar la entrega de datos astronómicos para que posteriormente sean publicados al Observatorio Virtual.

2.1. Datos Astronómicos

Los datos astronómicos son datos obtenidos por instrumentos astronómicos, como telescopios u otros, que tienen relación con uno o varios cuerpos celestes o regiones del espacio.

2.1.1. Tipos de productos de datos

Los productos de datos hacen referencia a datos de alto nivel (ver sección 2.1.2) usados para análisis científico. Estos generalmente, son producidos aplicando varios procesos y filtrados sobre los datos de bajo nivel, que provienen directamente de los instrumentos astronómicos.

Los productos de datos relevantes para la memoria se muestran a continuación.

2.1.1.1. Catálogos

Los catálogos astronómicos son conjuntos de datos tabulados de objetos astronómicos (estrellas, nebulosas, galaxias, planetas, etc.) y comúnmente categorizados.

Generalmente, los catálogos poseen datos básicos de los objetos de estudio, como la posición, tamaño, velocidad radial, etc., pero también datos más procesados, resultantes de la aplicación de uno o varios modelos matemáticos.

Los catálogos se encuentran en diferentes formatos de archivos, pero regularmente están en formato FITS (ver sección 2.1.3.1), tablas ASCII o VOTable (ver sección 2.1.3.2).

2.1.1.2. Imágenes

Una imagen, en el contexto de las observaciones astronómicas, es una representación matricial de dos dimensiones que simboliza una región u objeto en el espacio. Estas imágenes pueden ser tomadas para obtener representaciones reales de los objetos, es decir, en el cual se reflejen los colores que vería el ojo humano, pero también se pueden aplicar filtros, asignándole colores a ondas electromagnéticas invisibles a simple vista, para así poder inferir propiedades físicas de los objetos.

En la astronomía generalmente las imágenes se guardan en formato FITS y son tomados con detectores electrónicos como un CCD (Dispositivo de carga acoplada o *Charge Coupled Device* en inglés), que son sensores con diminutas células fotoeléctricas que registran la imagen (Hubble European Space Agency Information Centre, 2009).

2.1.1.3. Espectros

Los espectros en la astronomía son tipos de datos que representan generalmente, la intensidad lumínica o energía, respecto del tiempo o longitud de onda de algún objeto en estudio. Con esta información se puede averiguar la composición química, temperatura y otras propiedades físicas del

objeto en cuestión, pero también puede ser usado para la detección de planetas o *quasars* (Schwartz et al., 2012).

Comúnmente los espectros se almacenan en archivos de formato FITS, formatos de tablas ASCII o VOTable.

2.1.2. Niveles de calibración

Para diferenciar datos de alto nivel con datos de bajo nivel, en la IVOA existe el concepto de nivel de procesamiento/calibración (IVOA, 2017). Estos niveles van del 0 al 4 y se definen a continuación:

- **Nivel de calibración 0**: Datos instrumentales sin procesar.
- **Nivel de calibración 1**: Datos instrumentales en un formato estándar (FITS, VOTable, etc.).
- **Nivel de calibración 2**: Datos procesados y preparados para hacer análisis científico.
- **Nivel de calibración 3**: Datos altamente procesados y resultantes de la combinación de datos de múltiples fuentes.
- **Nivel de calibración 4**: Datos generados después de aplicar métodos matemáticos o análisis científico.

Los datos de alto nivel, generalmente hacen referencia a datos con un nivel de calibración mayor que 2.

2.1.3. Formatos de archivos comunes

A continuación, se presentarán algunos de los formatos de archivos que son usados comúnmente en el ámbito de la astronomía, y que además, son relevantes para la memoria.

2.1.3.1. FITS

Flexible Image Transport System es el formato de datos estándar utilizado en el campo de la astronomía, aprobado por la NASA y la Unión Astronómica Internacional (IAU). Tiene orígenes en los años 70 y posee muchas otras características que las de un formato de imagen de uso común, como JPEG o PNG, ya que fue desarrollado con la idea de que sería utilizado para el transporte, análisis y almacenamiento de datos científicos (IAU FITS Working Group, 2017).

Cabe destacar que el término Image (del acrónimo FITS) no se aplica de forma rigurosa, ya que la mayoría de las veces los archivos FITS son usados para guardar datos en términos generales y no únicamente imágenes, como por ejemplo, espectros (arreglos de una dimensión) o cubos de datos (arreglos de 3 o más dimensiones).

En términos de estructura, el formato FITS consiste en uno o más HDUs (*Header + Data Units*), donde en el *Data Unit* se almacenan los datos y en el Header se guardan metadatos. El primer HDU es llamado *primary* HDU y a los siguientes se les denomina FITS *extensions* (ver figura 2.1). Asimismo, en un archivo FITS, la unidad de datos en cada HDU es opcional, por lo que perfectamente un archivo FITS puede tener la primera unidad de datos vacía y tener únicamente datos en las extensiones.

Una de las grandes fortalezas de este formato, es el hecho de contar con metadatos en los *Headers*, como por ejemplo, los tipos de datos almacenados, las coordenadas y el observatorio de donde se obtuvieron, entre otros, haciéndola una poderosa herramienta para almacenar datos de observaciones astronómicas, y es por esto que es ampliamente usada por la comunidad de astrónomos.

Según el estándar, las FITS extensions pueden representar 3 tipos de datos en sus *data units*:

- IMAGE: Arreglos de datos.
- TABLE: Datos tabulares en formato ASCII.
- BINTABLE: Datos tabulares en formato binario (más eficiente en términos de memoria).

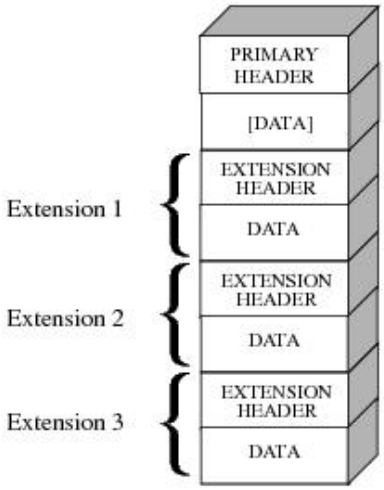


Figura 2.1: Estructura de un archivo de formato FITS.
Fuente: (2011). Retrieved from http://www.stsci.edu/hst/HST_overview/documents/datahandbook/intro_ch23.html

Además, existen distintas librerías para el soporte de este formato en la mayoría de los lenguajes de programación utilizados en el ámbito científico, como: C, Fortran, Java, Perl y Python.

2.1.3.2. VOTable

Es un formato de archivo en XML desarrollado por la IVOA que busca proveer una forma interoperable de almacenar y acceder a datos astronómicos, el cual fue diseñado para ser lo más parecido a un archivo FITS con extensiones del tipo BINTABLE (IVOA, 2013).

Básicamente la estructura de un VOTable consta de Metadatos (parámetros, descripciones, campos, información, etc.) y una lista de tablas, que consisten de un conjunto de campos, más una tabla de datos. A su vez, en cada celda de esta tabla de datos puede ir un dato primitivo o un arreglo (unidimensional o multidimensional) de datos primitivos.

La ventaja de este formato respecto a FITS, es que al tratarse de un archivo XML no necesita herramientas tan especializadas para su manejo, además de que no tiene una semántica tan compleja y soporta el uso de caracteres Unicode.

En el código 1, se muestra un pequeño fragmento de un archivo descargado desde el servicio de

VizieR (más información de este servicio en la sección 2.4.2) en formato VOTable, el cual tiene una tabla con la temperatura de 4 estrellas.

Código 1: VOTable

```
<INFO ID="MaxTuples" name="-out.max" value="1"/>
  <INFO name="CatalogsExamined" value="2"/>

<RESOURCE ID="yCat_16891031" name="J/ApJ/689/1031">
  <DESCRIPTION>Heavy element in giant stars</DESCRIPTION>
  <COOSYS ID="J2000" system="eq_FK5" equinox="J2000"/>
  <TABLE ID="J_ApJ_689_1_table1" name="J/ApJ/689/1/table1">
    <DESCRIPTION>Stellar parameters Yong et al</DESCRIPTION>

    <FIELD name="Name" ucd="meta.id;meta.main" datatype="char" arraysize="8">
      <DESCRIPTION>Star designation</DESCRIPTION>
    </FIELD>
    <FIELD name="Teff" ucd="phys.temperature.effective" datatype="short" unit="K">
      <DESCRIPTION>Effective temperature (1)</DESCRIPTION>
    </FIELD>

    <DATA>
      <TABLEDATA>
        <TR><TD>M4 L1411</TD><TD>4025</TD></TR>
        <TR><TD>M4 L1501</TD><TD>4175</TD></TR>
        <TR><TD>M4 L1514</TD><TD>3950</TD></TR>
        <TR><TD>M4 L2307</TD><TD>4125</TD></TR>
      </TABLEDATA>
    </DATA>
  </TABLE>
</RESOURCE>
```

2.1.3.3. CSV (Comma-separated values)

Corresponde a un formato de archivo ASCII (archivo de texto), que generalmente es usado para almacenar datos de tipo tabular, en donde cada línea representa un registro de la tabla y las comas marcan la separación entre distintas columnas. Es importante mencionar que, aunque este formato sea ampliamente usado, no existe una estandarización oficial. Sin embargo, las implementaciones del formato CSV generalmente cumplen con varias reglas por convenciones establecidas (Shafranovich, 2005). Particularmente en el ámbito de la astronomía, es común que se usen varias líneas en la parte superior del archivo para describir a los datos.

2.1.3.4. TSV

Es un formato de archivo análogo al CSV, pero en este caso los valores están separados por un *tab* en vez de una coma.

En la figura 2.2 se muestra, como ejemplo, parte del contenido del catálogo JA+A587A107 descargado en formato TSV, desde el servicio VizieR.

```
#Name: J/A+A/587/A107
#Title: Transitions between Ni II bound states (Cassidy+, 2016)
#Table  J_A_A_587_A107_table4:
#Name: J/A+A/587/A107/table4
#Title: E1 transitions in NiII
#Column Lo  (a23)   Spectroscopic designation of lower level    [ucd=phys.atmol.qn]
#Column Up  (a23)   Spectroscopic designation of upper level    [ucd=phys.atmol.qn]
Lo  Up

---------------------- ----------------------
3d8(3P)4s 4P0.5         3d8(3F)4p 4D0.5
3d8(3P)4s 4P0.5         3d8(3P)4p 4P0.5
3d8(3P)4s 4P0.5         3d8(1D)4p 2P0.5
3d8(3P)4s 4P0.5         3d8(3P)4p 4D0.5
3d8(3P)4s 4P0.5         3d8(3P)4p 2P0.5
3d8(3P)4s 4P0.5         3d8(3P)4p 2S0.5
3d8(3P)4s 4P0.5         3d7(4F) 4s4p(3P) 6F0.5
3d8(3P)4s 4P0.5         3d7(4F) 4s4p(3P) 6D0.5
3d8(3P)4s 4P0.5         3d7(4F) 4s4p(3P) 4D0.5
3d8(3P)4s 4P0.5         3d8(1S)4p 2P0.5
3d8(3P)4s 4P0.5         3d7(2P) 4s4p(3P) 4P0.5
3d8(3P)4s 4P0.5         3d7(4P) 4s4p(3P) 6D0.5
3d8(3P)4s 4P0.5         3d7(4P) 4s4p(3P) 4D0.5
3d8(3P)4s 4P0.5         3d7(2P) 4s4p(3P) 4D0.5
3d8(3P)4s 4P0.5         3d7(4P) 4s4p(3P) 2S0.5
3d8(3P)4s 4P0.5         3d7(23D) 4s4p(3P) 4D0.5
3d8(3P)4s 4P0.5         3d7(4P) 4s4p(3P) 4P0.5
3d8(3P)4s 4P0.5         3d7(2P) 4s4p(3P) 2S0.5
3d8(3P)4s 4P0.5         3d7(2P) 4s4p(3P) 2P0.5
3d8(3P)4s 4P0.5         3d7(4P) 4s4p(3P) 2P0.5
```

Figura 2.2: Parte de la tabla 4 del catálogo JA+A587A107 descargado en formato TSV.
Fuente: Elaboración propia.

2.2. Observatorio Virtual

A continuación, se definen los conceptos del Observatorio Virtual y la IVOA, es esencial entender estas nociones, ya que tienen una relación directa con los datos de los astrónomos, considerando que estos quedarán disponibles en el OV, una vez sean publicadas.

2.2.1. Cómo se conforma el OV

El Observatorio Virtual (OV) es toda una infraestructura desarrollada con el fin de otorgarle, a todo aquel que lo desee, acceso libre a sus datos astronómicos almacenados en varios centros de datos repartidos por todo el mundo. Promoviendo el análisis e intercambio científico dentro de la comunidad con interés en la astronomía.

Esta infraestructura también considera los protocolos y estándares acordados por la IVOA, que hacen posible la uniformidad para el acceso de los datos y el desarrollo de programas compatibles con el OV, que proveen una interfaz gráfica para acceder y analizar los datos astronómicos, almacenados en el Observatorio Virtual.

En resumen, el OV es un gran centro de datos descentralizado, incluyendo los protocolos, estándares y programas, usados para hacer posible el almacenamiento, intercambio y análisis de datos astronómicos.

El OV está conformado por los proyectos listados a continuación (IVOA, 2014):

- Argentine Virtual Observatory: http://nova.conicet.gov.ar
- Armenian Virtual Observatory: https://www.aras.am//Arvo/arvo.htm
- AstroGrid, United Kingdom: http://www.astrogrid.org
- Australian Virtual Observatory: http://aus-vo.org.au

- Brazilian Virtual Observatory: http://bravo.iag.usp.br
- Chinese Virtual Observatory: http://www.china-vo.org
- Canadian Virtual Observatory: http://www.cadc-ccda.hia-iha.nrc-cnrc.gc.ca/cvo
- Chilean Virtual Observatory: https://www.chivo.cl
- European Space Agency: https://www.cosmos.esa.int/web/esdc
- European Virtual Observatory: http://www.euro-vo.org
- German Astrophysical Virtual Observatory: http://www.g-vo.org
- Hungarian Virtual Observatory: http://hvo.elte.hu/en
- Japanese Virtual Observatory: http://jvo.nao.ac.jp
- Observatoire Virtuel France: http://www.france-vo.org
- Russian Virtual Observatory: http://www.inasan.rssi.ru/eng/rvo
- South African Astroinformatics Alliance: https://www.sa3.ac.za
- Spanish Virtual Observatory: http://svo.cab.inta-csic.es/main/index.php
- Italian Virtual Observatory: http://vobs.astro.it
- Ukrainian Virtual Observatory: http://www.ukr-vo.org
- US Virtual Observatory Alliance: http://usvoa.cfa.harvard.edu
- Virtual Observatory India: http://voi.iucaa.in/voi

2.2.2. IVOA

La IVOA es una organización formada en el año 2002, que busca promover la interoperabilidad entre los diferentes proyectos de Observatorios Virtuales. Para este fin, este organismo internacional, ha definido grupos de trabajo que desarrollan protocolos para la comunicación de los datos astronómicos (Peter et al., 2004).

Algunos de estos protocolos son:

- **ADQL**: Lenguaje de consultas basado en SQL, desarrollado para realizar *queries* a datos astronómicos en los distintos servicios del Observatorio Virtual (IVOA, 2018a).
- **TAP**: Protocolo orientado al acceso de datos tabulares, por medio de consultas ADQL de forma obligatoria, o por medio de otro lenguaje de consultas de forma opcional (IVOA, 2018b).
- **SCS**: Protocolo que define el acceso a los registros de un catálogo, por medio de una consulta que describe un cono en el cielo (IVOA, 2018c).
- **ObsCore**: Modelo de datos usado para describir los metadatos, que especifican las características de la mayoría de los productos de datos publicados en el Observatorio Virtual (IVOA, 2017).
- **SSAP**: Protocolo de la IVOA que define una manera homogénea de descubrir y acceder a los datos espectrales unidimensionales, provenientes del Observatorio Virtual (IVOA, 2011).
- **SIAP**: Protocolo que busca proveer una forma de descubrir y acceder a los datos de imágenes de 2 dimensiones o más (cubos de datos), provenientes del Observatorio Virtual (IVOA, 2015).

2.2.3. ChiVO

"El Observatorio Virtual Chileno (ChiVO) es un OV desarrollado en Chile y es uno de los 20 proyectos del OV actualmente en curso en el mundo. Nació de la necesidad de archivar datos que

requieren grandes capacidades de almacenamiento y el desarrollo de nuevas herramientas para el análisis de grandes volúmenes de datos. Esto debido a los volúmenes de datos a gran escala que generan los observatorios astronómicos en Chile, principalmente el proyecto ALMA que genera más de 1TB de datos por día." (ChiVO, 2018).

DaCHS, actualmente utilizado en ChiVO para la publicación de datos del proyecto ALMA en el Observatorio Virtual, es un elemento de suma importancia para la interfaz propuesta en esta memoria, ya que la intención es que también se utilice en la publicación de los datos astronómicos de alto nivel.

2.3. Software Astronómico

A continuación, se muestra algunos de los *softwares* comúnmente usados en el ámbito de la astronomía.

2.3.1. Lectura y manejo de datos

Para la manipulación de datos existe el paquete de Python Astropy, el cual está enfocado a proveer las funcionalidades que normalmente un astrónomo utiliza al realizar esta tarea. La creación de este paquete tuvo como motivo, el desarrollar una librería única que reuniera las funcionalidades de los diferentes paquetes relacionados a la astronomía (desarrollados de forma separada), y que a la fecha tiene más de 220 contribuidores y 21.000 *commits* en su repositorio de GitHub (Astropy Collaboration, 2018).

Las funciones que tiene son: manejo de WCS [1], manejo de archivos de formatos específicos usados en el dominio de la astronomía, conversión de unidades físicas comunes al dominio, entre otros (Collaboration et al., 2013).

Además de Astropy, existen varios paquetes más para el manejo de archivos, por ejemplo, para

[1] World Coordinate system.

el manejo de archivos en formato FITS, VOTable, y otros, existe STIL en el lenguaje Java (Taylor, 2018), mientras que para el manejo de archivos FITS en C y Fortran existe CFITSIO (HEASARC, 2018).

2.3.2. Descubrimiento de datos

Las herramientas para el descubrimiento de datos, tienen la función de descubrir y acceder a los datos disponibles en el Observatorio Virtual, de esta forma, el usuario puede hacer uso de estas herramientas para obtener los datos de su interés.

Actualmente existen decenas de programas compatibles con el Observatorio Virtual para el descubrimiento de datos astronómicos. A continuación, se describirán Aladin y TOPCAT, los cuales son claros ejemplos de éstos.

2.3.2.1. Aladin

Desarrollado en el lenguaje de programación Java y publicado en 1999 por el CDS (Centre de Données astronomiques de Strasbourg). Esta herramienta se encuentra disponible en Windows, Linux y Macintosh, y además es de código abierto (Bonnarel et al., 2000).

Aladin tiene varias funcionalidades, tales como:

- Buscar y/o visualizar espectros, catálogos e imágenes astronómicas digitalizadas en base a parámetros, como el nombre de un objeto astronómico o sus coordenadas.
- *Script mode* o *command line mode* para tener mayor control de Aladin.
- Soportar mensajes SAMP para intercambiar datos con otros programas o aplicaciones web.

Además, las búsquedas se pueden realizar en todo el Observatorio Virtual, pero también de forma segregada en una gran cantidad de *data centers*, como en la ESA, ESO, MAST, CDS, NED,

GAVO, etc. De igual forma permite que el usuario use archivos que se encuentren en su computador (ver figura 2.3).

Por otro lado, también soporta un gran número de formatos de archivos, por ejemplo:

- JPEG, GIF, PNG, PDS para imágenes.
- FITS para imágenes y datos tabulares.
- VOTable, CSV, TSV y en general cualquier archivo de texto con columnas y delimitadores para datos tabulares.

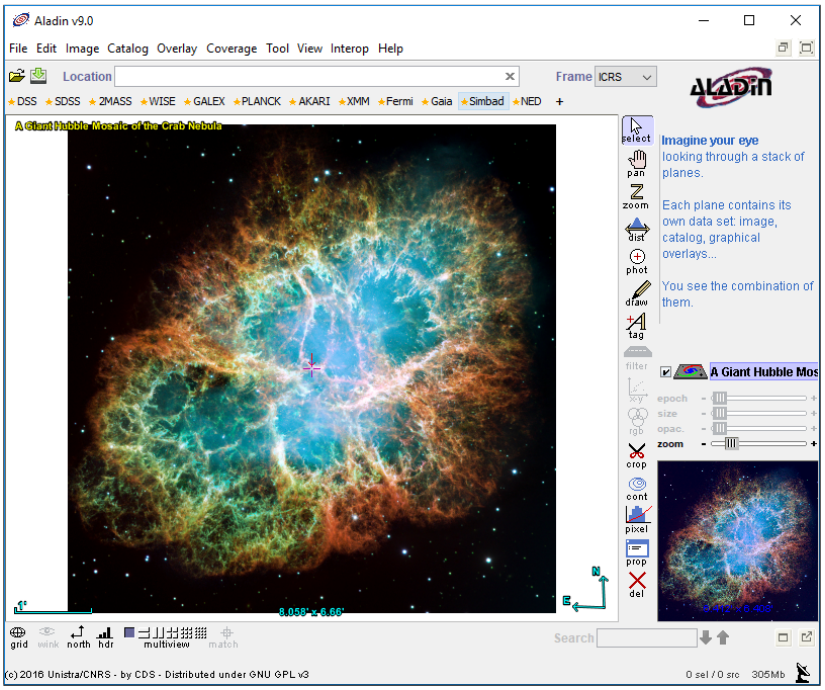


Figura 2.3: Imagen de la Nebulosa del Cangrejo visualizada desde Aladin.
Fuente: Elaboración propia.

Aladin igualmente permite que sus usuarios desarrollen sus propios *plugins* para agregar y extender las funcionalidades que deseen. Varios han sido creados con variadas funciones como rotar y editar imágenes, realizar seguimiento del *mouse*, etc.

2.3.2.2. TOPCAT

Visualizador de datos tabulares, programado en Java y principalmente en el Reino Unido bajo concesiones del *Science and Technology Facilities Council* y el *Particle Physics and Astronomy Research Council* (Taylor, 2014).

Esta aplicación incorpora gran parte de las funcionalidades que los astrónomos usan al analizar datos astronómicos, adicionalmente es compatible con varios formatos de archivos como FITS y VOTable. Al igual que Aladin, esta herramienta permite acceder a servicios compatibles con el Observatorio Virtual.

Por último, TOPCAT permite realizar varias operaciones, como el cálculo de datos estadísticos, visualización de gráficos para un estudio más detallado de los datos obtenidos, comunicarse con otras aplicaciones por medio del protocolo SAMP, editar y exportar tablas en múltiples formatos, etc.

2.3.3. Ingestión y publicación de datos

Para la publicación de datos existe una variedad de herramientas, pero una de las más completas es DaCHS. Este programa fue desarrollado en Python por Markus Demleitner del GAVO (GAVO, 2018) y está orientada a implementar la capa del acceso a los datos de un Observatorio Virtual. Tiene soporte para la ingestión y mapeo de varios tipos de datos, manejo de metadatos, etc., además de montar su propio servidor web para interactuar con los datos ingresados (Demleitner, 2018b).

Respecto a la jerarquía de los archivos y directorios usados para la ingestión de datos en DaCHS, a cada conjunto de datos publicado le corresponde un *resource directory*. Este es el directorio en donde se guardará tanto el *resource descriptor* (RD) como los datos a ser publicados. Tener en cuenta que el RD es un archivo en XML, en el cual se define todo el proceso de la ingestión y publicación de datos, partiendo desde la definición de los metadatos, hasta la declaración de las columnas y protocolos del OV a utilizar.

Se debe tener en cuenta que el uso de DaCHS no es tan sencillo, dependiendo de la complejidad del conjunto de datos que se desee publicar, puede ser necesario una cantidad significativa de trabajo escribiendo el *resource descriptor* correspondiente. Además, la documentación de DaCHS no es completa y acabada, por ejemplo, sólo se cuenta con un par de tutoriales ejemplificando el proceso de la publicación de datos, pero de casos muy particulares, a parte de las definiciones de los diferentes elementos que puede contener un *resource descriptor*.

Otras herramientas que son usadas para publicar datos al OV son: VO-Dance (INAF, 2014), SVOCat (SVO, 2016), entre otros.

2.4. Ingestión de Datos vía Web

Contar con interfaces web para realizar tareas como la ingestión de datos astronómicos, puede ser de gran ayuda para el punto de vista de un astrónomo, ya que estos comúnmente no tienen la *expertise* necesaria para realizar directamente la publicación de los datos en el OV y deben hacer entrega de estos a una iniciativa del Observatorio Virtual, para que finalmente sea esta la que publique sus datos. Pero hacerlo de una forma estandarizada también es importante, debido a que así el personal técnico que recibe estos datos astronómicos pueden ahorrar una gran cantidad de tiempo automatizando este proceso.

Es por esta razón que a continuación, se muestran herramientas de desarrollo que pueden ayudar en el desarrollo de la interfaz, además de un servicio web en el que actualmente se puede hacer entrega de datos astronómicos para que posteriormente sean publicados, el cual puede servir como referencia para la memoria.

2.4.1. Frameworks para el desarrollo de aplicaciones web

Un *framework* es un concepto abstracto que hace referencia a un *software* que proporciona una funcionalidad genérica, la cual puede ser extendida por el desarrollador, es decir, un entorno

de desarrollo reutilizable que facilita el desarrollo de aplicaciones, implementando una estructura estándar de desarrollo (Reza and Mardiyanto, 2014).

A continuación, se listarán algunos *frameworks* que permiten crear aplicaciones web de forma rápida y ordenadamente.

2.4.1.1. Rails

Es un *framework* de código abierto creado por David Heinemeier Hansso y escrito en el lenguaje Ruby, el cual facilita el desarrollo de aplicaciones web siguiendo el modelo MVC, que busca estructurar un sistema en 3 componentes:

- Vista: Parte visual que tiene interacción directa con el usuario.
- Controlador: Controla el flujo de datos entre la vista y el modelo.
- Modelo: Define el modelo de los datos y la forma en que estos son accedidos, además de las reglas de negocio.

Por medio de pocos comandos, Rails permite realizar tareas comunes a la hora de desarrollar una aplicación web, como crear bases de datos, generar código HTML, generar interfaces para realizar operaciones CRUD sobre recursos, etc.

En la figura 2.4 se puede observar que Rails tuvo un aumento de popularidad en las búsquedas realizadas por Google durante el año 2005, alcanzando un *peak* entre los años 2006 y 2007. Esta popularidad se ha mantenido relativamente constante desde el año 2010.

Rails ha sido usado desde pequeñas compañías, hasta grandes firmas, tales como: GitHub [2], Twitter [3] y Groupon [4] (Hartl, 2012).

[2] https://github.com
[3] https://twitter.com
[4] https://www.groupon.com

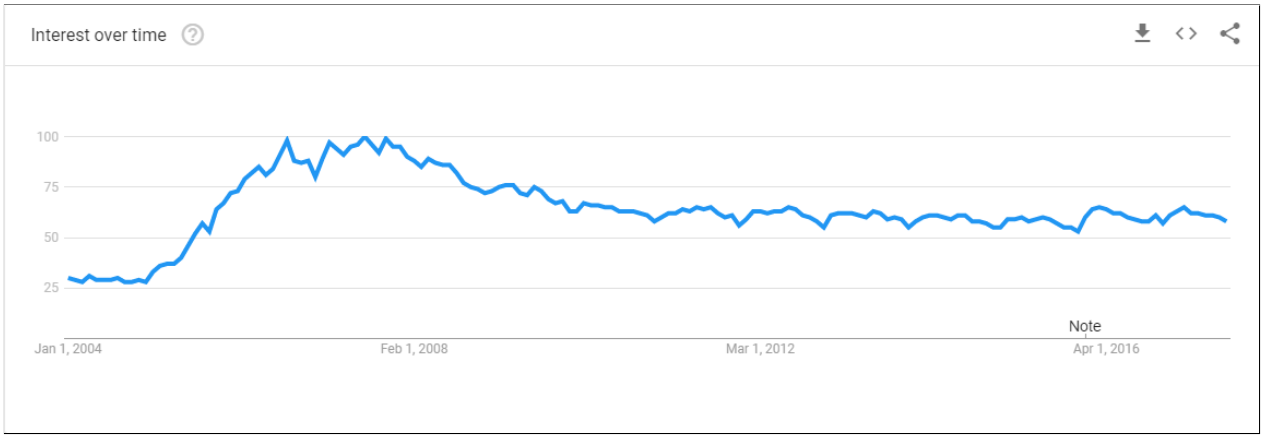


Figura 2.4: Trafico de Google Trends al buscar el termino rails.
Fuente: Elaboración propia.

2.4.1.2. Django

Al igual que Rails, Django es un *framework* de código abierto liberado en el año 2005 con el fin de poder desarrollar aplicaciones web que obedezcan el modelo MVC de forma rápida. Django fue creado por el diario Lawrence Journal-World y posteriormente su mantención fue transferida a la Django Software Foundation (DSF, 2018).

Las principales características que el proyecto DSF declara que Django posee son:

- **Desarrollo ágil**: Se provee un conjunto de comandos para la creación de un proyecto, configuración y ejecución de un servidor web, la creación de vistas y un conjunto de elementos para el desarrollo rápido de una aplicación web.
- **Seguridad**: Tiene características que facilitan la protección respecto a los principales tipos de ataques, tales como: XSS[5], CSRF [6] y SQL *Injection* [7].
- **Escalabilidad**: Django está desarrollado con la idea de que aproveche todo el *hardware* que se tenga disponible, es decir, servidores web, bases de datos, etc., sin mayores cambios en el código.

[5] Ataque en el que se inyecta un *script* en el contenido de un sitio web.
[6] Ataque que provoca que un usuario ejecute acciones indeseadas en un sitio web en la que está autenticado.
[7] Ataque en el que se ejecutan comandos SQL por medio del *input* de una aplicación web.

Django ha sido usado en el desarrollo de varias páginas web, tales como: Instagram [8], Pinterest [9] y Bitbucket [10].

2.4.2. VizieR

VizieR se empezó a desarrollar el año 1993 y ha sufrido cambios significativos desde entonces, actualmente guarda más de 16.000 catálogos y permite el acceso de estos públicamente.

Este servicio web es un referente para esta memoria, ya que permite la publicación de datos de alto nivel al Observatorio Virtual relacionados a algún artículo o *paper* [11]. El proceso a seguir para el envío de datos tabulares y la publicación de estos datos en el observatorio virtual se puede ver gráficamente en el BPMN de la figura 2.5, que se realizó de acuerdo a la documentación provista por VizieR.

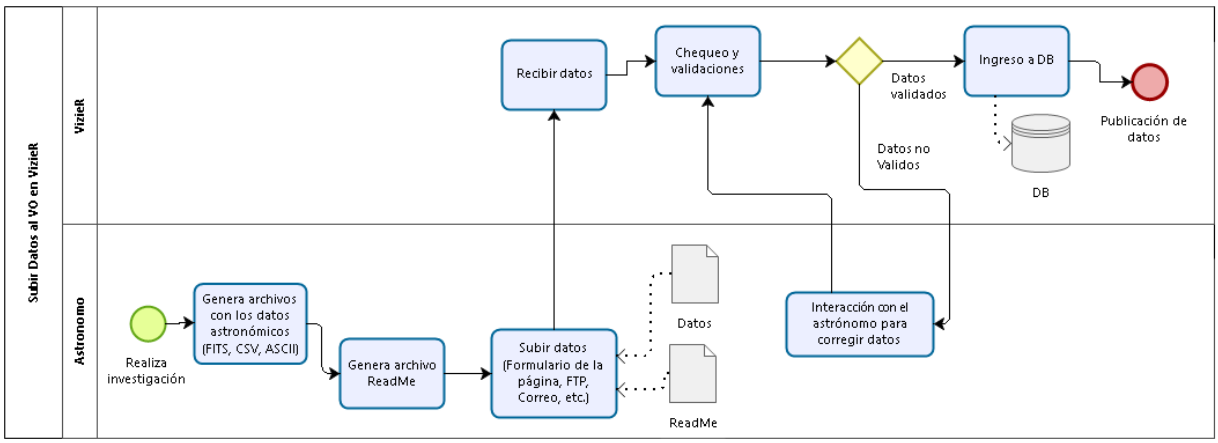


Figura 2.5: Proceso de subida de datos a VizierR.
Fuente: Elaboración propia.

Lo que le compete al astrónomo son las etapas de la preparación de los archivos (que contienen los datos astronómicos), la preparación del archivo ReadMe (donde se encuentran los metadatos que

[8] https://www.instagram.com/
[9] https://www.pinterest.com/
[10] https://bitbucket.org
[11] En el enlace http://vizier.u-strasbg.fr/vizier/submit.htx se describe el proceso de la publicación de datos en VizieR.

describen a los datos astronómicos) y la subida de los archivos.

En VizieR también se realizan procedimientos de verificación de compatibilidad entre los datos y los metadatos, lo cual puede generar interacciones futuras con el astrónomo, en el caso de encontrar inconsistencias antes de que sus datos sean publicados.

En la siguiente sección se realizará una descripción más detallada del proceso de publicación de datos en esta interfaz web.

2.4.2.1. Proceso de ingestión y publicación de datos por VizieR

Se espera que cada archivo a subir represente un catálogo, un espectro o una imagen, es decir, debe ser un conjunto de datos de alto nivel. El formato del archivo dependerá del tipo de producto de dato:

- Para las **tablas** o **catálogos** se permiten los formatos FITS, TSV, CSV o latex.
- Para los **espectros** se permite únicamente el formato FITS.
- Para las **imágenes** se permite únicamente el formato FITS.

Además, se pide que en este proceso, los archivos a ser subidos cumplan con el estándar ISO 9660 (ISO, 1988), o sea, que los nombres de los archivos tengan un máximo de 8 caracteres, seguido por un punto y 3 caracteres como máximo después del punto especificando la extensión.

2.4.2.1.1 Preparación del archivo ReadMe

En el proceso para publicar datos al Observatorio Virtual desde VizieR, se pide la construcción de un archivo ReadMe. Este archivo tendrá la finalidad de describir los datos a subir de la manera menos ambigua posible. Además, deberá tener los siguientes contenidos:

- Título abreviado, el primer autor y el año de la publicación asociada a los datos astronómicos que se desean publicar (preferiblemente en menos de 80 caracteres).
- Título completo y todos los autores de la publicación, además de la referencia bibliográfica.
- ***Keywords*:** Palabras claves o *keywords* que describan el contenido de los archivos a ser subidos.
- ***Description*:** Una descripción del contexto en que se realiza la observación y se obtienen los datos, como por ejemplo, que instrumento se utiliza, en que condiciones se produce la observación, etc.
- ***File Summary*:** En esta sección especifica un pequeño título, cual es el largo de la línea más larga y el número de registros o entradas por cada archivo.
- ***Byte-by-byte Description of file*:** Describe la estructura de cada uno de los archivos a subir. Esta descripción se hace en una forma tabular, en donde cada fila describe uno o varios campos (columna) del archivo de datos, como la unidad de medida utilizada, el formato (entero, carácter, etc.), una etiqueta y una explicación.

2.4.2.1.2 Subida de datos

Después de la preparación de los archivos y la creación del archivo ReadMe, será necesario la entrega de estos por alguno de los siguientes medios:

- En el caso de que los datos astronómicos estén en archivos de formato ASCII, se pueden subir usando el formulario en la página de VizieR [12].
- Alternativamente en el caso de tener archivos binarios, se pueden subir los datos vía FTP [13].
- Otra opción para archivos no muy pesados es vía *e-mail* [14].

[12] Link del formulario en VizieR: http://cdsarc.u-strasbg.fr/viz-bin/Submit.
[13] IP del servidor FTP para subir los datos: 130.79.128.5.
[14] E-mail para hacer entrega de los datos astronómicos: cats@cdsarc.u-strasbg.fr.

- Por último, también se puede contactar con VizieR para coordinar con ellos la entrega de los datos por otros medios, como por ejemplo, entregar los archivos físicamente dejando un CD o DVD.

En caso de que se encuentre algún tipo de inconsistencia o incoherencia en los datos, será necesario realizar interacciones con los autores para resolver estos errores, deteniendo momentáneamente el proceso.

Capítulo 3 : Propuesta de incorporación de datos de alto nivel a ChiVO

En este capítulo, se detalla el diseño de la interfaz en conjunto del protocolo. La interfaz hace referencia al medio que tienen tanto el astrónomo como el administrador para interactuar con la aplicación web. En otras palabras, sus diferentes vistas, elementos gráficos y contenido. Mientras que el protocolo para la ingestión de los datos alude al conjunto de reglas establecidas para que el astrónomo pueda ingresar sus datos al sistema y que estos sean publicados, es decir, qué tipos de datos y formatos de archivos son aceptados, cuales son y en qué consisten los pasos en el proceso de ingestión de los datos, etc.

Uno de los objetivos de la interfaz propuesta es que las ingestión de los datos sea de forma fácil e intuitiva. Esto significa desarrollar un sistema que tenga *widgets* [15] y elementos gráficos que permitan una realización rápida del proceso, en vez de que el astrónomo tenga que crear un archivo de configuración, como es el caso que se vio en el capítulo anterior, donde el astrónomo tenia que crear un archivo ReadMe para la ingestión de datos a VizieR. Este es un punto importante, ya que incluso un astrónomo (en el caso de no poseer una forma amena o cautivadora para publicar los datos producto de su esfuerzo) podría no realizar esta labor si es que no es un requerimiento u obligación.

Por otro lado, el protocolo definido también significa un ahorro de tiempo para los administrado-

[15] Elementos que permiten una interacción con el usuario, como los elementos gráficos que se ven comúnmente en los formularios web.

res que realicen el proceso de publicación, ya que el sistema planteado debiese realizar de forma automática varios procesos que, de otra forma, se harían de forma manual. De esta manera se le dará un valor agregado a la interfaz, ya que tanto los astrónomos como el personal técnico de ChiVO se verán beneficiados de este sistema, gastando menos recursos para realizar esta actividad.

A continuación, se describen los diferentes aspectos de esta interfaz.

3.1. Tipos de Usuarios

Los dos tipos de usuarios que interactuarán con el sistema:

- **Astrónomo**: Es el individuo que utilizará el sistema propuesto con el fin de subir sus datos astronómicos (resultantes de una investigación científica) a ChiVO.
- **Administrador**: Es la persona encargada de que los datos subidos a ChiVO sean publicados en el Observatorio Virtual.

3.2. Tipos de Datos Aceptables

Se contemplan cuatro formatos de archivos para la publicación de datos. Dependiendo del tipo de producto de dato, estos son:

- FITS y VOTable en el caso de subir un **espectro**.
- FITS en el caso de subir una **imagen**.
- FITS, CSV, TSV y VOTable en el caso de subir un **catálogo**.

El objetivo final de la interfaz es que sea útil para los astrónomos que necesiten publicar datos producidos en sus propias investigaciones, los cuales pueden contener datos ya publicados en el OV, esto implica que es espera que los datos ingresados tengan un nivel de calibración 3 o superior.

3.3. Resumen del Protocolo

De manera introductoria, se realiza una definición a grandes rasgos de cada uno de los pasos del protocolo propuesto.

1. El astrónomo ingresa su correo para comenzar el proceso de publicación de datos. En ese instante se genera una URL que le permitirá continuar posteriormente con este proceso y se le envía al correo ingresado (ver figura 3.1).

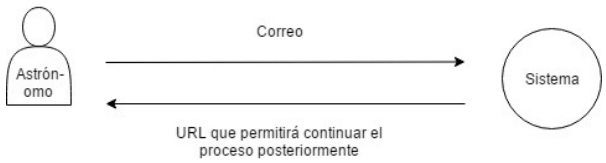


Figura 3.1: Ingreso de correo para iniciar proceso de publicación.
Fuente: Elaboración propia.

2. El astrónomo ingresa un título y una descripción para el conjunto de datos, además de los archivos con los datos astronómico (ver figura 3.2).

3. El astrónomo a continuación debe ingresar metadatos de la observación astronómica, como el nombre del instrumento utilizado, autores, etc.

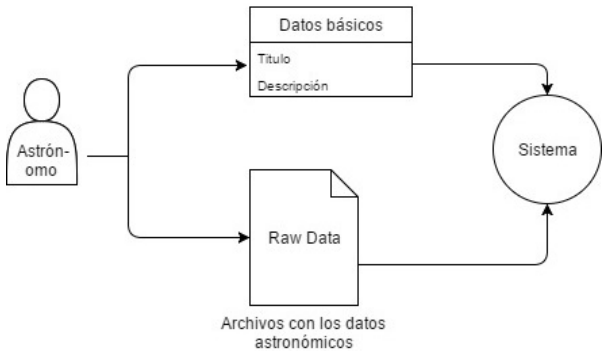


Figura 3.2: Ingreso de los datos básicos y archivos con los datos.
Fuente: Elaboración propia.

4. En seguida el astrónomo debe especificar que datos dentro de los archivos desea que sean publicados en el Observatorio Virtual. También deberá escoger que servicios/protocolos querrá proveer (SCS, ADQL, ObsCore, etc.) en cada uno de los datos (ver figura 3.3).

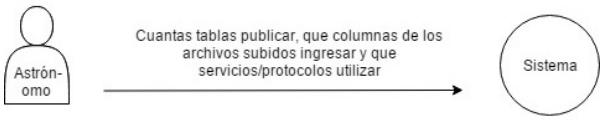


Figura 3.3: Selección de los datos a publicar.
Fuente: Elaboración propia.

5. El administrador verifica los datos y aprueba la petición de publicación de datos en el sistema en caso de no encontrar inconsistencias.

6. El administrador le pide al sistema que genere el *resource descriptor* de los datos subidos.

7. El administrador verifica que el *resource descriptor* de DaCHS esté correctamente construido, realizando correcciones de ser necesario y posteriormente publica los datos de la observación astronómica al Observatorio Virtual (ver figura 3.4).

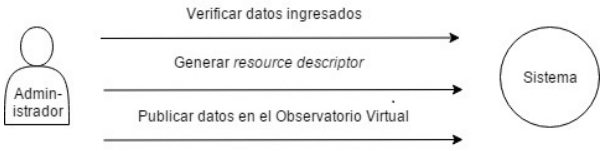


Figura 3.4: Administrador verifica los datos, genera el *resource descriptor* asociado y publica los datos en el observatorio virtual.
Fuente: Elaboración propia.

En base a este protocolo se construyó el BPMN de la figura 3.5 que explica el proceso, tanto desde el punto de vista del astrónomo, como del administrador del sistema.

Cabe destacar que la formulación de los pasos del protocolo se realizó después de estudiar la forma en que actualmente se realizan las publicaciones de los datos del proyecto ALMA por ChiVO y de observar el protocolo de ingreso de datos a VizieR (ver sección 2.4.2.1). Además, se recibieron

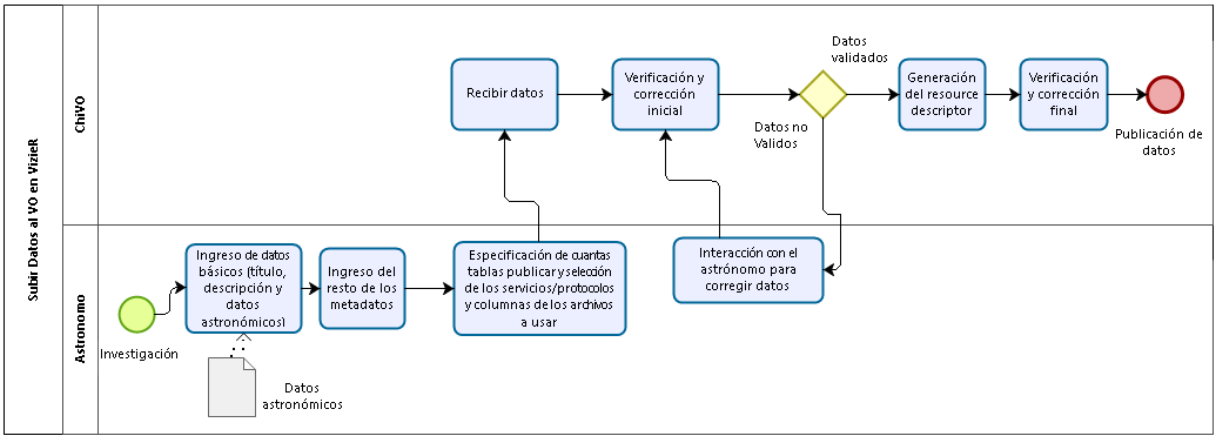


Figura 3.5: Proceso de subida de datos de alto nivel a ChiVO.
Fuente: Elaboración propia.

las observaciones de un científico de ChiVO. La puesta en ejecución de esta primera aproximación del problema, da a pie a mejoras una vez se tengan usuarios continuamente usando la interfaz.

3.4. Descripción del Protocolo

En esta sección se describe detalladamente todas las etapas del protocolo, desde que el astrónomo ingresa a la plataforma para realizar una petición de publicación de sus datos, hasta el final y que efectivamente estos son publicados por el administrador del sistema.

Cabe mencionar primero se describen las etapas que le competen al astrónomo y posteriormente las etapas que debe realizar el administrador.

3.4.1. Etapas correspondientes al astrónomo

3.4.1.1. Registro e inicio de sesión de los astrónomos

Con el fin de hacer el proceso lo más ágil posible, se plantea que en el caso del astrónomo no exista el registro usual, en donde el usuario ingresa su correo y una contraseña. En cambio:

1. El astrónomo deberá ingresar únicamente su correo para comenzar el proceso de subida de datos a la interfaz.

2. El sistema le enviará un e-mail al astrónomo con la URL que le permitirá continuar el proceso en caso de que no termine este inmediatamente (al acceder a esta URL se precargarán todos los datos ya ingresados en los respectivos formularios).

3.4.1.2. Ingreso de los datos básicos

Al iniciar el proceso, lo primero que se le pide al astrónomo es que ingrese un título y una descripción para el conjunto de datos a subir, además de los archivos con los datos astronómicos.

En el caso de ingresar uno o varios archivos FITS será necesario especificar el índice de cada uno de los HDU que se utilizarán al publicar los datos.

En la figura 3.6 se muestra un ejemplo de un formulario para el ingreso de estos datos.

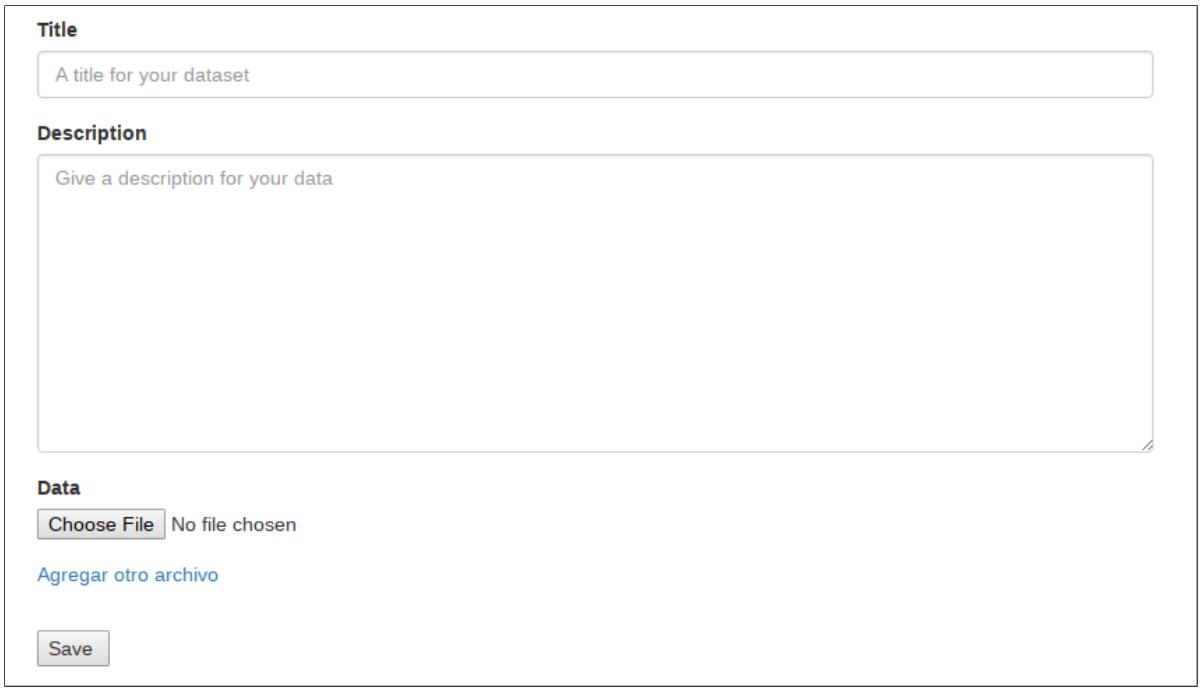


Figura 3.6: Ejemplo de formulario para el ingreso de los datos básicos.
Fuente: Elaboración propia.

3.4.1.3. Ingreso de los metadatos

En esta etapa se le pedirá al astrónomo que ingrese, por medio de un formulario, los siguientes metadatos que describirán el conjunto de datos que desea publicar:

- **Creadores**: Nombre de los autores del conjunto de datos separados por punto y coma.
- **Fuente**: *Paper* asociado a los datos astronómicos. Preferiblemente poner su *bibcode* [16].
- **Temas**: Temas o tópicos de los datos, por ejemplo: *catalog of stars*, *detection of planets and satellites*, etc.

  Se recomienda que las opciones de los valores a ingresar en este campo sean obtenidos del IVOAT Thesaurus [17], un compendio de varios conceptos relacionados a la astronomía (IVOA, 2009). Considerar que se debe permitir la elección de varios tópicos.
- **Instrumento**: Nombre del instrumento utilizado para recolectar los datos.
- **Instalación**: Nombre de la instalación u observatorio de donde se obtuvieron los datos.
- **Tipos de conjunto de datos**: Tipos de los datos a subir. Los posibles valores que pueden ser asignados a este campo se encuentran en el Anexo A.2.1 [17]. Se debe permitir la elección de varios tipos de conjunto de datos.
- **Cobertura espacial**: Representación de la cobertura espacial en formato ICRS.
- **Cobertura espectral**: Intervalos de la cobertura espectral de longitudes de onda de vacío del baricentro del sistema solar en metros.
- **Cobertura temporal**: Intervalos de la cobertura temporal en *Modified Julian Date*.

[16] Código bibliográfico que consiste de 19 dígitos que describen un artículo o *paper* (ADS, nd).

[17] Los posibles valores de los temas, tipos de conjunto de datos y bandas de onda de cobertura, debieran estar en archivos de texto y que los contenidos de estos se carguen en correspondientes *multi-comboboxes* para controlar las opciones que puede ingresar el astrónomo.

- **Bandas de onda de cobertura**: Tipos de ondas asociadas a los datos astronómicos. Los posibles valores para este campo son los siguientes [17]:

  - Radio
  - Millimeter
  - Infrared
  - Optical
  - UV (Ultraviolet)
  - EUV (Extreme ultraviolet)
  - Xray
  - Gammaray

  Tener en cuenta que se debe permitir la elección de varias bandas de onda de cobertura.

Además de estos campos ingresados por el astrónomo, será necesario que también se guarde en la base de datos la fecha con formato ISO en UTC del momento en que el astrónomo complete esta etapa. Este es un dato que se usará posteriormente en la generación del *resource descriptor*.

3.4.1.4. Selección de los datos a publicar

En esta etapa se le pedirá al astrónomo que seleccione los datos a incorporar al sistema. Para esto el usuario tendrá que definir tablas (asignándoles nombres distintos a cada una) y establecer como se agrupan las columnas o datos de los archivos en estas tablas.

En la figura 3.7 se muestra un formulario de ejemplo para esta etapa, en el cual se está especificando la creación de 2 tablas. En la primera el astrónomo especifico que se publicaran los campos RA y DEC, mientras que en la segunda se publicarán los campos GLAT y GLON. Tener en cuenta que estos datos (RA, DEC, GLAT y GLON) provienen de los archivos subidos en la etapa 3.4.1.2, es decir, el sistema deberá acceder al contenido de estos archivos y permitir que el astrónomo elija,

de forma sencilla, que datos publicar. Para más detalles sobre el procesamiento del contenido de los archivos ver el anexo A.4.

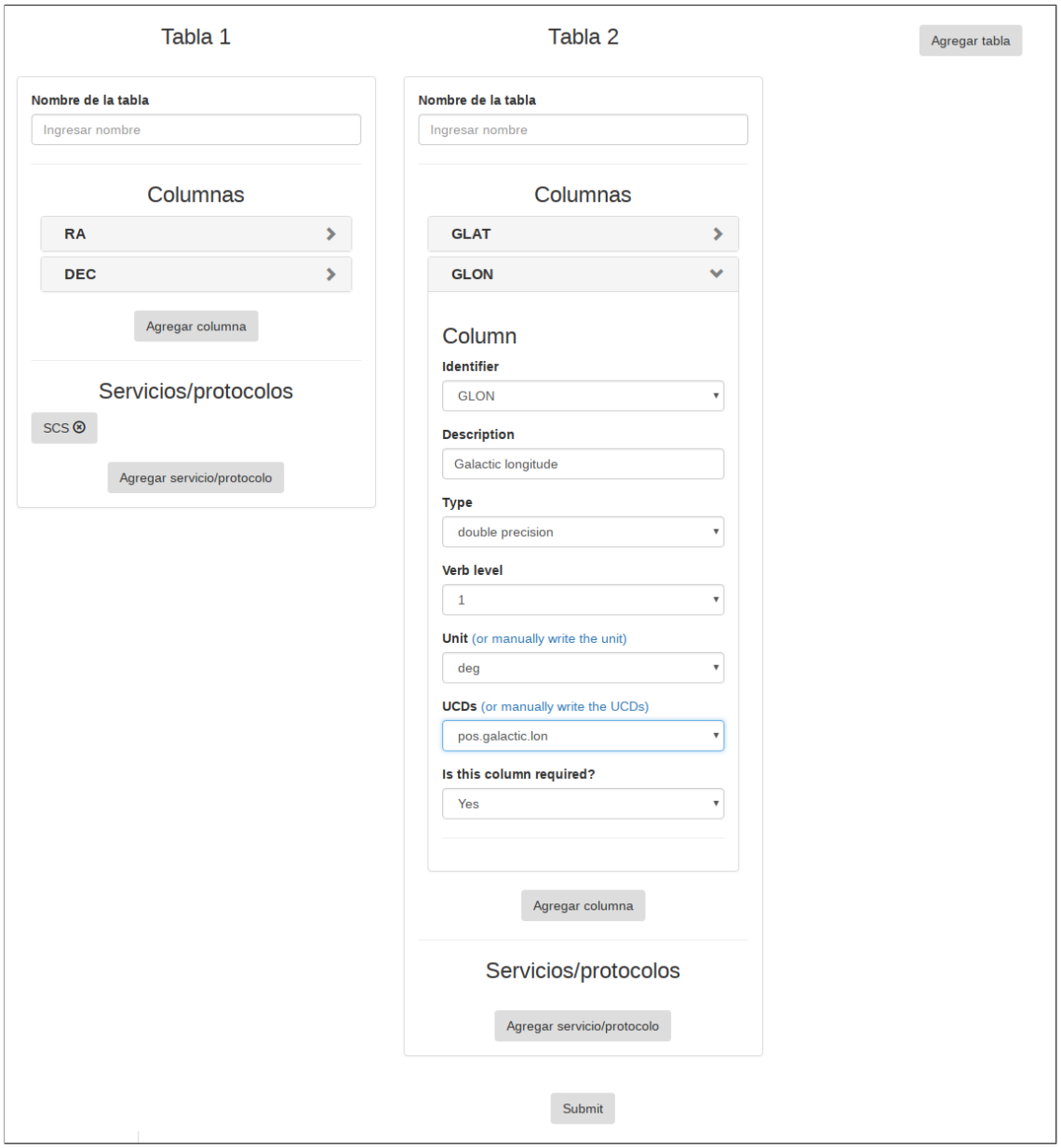


Figura 3.7: Especificación de las columnas a publicar.
Fuente: Elaboración propia.

Puede existir el caso en donde un astrónomo desee publicar, en una misma columna, datos que se encuentren repartidos en varios archivos. Esto quiere decir que, usando la figura 3.7 de ejemplo, los campos GLAT y GLON podrían provenir de varios de los archivos subidos por el astrónomo.

El sistema tendrá que permitirle al astrónomo especificar estos casos, pero será necesario tener en cuenta un par de restricciones:

- Todos los datos que se agrupen en una misma tabla deben pertenecer al mismo conjunto de archivos, por ejemplo, no puede darse el caso que GLAT se encuentre en un conjunto de archivos y GLON provenga de otro conjunto de archivos distintos.
- Todos los archivos asociados a los datos que el astrónomo quiere publicar en una misma tabla deben tener el mismo formato y estructura.

Además, el astrónomo deberá especificar que protocolos/servicios de la IVOA desea utilizar en cada una de las tablas, lo que está sujeto a algunas restricciones, las cuales se pueden ver en el anexo A.3. Asimismo, la interfaz debiese incluir documentación respecto a estos protocolos y servicios, en caso de que un astrónomo no conozca sus significados.

Un punto importante a considerar es que para cada columna seleccionada por el astrónomo será necesario especificar los metadatos de esta, éstos son: descripción, tipo de dato, unidad, importancia (verb level), UCD y por último un valor de verdadero o falso que determine si la columna es requerida. Para esto, se debería proveer de *inputs*, en donde el astrónomo indique estos valores y en el caso de los campos que tienen vocabularios controlados se debiesen mostrar las opciones disponibles en varios *comboboxes*, siendo el contenido de estos cargados desde varios archivos de texto. Las opciones disponibles para las unidades y tipo de dato se pueden ver en los anexos A.1.1 y A.2.5 respectivamente, mientras que en el caso de las UCD estas opciones se encuentran en el enlace http://www.ivoa.net/documents/UCD1+/index.html (IVOA, 2018d).

Además, la unidad, tipo de dato y UCD generalmente se encuentran en los archivos FITS y VOTable, por lo que cuando un astrónomo seleccione un dato a publicar que se encuentre en un archivo con estos formatos, se debiese seleccionar automáticamente las opciones de unidad, tipo de dato y UCD, facilitándole la tarea al astrónomo. En el caso de seleccionar datos que provengan de

archivos en formato CSV, TSV o que aunque se encuentren en archivos de formato FITS o VOTable, no se tengan estos datos, el astrónomo deberá seleccionarlos manualmente.

Por último, cabe mencionar un par de detalles más:

- En caso de que el astrónomo quiera usar el protocolo ObsCore, se deberá agregar la opción de que un astrónomo declare una columna como perteneciente a este estándar y que realice el *match* para identificar a cual columna del ObsCore corresponde. Ante este hecho no será necesario ingresar los metadatos de la columna, es decir, descripción, tipo de dato, etc. Pero si se debiese indicar si la columna representa un dato único (entero, *string*, etc.) o un arreglo de datos.
- En caso de que un dato seleccionado por el astrónomo pueda tener un prefijo del Sistema Internacional, es necesario permitirle al astrónomo ingresar este prefijo y que especifique si desea realizar alguna operación matemática a los datos antes de ser publicados. Los prefijos se pueden visualizar en el anexo A.1.3.

Habiendo realizado las etapas anteriormente mencionadas, se habrá creado una petición de publicación de datos en el sistema.

3.4.2. Etapas correspondientes al administrador

A continuación, se detallan las etapas del protocolo que le competen al administrador, a la hora de publicar los datos que un astrónomo subió al sistema.

3.4.2.1. Verificación y corrección inicial

Es necesario considerar que los datos tienen que ser verificados y aprobados, puesto que estos quedarán a disposición para toda la comunidad científica en caso de ser publicados.

Después de que un astrónomo haya finalizado la etapa descrita en la sección 3.4.1.4 y se haya creado una petición de publicación de datos en la interfaz, el administrador deberá realizar las siguientes acciones:

1. Verificar la coherencia de los datos subidos por el astrónomo (datos básicos, metadatos, columnas y archivos).

2. En caso de comprobar que la petición de publicación de datos no presenta errores y no se tiene duda respecto a los datos, será aprobada. Si esta presenta incoherencias, entonces el administrador tiene 2 opciones:

   - Si el error es considerado factible de corregir por parte del administrador (por ejemplo, un error tipográfico), éste deberá arreglarlo, editando la petición y aprobándola.
   - Si el administrador no está seguro de como arreglar el error o tiene dudas respecto a los datos, deberá contactar al astrónomo para resolver las dudas planteadas.

3.4.2.2. Generación del *resource descriptor* para DaCHS

En esta etapa se le permitirá al administrador crear de forma **automática**, el *resource descriptor* de un conjunto de datos subidos por un astrónomo, el cual debe estar previamente aprobado por un administrador.

Es necesario considerar que los *resource descriptors* son documentos escritos en formato XML, por lo tanto para el desarrollo de esta funcionalidad, se requerirá desarrollar un programa o *script* que escriba en este formato y que al usar los datos entregados por el astrónomo en la sección 3.4.1 genere el RD.

3.4.2.2.1 Resumen

De manera introductoria, se realiza una definición a grandes rasgos sobre como se generará automáticamente el *resource descriptor* (RD). Pero, se considera que antes de ahondar en esta

temática es necesario tener una noción general de la función y estructura general de un RD, esta explicación se puede ver en el anexo A.5. También se recomienda observar el anexo A.6, donde se muestra el ejemplo de la construcción de un RD.

A continuación se detalla, etapa por etapa, la generación automática del RD asociado a la petición de publicación de datos de un astrónomo.

3.4.2.2.2 Definición de los Metadatos

Para comenzar la construcción del esqueleto de nuestro *resource descriptor* (Demleitner, 2018a), primero se define el elemento raíz `resource` . Este elemento determina el esquema de la base de dato de DaCHS que se usa en la ingestión. Para este fin se utiliza el título ingresado en la sección 3.4.1.2, pero aplicándole cambios:

- Se cambian los caracteres a minúscula.
- Se remplazan los espacios por guiones bajo.
- Si el primer carácter es un número, este es eliminado.

De esta forma, si el título ingresado por el astrónomo es “ARIHIP astrometric catalogue”, el resultado en el RD debiese ser `<resource schema="arihip_astrometric_catalogue">` .

Cabe destacar que los títulos ingresados por los astrónomos deben ser únicos, ya que si se generan 2 *resource descriptors* distintos con el mismo esquema se podrían generar conflictos. Por esto es necesario que, si un astrónomo ingresa un título ya existente en la base de datos, la interfaz le indique esta situación y le pida que elija otro título.

Continuando con la definición del RD, primero se declaran los metadatos del conjunto de datos astronómicos. Estos son útiles, ya que dan a entender el contexto de los datos, antes de acceder a

ellos. En el protocolo estos metadatos son ingresados por el astrónomo en las etapas 3.4.1.2 y 3.4.1.3 por medio de formularios.

Para dar un ejemplo, suponiendo que un astrónomo el 4 de Enero del 2018 a las 10:13 PM ingresó los datos mostrados a continuación (Wielen et al., 2010) en las etapas anteriormente mencionadas:

- **Título**: ARIHIP astrometric catalogue
- **Descripción**: The catalogue ARIHIP has been constructed by selecting the 'best data' for a given star from combinations of HIPPARCOS data with Boss' GC and/or the Tycho-2 catalogue as well as the FK6. It provides 'best data' for 90 842 stars with a typical mean error of 0.89 mas/year (about a factor of 1.3 better than Hipparcos for this sample of stars).
- **Creadores**: Wielen, R.; Schwan, H.; Dettbarn, C.
- **Temas**: Catalogs ; Astronometry ; Stars: Proper Motions
- **Tipo**: Catalog
- **Banda de onda de cobertura**: Optical
- **Cobertura temporal**: 1989-09-01 1993-08-15
- **Cobertura espacial**: 0/0-11
- **Cobertura espectral**: 4.8e-7 7.3e-7

El resultado al generarse el *resource descriptor* debiese ser el que se ve a continuación en el código 2:

Código 2: Metadatos en el RD

```
<meta name="title">arihip_astrometric_catalogue</meta>
<meta name="creationDate">2018-01-04T10:13:00</meta>

<meta name="description">
  The catalogue ARIHIP has been constructed by selecting the 'best data' for a given
      star from combinations of HIPPARCOS data with Boss' GC and/or the Tycho-2
      catalogue as well as the FK6. It provides 'best data' for 90 842 stars with a
      typical mean error of 0.89 mas/year (about a factor of 1.3 better than Hipparcos
      for this sample of stars).
</meta>

<meta name="creator">
  <meta name="name">Wielen, R. ; Schwan, H. ; Dettbarn, C.</meta>
</meta>

<meta name="subject">Catalogs</meta>
<meta name="subject">Astrometry</meta>
<meta name="subject">Galaxies: evolution</meta>

<meta name="type">Catalog</meta>

<meta name="coverage.waveband">Optical</meta>

<coverage>
        <temporal>55031.5 55031.5</temporal>
        <spatial>6/34535</spatial>
        <spectral>4e-7 8e-7</spectral>
</coverage>
```

Recordar que se dictó que se debe guardar la fecha al momento que el astrónomo ingresa los

metadatos, esto se debe a que el elemento `creationDate` debe contener esta fecha.

3.4.2.2.3 Especificación de las columnas

A continuación, se debe definir los elementos `table` . Entre estas etiquetas se plasma toda la información ingresada por el astrónomo durante la selección de los datos a publicar.

Por cada tabla declarada por el astrónomo se crea un elemento `table` en el RD. Este elemento tiene los siguientes atributos:

- `id` : Nombre de la tabla que se genera en la base de datos de DaCHS. Se le asigna el nombre de la tabla ingresada por el astrónomo.
- `onDisk` : Define si los datos son guardados en el disco duro o solamente en memoria, en este caso siempre se busca lo primero, así que se le designa el valor de *True*.
- `adql` : Define si están habilitadas las consultas ADQL en la tabla. Las tablas que el astrónomo especificó que soportarán este protocolo deben tener un valor de *True*, en caso contrario se le asigna el valor de *False*.
- `primary` : Especifica la columna que es considerada como la clave primaria de la tabla. Esta columna corresponde a la que contiene un UCD igual a `meta.id;meta.main`. En caso de que no exista dicha columna, se omite este atributo.

Además, en los elementos `table` se definen los elementos hijos `column` , los cuales describen las columnas de las tablas en la base de datos, donde se guardan los datos subidos por el astrónomo.

Al usar el ejemplo de la figura 3.7, el elemento `column` correspondiente a la columna GLON que se debiese generar en el RD, se puede ver en el siguiente código:

Código 3: Columa GLON

```
<column name="glon" type="double precision" ucd="pos.galactic.lon" verbLevel="1" unit="
    deg" description="Galactic longitude" required="True"/>
```

Por último, se tiene el atributo o elemento hijo `mixin` . En DaCHS representan la adición de todos los elementos necesarios (columnas, metadatos, indices, etc.) para que una tabla pueda servir un estándar de la IVOA (ObsCore, SIA, SCS, etc.). Dependiendo de los servicios/protocolos que el astrónomo asigne a cada tabla, diferentes mixins son necesarios:

- En el caso de que el usuario haya especificado que se use el protocolo **SCS** en una tabla, ésta tendría que usar el mixin `scs#q3cindex` . Este mixin tiene como condición que como mínimo existan columnas con datos posicionales RA y DEC, además de otra que actúe como un identificador. Para ésto, DaCHS corrobora que se tenga únicamente una columna con las UCD `pos.eq.ra;meta.main`, `pos.eq.dec;meta.main` y `meta.id;meta.main` en la tabla correspondiente.

  Tener en cuenta que los datos RA y DEC deben estar al menos en el estándar J2000 (Murray, 1989).

- En el caso de que el usuario haya especificado que desearía proveer el servicio **ObsCore** se tiene tres casos dependiendo del tipo de producto de dato:

  - Para catálogos se usa `obscore#publish`
  - Para imágenes se usa `obscore#publishSIAP`
  - Para espectro se usa `obscore#publishSSAPMIXC`

  En los tres casos mencionados es necesario definir una serie de atributos dentro del elemento `mixin` , los cuales mapean los datos provistos por el astrónomo (en el que especificó que pertenecen al ObsCore) con una columna del Obscore (IVOA, 2017). Estos atributos corresponden a los que se ven en el anexo A.2.2.

Como ejemplo ver el código 4 que muestra la definición de un mixin obscore en un *resource descriptor*:

Código 4: Ejemplo del mixin obscore#publish

```
<mixin
    size="10"
    mime="'application/x-votable+xml;content=datalink'"
    calibLevel="3"
    collectionName="'CALIFA'"
    coverage="s_region"
    dec="s_dec"
    emResPower="em_res_power"
    expTime="t_exptime"
    facilityName="'Calar Alto'"
    fov="0.01"
    instrumentName="'PMAS/PPAK at 3.5m Calar Alto'"
    oUCD="'phot.flux;em.opt'"
    productType="'cube'"
    ra="s_ra"
    sResolution="0.0002778"
    title="obs_title"
    tMax="t_max"
    tMin="t_min"
    targetClass="'Galaxy'"
    targetName="target_name"
    emMin="em_min"
    emMax="em_max"
>//obscore#publish</mixin>
```

- Si el astrónomo especificó el servicio SIAP en una de las tablas, se usa el mixin `siap#pgs` para que pueda responder consultas de este tipo. Este mixin no tiene atributos.

- Si el astrónomo especificó proveer el servicio SSAP se usa el mixin `ssap#mixc` . En este caso existen atributos, a los cuales, al igual que en el caso del mixin obscore, se deben mapear a alguna columna ingresada por el astrónomo. Estos atributos se pueden ver en el anexo A.2.3, pero con el fin de no sobrecargar al astrónomo con tareas al momento de seleccionar las columnas, es responsabilidad del administrador realizar este *matching*, editando el *resource descriptor*, después de ser generado.

  Un ejemplo de este mixin se puede ver a continuación en el código 5:

  Código 5: Ejemplo del mixin ssap#mixc

```
<mixin
    spectralUnit="10**-10 m"
    fluxUnit="0.1 J/(m**2.s.m)"
    spectralUCD="em.wl"
    fluxUCD="phot.flux.density"
>//ssap#mixc</mixin>
```

  En caso de contar con datos de las posiciones, apertura y que estos se publiquen vía ObsCore, es altamente recomendado agregar el mixin `ssap#simpleCoverage` , el cual no tiene atributos, pero si se pide que al atributo `coverage` del mixin `obscore#publishSSAPMIXC` se le asigne el valor de `ssa_region` . Esto también debiese ser tarea del administrador.

Por último, tener en cuenta que, al momento de definir los atributos de un mixin, se tienen 3 casos:

- Se asigna un número. En este caso se encierra el valor entre comillas dobles.
- Se asigna una cadena de caracteres. En este caso se encierra el valor entre comillas simples y dobles.
- Se asigna una columna definida en el *resource descriptor*. En esta ocasión se encierra el nombre

de la columna entre comillas dobles. Por ejemplo, en el mixin del código 4, al atributo coverage le corresponde los valores de la columna `s_region` .

Para diferenciar estos casos, se solicita que el astrónomo, al momento de declarar que una de las columnas que seleccionó pertenece al ObsCore, declare si se trata de un dato único o de un arreglo de datos.

3.4.2.2.4 Ingestión de los datos

En esta sección se debiese definir el elemento `data` , que determina que archivos son usados en la construcción de cada tabla.

Tener en cuenta que por cada elemento `table` definido en el RD resultante se tiene que crear un elemento `data` asociado. Este elemento tiene el atributo `id` , que no tiene mucha relevancia, sólo sirve para asignarle un nombre o un identificador al proceso de ingestión. En el caso del protocolo propuesto se asigna el *string* resultante de concatenar la cadena de caracteres `import_` con el nombre de la tabla correspondiente.

El primer elemento hijo de `data` a definir es `sources` , este permite declarar la ruta de los archivos usados en la ingestión. Para saber la ruta a ingresar en `sources` es necesario considerar el árbol de directorios comúnmente usados para un conjunto de datos. Como se mencionó en la sección 2.3.3, DaCHS tiene un *resource directory* por cada conjunto de datos a publicar. En este directorio se guarda tanto el RD como los mismos datos astronómicos a ser publicados. Estos últimos se encuentran generalmente en el directorio *data*.

Es decir, que la jerarquía de los directorios y archivos es representado en la figura 3.8:

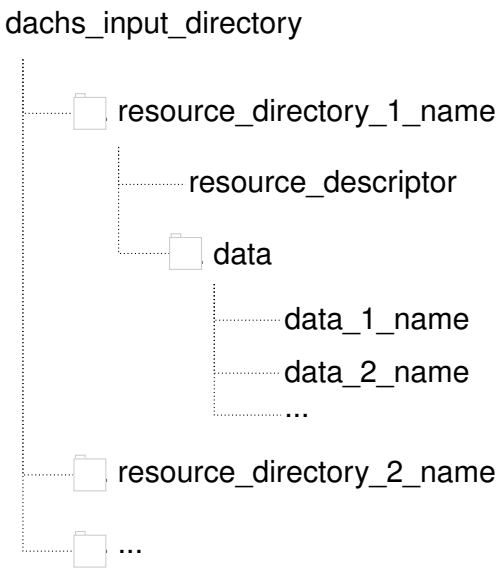


Figura 3.8: Organización típica de los archivos con los datos astronómicos y sus respectivos *resource descriptors* usados por DaCHS.
Fuente: Elaboración propia.

Por lo que desde la ruta del q.rd, los datos astronómicos se encuentran siempre en la ruta data \astronomical_data_filename. Debido a esto, suponiendo un caso en que los archivos con los datos astronómicos son asu.fits y asu_2.fits, el elemento `sources` debiese escribirse como el ejemplo del código 6:

Código 6: Ejemplo del elemento sources

```
<sources>
    <pattern>data/asu.fits</pattern>
    <pattern>data/asu_2.fits</pattern>
</sources>
```

En el caso de que exista sólo un archivo de datos para esta tabla, perfectamente se puede tener: `<sources>data/asu.fits</sources>`

Ya habiendo declarado los archivos a usar, falta especificar como mapear las columnas de los archivos a las columnas de las tablas declaradas en el *resource descriptor*. Para aquello existe el concepto de *grammar* que analiza los archivos y retorna el mapeo (diccionarios) de las columnas en

los archivos especificados.

Como en el contexto de la astronomía existe una gran variedad de formatos de archivos, existe una gran variedad de *grammars* justamente para cubrir la mayoría de los casos posibles.

Debido a que se declaró que esta interfaz soporta el ingreso de datos FITS, VOTable, CSV y TSV se utiliza únicamente 5 *grammars* (aunque existen muchos más), los cuales son:

- En el caso de que el astrónomo haya seleccionado columnas provenientes de una unidad de datos de un archivo FITS se usa `fitsTableGrammar` .

  Al usar este *grammar* por defecto se accede al HDU 1, pero también se puede definir el HDU a utilizar con el atributo `hdu` .

  Ejemplo: `<fitsTableGrammar hdu="3/>`

- En el caso de que el astrónomo haya seleccionado columnas provenientes de una cabecera de FITS se usa `fitsProdGrammar` .

  Al usar este *grammar* por defecto se accede al HDU primario, pero también se puede definir el HDU a utilizar con el atributo `hdu` .

  Ejemplo: `<fitsProdGrammar hdu="4/>`

- En el caso de publicar datos de archivos en formato VOTable se usa `voTableGrammar` .

  Tener en cuenta que en este caso sólo se puede acceder a los datos de la primera tabla definida en el archivo.

- En el caso de que los datos astronómicos se encuentren en un archivo de formato CSV o TSV se usa `csvGrammar` .

  Como este tipo de formato (al igual que la mayoría de los formatos *delimiter separated values*) no está estandarizado, se le debe especificar a DaCHS las columnas existentes en el archivo, debido a esto, aquí se ingresan los nombres de todas las columnas ingresadas por el astrónomo,

como se menciona en el anexo A.4. Así DaCHS sabe que columnas existen en el archivo CSV o TSV al momento de realizar la ingestión.

A continuación, se muestra un ejemplo de como debiese ser el elemento `csvGrammar`, en el caso de la ingestión de un archivo CSV con las columnas lambda, b5v, a0v, a5v, f0v, g0v, g5v, k0iii, m0iii, m5iii.

Código 7: csvGrammar

```
<csvGrammar strip="True" topIgnoredLines="49" delimiter=",">
  <names>lambda,b5v,a0v,a5v,f0v,f5v,g0v,g5v,k0iii,m0iii,m5iii</names>
</csvGrammar>
```

Tener en cuenta de que es necesario proveer los siguientes atributos:

- `topIgnoredLines` : Números de líneas superiores ignoradas.
- `delimite` : Delimitador, el cual es una coma por defecto, pero para el caso de archivos TSV se debiese usar un *tab*.
- `strip` : Especifica si se eliminan los espacios en blanco entre delimitadores.

Estos atributos debiesen ser asignados por el administrador, verificando la estructura de los archivos y editando el RD una vez se haya generado.

Observando el ejemplo del código anterior, se puede inferir que al momento de la ingestión de los datos se ignoran las primeras 49 líneas, se está usando un archivo en formato CSV y se eliminan los espacios en blanco entre las comas.

A continuación, se tiene el elemento `make`, en el que se define como usar el mapeo retornado por los *grammar* para llenar las tablas en la base de datos de DaCHS.

En este elemento se tiene el atributo `id` que especifica la tabla en donde se ingresan los datos, además del elemento `rowmaker` que define el emparejamiento entre el diccionario devuelto por el *grammar* y la columnas de la tabla.

Para ejemplificar cómo se conecta el concepto de los elementos `table` y `make`, ver el código 8:

Código 8: Ejemplo de table

```
<table id="main" onDisk="True" adql="True" mixin="//scs#q3cindex">
    <column name="_ra" description="Right ascencion" type="double precision" ucd="pos.eq.
        ra;meta.main" unit="deg" verbLevel="1" required="True"/>
    <column name="_de" description="Declination" type="double precision" ucd="pos.eq.dec
        ;meta.main" unit="deg" verbLevel="1" required="True"/>
</table>

<make table="main">
    <rowmaker>
        <map dest="_ra">@_RA</map>
        <map dest="_de">@_DE</map>
    </rowmaker>
</make>
```

Con el carácter arroba se accede a los elementos retornados por el *grammar*, es decir, los datos de los archivos subidos por el astrónomo, por lo que @_RA y @_DE hacen referencia a las columnas _RA y _DE de los archivos. Además, con `map` se declara hacia que columnas de la base de datos se ingresan los datos de las columnas de los archivos.

También existe el caso en donde no se quiera simplemente mapear los datos de los archivos a la base de datos, sino que también aplicar operaciones matemáticas a estos de acuerdo al deseo del astrónomo. Para esto se puede usar el elemento `var` antes de `map`.

Por ejemplo, en la siguiente sección del RD, se esta ingresando los datos del *right ascencion* multiplicados por 10. Esto sucede ya que al usar `var` se define una variable en la cual se permite usar expresiones de Python.

Código 9: Ejemplo de table

```
<var name="_tmpRA">float(@_RA)</var>

<map dest="_ra">(@_tmpRA * 10.0)</map>
```

Se debe considerar que de haber habilitado el protocolo SIAP, dentro del `rowmaker` tendrán que ejecutarse los procedimientos `siap#computePGS` y `siap#setMeta` . `siap#computePGS` no tiene argumentos, pero espera *keywords* especiales tipo WCS [18] en el diccionario retornado por el *grammar*. Por otro lado, `siap#setMeta` se encarga de establecer metadatos para el protocolo SIAP, al igual que con el ObsCore es necesario que se emparejen columnas seleccionadas por el astrónomo con los parámetros de este procedimiento. La lista de estos parámetros se muestra en el anexo A.2.4. Esta tarea también tiene que ser realizada por el administrador, editando el *resource descriptor* después de haber sido generado.

3.4.2.2.5 Declaración de los servicios

Finalmente, en el elemento `service` se define la forma de hacer las *queries* a los datos y el cómo se visualizan los correspondientes resultados.

Por cada tabla se debe crear un elemento `service` . Los primeros atributos que este elemento tiene, son:

- `id` : Identifica el servicio y forma parte de la URL.
- `allowed` : Define los distintos tipos de *renderers* disponibles para el uso de ese servicio. Estos determinan visualmente los resultados de las *queries*. Existen varias posibilidades como *form* (para el uso de formularios web al realizar consultas a los datos ingresados) y otros que dependen de los servicios que se escogieron:

[18] Un estándar del formato FITS que define coordenadas astronómicas.

- scs.xml para el caso de **SCS**.
- siap.xml para el caso de **SIAP**.
- ssap.xml para el caso de **SSAP**.

Los elementos hijos comunes de la etiqueta `service` son:

- `meta` : El cual define metadatos que describan los servicios, como un título y un nombre corto. Un ejemplo se puede ver en el código 10:

Código 10: Metadatos de un servicio

```
<meta name="title">Lensed QSOs, bidimensional spectra (SSAP)</meta>
<meta name="shortName">mlqso bidi ssa</meta>
```

En este protocolo se usa el nombre de la tabla relacionada para asignarle estos datos, donde:

- A `title` se le debe asignar un *string* dependiendo del tipo de los protocolos/servicios de la tabla, pero concatenándole también el título general de los datos y el nombre de la tabla.
  - En caso de usar SCS el primer *string* es `SCS` .
  - En caso de usar SIAP el primer *string* es `SIAP` .
  - En caso de usar SSAP el primer *string* es `SSAP` .
  - En otro caso se debiese usar como *string* el mismo `SCS` .

  Por ejemplo, si el título ingresado fue “ARIHIP astrometric catalogue” y se desea publicar una tabla con el nombre *data* que soporte el protocolo SCS, por defecto el título de este servicio debiese ser “SCS ARIHIP astrometric catalogue data”.
- A `shortName` se le debe asignar el acrónimo en mayúscula del *string* asignado a `title` .

- ***Core***: Estos elementos definen las consultas en la base de datos. Dependiendo de los servicios/-protocolos escogidos por el usuario se tiene que optar por uno de los siguientes: `dbCore` , `scsCore` , `siapCutoutCore` o `ssapCore` .

  En el elemento *core* se pueden definir *inputs* con los elementos `condDesc` e `inputKey` , es decir, la entrada que debe ingresar el usuario para buscar datos que calcen con lo ingresado en el *input*. También se puede especificar el *output* con `outputTable` .

  Por ejemplo, con la configuración del código 11, se define un servicio, en el cual se puede ingresar el valor de la columna *identifier* y son retornados todos los datos que le correspondan al *input* ingresado.

Código 11: Definiendo parámetros de los formularios

```
<condDesc>
    <inputKey original="identifier" required="True"/>
</condDesc>
<outputTable verbLevel="30"/>
```

La elección del *core* se divide en 4 casos:

- En el caso de que a una tabla no se le haya asignado servicio/protocolo por parte del astrónomo, se tiene que hacer uso del *render form* y el *core* `dbCore` , solo si se tiene una columna con el UCD `meta.id;meta.main`, usando esta columna como *input*.

- En el caso de que a una tabla se le haya asignado servicio/protocolo SCS se deberá usar el *render form* y scs.xml, además del *core* `scsCore` .

  Dentro de la definición de este *core* también se debe usar el elemento `FEED` , asignándole el atributo `source=//scs#coreDescs"` . Con esto se define automáticamente los *inputs* necesarios para realizar una consulta SCS.

  Un ejemplo de este tipo de servicios es:

Código 12: Ejemplo de servicio SCS

```
<service id="cone" defaultRenderer="form" allowed="scs.xml,form,static">
    <meta name="title">lat SCS</meta>
    <meta name="shortName">latc</meta>
    <scsCore queriedTable="main">
        <FEED source="//scs#coreDescs"/>
    </scsCore>
    <publish render="scs.xml" sets="ivo_managed"/>
    <outputTable verbLevel="30"/>
</service>
```

- En el caso de que a una tabla se le haya asignado servicio/protocolo SSAP se debe usar los *renders form* y ssap.xml, además del *core* `ssapCore` .

  Dentro de la definición de este *core* también se debe usar el elemento `FEED` , con el atributo `source=//ssap#hcd_condDescs"` . Con esto se definen automáticamente los *inputs* necesarios para realizar una consulta SSAP.

  Además, se debe definir una serie de metadatos que son particulares al protocolo SSAP, los cuales se pueden visualizar en el ejemplo siguiente:

Código 13: Ejemplo de servicio SSAP

```
<service id="ssa" allowed="form,ssap.xml,static">
    <meta name="shortName">zCosmos SSAP</meta>
    <meta name="title">zCosmos Bright Spectroscopic Observations DR2</meta>
    <meta name="ssap.dataSource">pointed</meta>
    <meta name="ssap.testQuery">MAXREC=1</meta>
    <meta name="ssap.creationType">archival</meta>
    <meta name="ssap.complianceLevel">query</meta>
    <publish render="ssap.xml" sets="ivo_managed"/>
    <publish render="form" sets="ivo_managed,local" service="web"/>

    <property name="datalink">sdl</property>

    <ssapCore queriedTable="data">
        <FEED source="//ssap#hcd_condDescs"/>
    </ssapCore>
</service>
```

- En el caso de que a una tabla se le haya asignado servicio/protocolo SIAP se debe usar los *renders form* y siap.xml, además de los *cores* `siapCutoutCore` o `dbCore` .

  Dentro del *core* también se debe definir 2 elementos `condDesc` , uno con el atributo `original=//siap#protoInput"` y otro con `original=//siap#humanInput"` . Con esto se definen automáticamente los *inputs* necesarios para realizar una consulta SIAP.

Con esto concluye los aspectos a considerar para la generación automática del RD.

3.4.2.3. Verificación y corrección final

En esta etapa se verifica que el *resource descriptor* esté correctamente generado en 2 pasos:

1. **Verificar que el *resource descriptor* sea correcto en su semántica**: Aquí se corrobora que los elementos definidos en el RD tengan coherencia con los datos provistos por el astrónomo.

   - El tipo de dato, unidad y UCD de las columnas deben ser coherentes a su nombre y descripción.
   - El *grammar* a usar debe ser coherente con el formato y contenido del archivo.
   - Los mixins y cores/servicios deben ser coherentes con el tipo de producto de dato provisto.

2. **Realizar *debugging***: Por último, se debe usar el comando `gavo --debug val q` para validar un RD. En caso de que algún elemento del archivo presente problemas, este comando retorna un error.

   Un administrador debiese ejecutar este comando desde la misma interfaz propuesta.

3.4.2.4. Publicación de los datos

Por último, después de haber realizado todos los pasos anteriores de forma exitosa, se realiza la publicación de los datos en sí, lo cual consta básicamente de un comando de DaCHS `gavo --debug imp q` .

Después de esto se puede ver estadísticas de los datos ingresados por medio del comando `gavo info` . Así se puede comprobar que los datos hayan sido correctamente ingresados a la base de datos de DaCHS.

En este caso un administrador también debiese ejecutar ambos comandos desde la misma interfaz propuesta.

3.5. Base de Datos y Modelo de Datos

Considerando las características del proyecto, se debiese usar una base de datos relacional, el cual es el modelo de base de datos más usado en la actualidad. Además, el uso de este modelo de

base de datos suele ser bastante simple, ya que la organización de los datos y su estructura son bastante intuitivas, agrupando los datos en tablas compuestas por columnas y filas, en donde las tablas representan tipos de objetos [19], las filas registros específicos y las columnas atributos de los datos.

Employees Table

Employee_Number	First_name	Last_Name	Date_of_Birth	Car_Number
10001	John	Washington	28-Aug-43	5
10083	Arvid	Sharma	24-Nov-54	null
10120	Jonas	Ginsberg	01-Jan-69	null
10005	Florence	Wojokowski	04-Jul-71	12
10099	Sean	Washington	21-Sep-66	null
10035	Elizabeth	Yamaguchi	24-Dec-59	null

Figura 3.9: Ejemplo de una tabla en una base de datos relacional.
Fuente: Retrieved from https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html

El motor de base de datos escogido es PostgreSQL, ya que ha obtenido una gran reputación en servidores web por ser una base de datos potente y *open source*, además que muchas de las herramientas complementarias que tiene este motor de base de datos también son de código libre y sin cargo para su uso.

Para el diseño del modelo de datos se tiene que pensar en cual es la estructura lógica en que se desean guardar los datos necesarios para la construcción del *resource descriptor*.

Como en este caso se está trabajando con datos asociados a archivos, es lógico contener en una tabla los metadatos del archivo en sí como la ruta, el formato y a qué astrónomo le corresponden los datos de este archivo. Otras tablas deberán ser usadas para guardar los metadatos de las columnas, el nombre de las tablas, los correos y datos de los administradores, etc.

Considerando lo anterior, el modelo de datos planteado para la interfaz se puede ver en la figura 3.10.

[19] En este contexto tipo de objeto hace referencia a que, por ejemplo, una tabla podría representar registros de autos, mesas o personas.

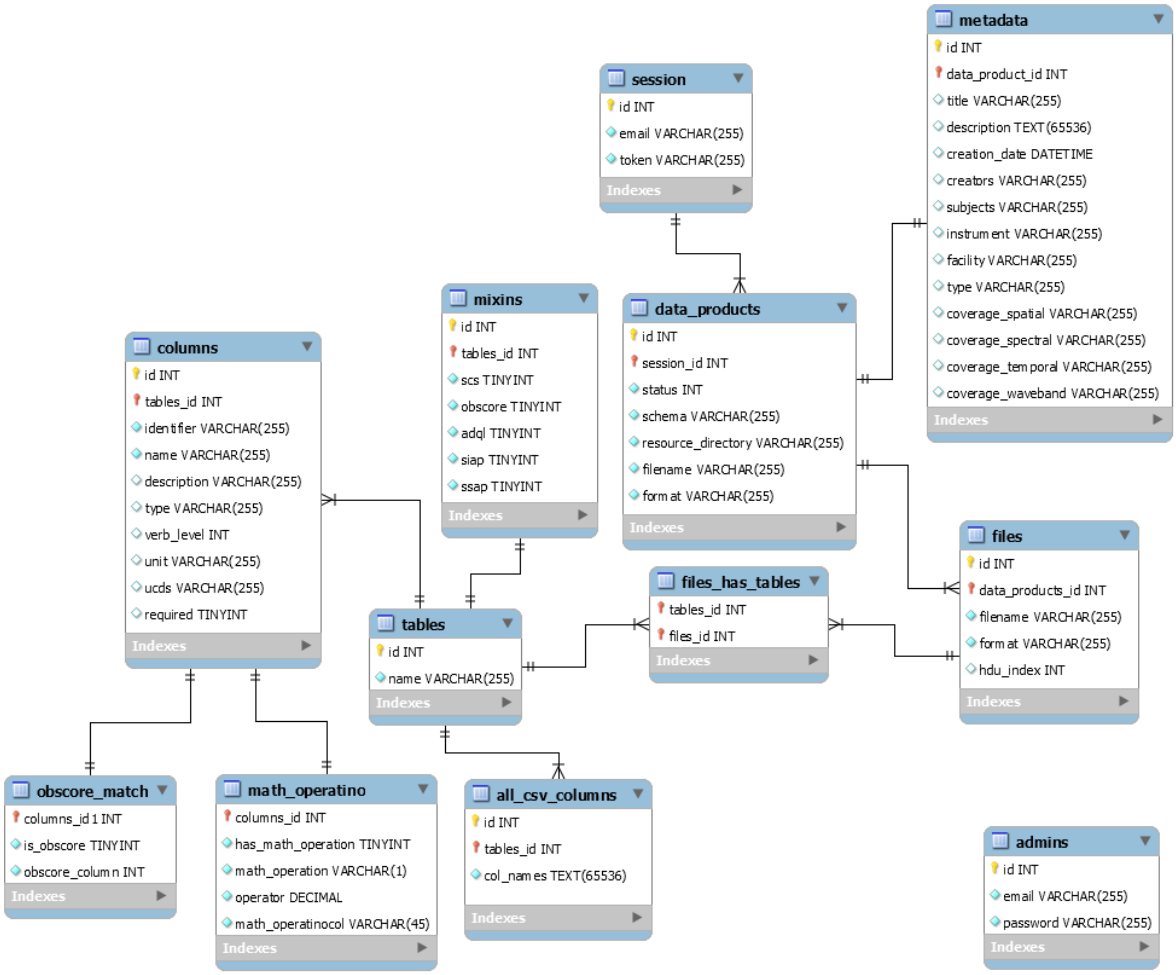


Figura 3.10: Modelo de datos de la interfaz propuesta.
Fuente: Elaboración propia.

3.6. *Framework* para el Desarrollo de la Interfaz

La elección del *framework* es clave en el proceso subsiguiente del desarrollo de la interfaz, ya que de esta elección depende todo el desarrollo futuro hasta la implementación y posterior mantención.

Los puntos a considerar son:

- Soporte de la comunidad.
- Tiempos de desarrollo.
- Librerías de terceros.

El soporte de la comunidad hoy en día es un aspecto a tener en cuenta, ya que generalmente si la comunidad que usa una tecnología o lenguaje de programación es grande, entonces más soporte o ayuda por parte de la comunidad puede recibir alguien que usa dicha tecnología.

Los números que se obtienen analizando a los más grandes sitios de preguntas y respuestas, *hosting* de repositorios y otros sitios se muestran en la tabla 3.1:

Tabla 3.1: Comparación del tamaño de la comunidad de Rails y Django.
Fuente: Elaboración propia.

Framework	**Preguntas en Stackoverflow [miles]**	**Projectos en GitHub [miles]**	**Seguidores en Quora [miles]**	**Seguidores en los subreddits respectivos [miles]**
Rails	352	256	132	21
Django	156	123	97	22

Con esto se puede concluir que la cantidad de contenido en la Internet sobre Rails es mayor que la de Django, pero aun así ambas comunidades son grandes y activas dentro de los portales de preguntas y respuestas como Stack Overflow o principales plataformas de desarrollo como GitHub.

En el tiempo de desarrollo ambos *frameworks* son parecidos según el estudio de Askins and Green (2006) siendo la creación de un sitio web estándar por medio de Django un poco más rápido. La diferencia se atribuye a que la creación del *dashboard* del administrador se hace de forma automática y nativa en Django. En cambio, en Rails esta tiene que ser creada a mano o por medio de alguna librería de tercero. Sin considerar este hecho, la diferencia entre los tiempos de desarrollo es marginal y perfectamente la experiencia de los desarrolladores puede explicar esto.

Hay que pensar esta interfaz se creará desde cero y sin basarse en otro *software* ya existente, por lo que el factor más importante en el tiempo de desarrollo debiese ser la experiencia que el desarrollador tenga con el *framework* escogido. Por esta razón se recomienda que la experiencia del desarrollador sea un factor importante a considerar al momento de elegir el *framework* para desarrollar el sistema.

Respecto a las librerías de terceros, ambos lenguajes, Ruby y Python poseen paquetes o librerías (en el caso de Ruby se les llama gemas) creados por terceros para agilizar tareas cotidianas o para implementar funcionalidades, por ejemplo, los mismos *frameworks* Django y Rails son librerías que se crearon para facilitar la creación de páginas web por medio de estos lenguajes.

En el caso de Ruby se tienen 138 mil librerías de terceros (RubyGems Collaboration, 2017) y 121 mil en el caso de Python (Python Software Foundation, 2017). Pero, aunque los números en este caso favorezcan levemente a Ruby, en el ámbito académico de computación científica y de la astronomía, Python es indiscutiblemente el preferido, ya que la comunidad está constantemente desarrollando y manteniendo librerías enfocadas a estas áreas, mientras que en Ruby el foco es el desarrollo web.

Capítulo 4 : Factibilidad y Validación de la Propuesta

4.1. Prototipo

Como parte de esta memoria también se consideró como una meta el desarrollo de un prototipo funcional que sirve para verificar el flujo del proceso, pero considerando una menor cantidad de formatos de archivos, productos de datos y servicios/protocolos de la IVOA soportados.

Específicamente el prototipo considera solamente el tipo de usuario administrador, es decir, con solo acceder a la aplicación web uno puede empezar el proceso de petición de subida de datos.

Este prototipo soporta la ingestión de catálogos que estén en formato FITS (*binary* o ASCII *table*) y el único protocolo que puede proveer es el del SCS, aparte de ADQL que siempre se considera activado en este caso.

4.1.1. Ambiente de desarrollo del prototipo

El desarrollo del prototipo de la interfaz se llevó a cabo tanto en el sistema operativo Lubuntu 17.10 como en Debian 8.9.

El *framework* usado para el desarrollo de la interfaz fue Ruby on Rails en su versión 5.1.4, en conjunto con la versión 2.4.1 de Ruby, aunque también se probó en la versión 2.5.1.

Por último, parte de la aplicación se desarrolló en Python, este funcionó correctamente tanto en las versiones 2.7.9 como 2.7.14.

4.1.2. Deployment

Para el *deployment* se definen los pasos a seguir para dejar el prototipo funcionando correctamente en ambiente de desarrollo, de esta forma se puede estudiar el código y ver la interfaz de forma conveniente.

Los pasos a seguir son instalar RVM, instalar ruby, clonar el repositorio del prototipo y ejecutarlo. Es necesario mencionar que para realizar estos pasos es recomendado no ejecutar los comandos con permisos de administrador, a menos que sea totalmente necesario.

4.1.2.1. Instalación del administrador de paquetes de ruby

El primer paso es instalar un administrador de ambientes de ruby, esto es un programa que nos ayudara a mantener diferentes versiones de ruby y diferentes conjuntos gemas instalados en la misma máquina sin generar conflictos. Esto es bastante útil si se tienen varios proyectos de Ruby o Rails, evitando problemas en caso de que estos ocupen diferentes versiones del interprete o de las librerías existentes. En este caso se usará RVM, pero también existen otros como Rbenv, chruby, etc.

Para instalar RVM se siguen los pasos listados en RVM Collaboration (2018):

El primer paso que se establece es descargar un par de claves GPG usadas para verificar la integridad del instalador como se ve en el siguiente código.

Código 14: Instalar clave GPG

```
gpg --keyserver hkp://keys.gnupg.net --recv-keys 409
    B6B1796C275462A1703113804BB82D39DC0E3 7D2BAF1CF37B13E2069D6956105BD0E739499BDB
```

Y para instalar RVM se ejecuta `curl -sSL https://get.rvm.io | bash` .

Para estar seguro que el comando anterior haya funcionado correctamente, se prueba el comando `rvm -v` . Si se visualiza como resultado la versión de RVM se instaló correctamente.

4.1.2.2. Instalación de ruby

A continuación, se instala el intérprete de ruby. En el desarrollo del prototipo se usa la versión 2.4.1, así que se debiese instalar esta versión de ruby ejecutando `rvm install 2.4.1` . Eso si dependiendo de la salida podría ser necesario realizar un par de pasos extras:

- Si al ejecutar `rvm install 2.4.1` no se solicita ninguna clave y se instala ruby correctamente se continua.
- El hecho de que al ejecutar `rvm install 2.4.1` aparezca una solicitud para que el usuario de Linux ingrese su contraseña, quiere decir que es necesario instalar algunas dependencias para el correcto funcionamiento de ruby, como se dijo anteriormente es sumamente recomendado no ingresar la contraseña, ya que al otorgar permisos de *root* a RVM para instalar estos programas los usuarios sin estos permisos tendrán problemas al ejecutar ruby y sus gemas. Por esto, se debe realizar los siguientes pasos:
  - Ejecutar los comandos `rvm autolibs read-fail` y `rvm install 2.4.1` . A continuación, debieran aparecer los paquetes requeridos para la instalación de ruby, se instalan de forma normal con `sudo apt-apt get install <packages_name>` .
  - Se ejecuta el comando `rvm autolibs reset` y se vuelve a ejecutar el comando `rvm install 2.4.1` , ahora como todas las dependencias deberían estar instaladas, RVM debería instalar ruby.

4.1.2.3. Clonación del repositorio del prototipo y su ejecución

El último paso es clonar el repositorio y ejecutar la aplicación web en ambiente de desarrollo, para esto basta `git clone https://github.com/oicitrapdraz/publish_data_vo.git` en la ruta que se desee dejar el proyecto, entrar a su carpeta y proceder a instalar las gemas que usa el proyecto. Eso si, antes de continuar es necesario especificarle a RVM qué versión de ruby y qué conjunto de gemas se usan en el proyecto, con este fin se ejecutan los siguientes comandos:

Código 15: Configurar RVM para nuestro proyecto

```
$ bash --login
$ rvm 2.4.1
$ rvm gemset create publish_data_vo
$ rvm gemset use 2.4.1@publish_data_vo --default
```

Con los comandos listados anteriormente se le especifíca a RVM que use la versión 2.4.1 de ruby, que cree el conjunto de gemas publish_data_vo y por último que deje esta versión de ruby (con este conjunto de gemas), por defecto.

Ahora se procede a instalar la gema *bundler* que permite instalar fácilmente las gemas que necesita el proyecto.

Código 16: RVM

```
$ gem install bundler
$ bundle install
```

Notar que al ejecutar estos comandos puede que ocurra un error con la instalación de la gema pg [20]. Para solucionar este error se ejecuta el comando `sudo apt-get install libpq-dev` y se habilita el servicio con `service postgresql start` . Finalmente se vuelve a ejecutar `bundle install` .

[20]Gema utilizada como interfaz del PostgreSQL RDBMS en ruby.

Ya en este punto se tiene todo lo necesario para ejecutar el proyecto, sólo falta la creación de su base de datos con el comando `$ rake db:create` , aunque primero se debe considerar que si ocurrió el error anteriormente descrito en el código 16 y por consiguiente se instaló *PostgreSQL* recientemente, lo más probable es que ocurra un error del tipo *PG::ConnectionBad: FATAL: role username does not exist*. Esto se debe a que, por defecto, PostgreSQL intenta conectarse a su servidor usando un rol que tenga el mismo nombre del usuario de Linux utilizado, mientras que al instalar PostgreSQL solo se crea el rol postgres.

En este caso la solución es sencilla, sólo basta ejecutar los comandos del código 20 para crear un nuevo rol de PostgresSQL:

Código 17: Creación de usuario d Postgres

```
$ sudo -u postgres psql postgres
$ CREATE USER username WITH CREATEDB;
$ \q
```

Donde username es el usuario del sistema.

Por último, sólo se debe crear las tablas en la base de datos definidas en el proyecto con `rake db:migrate` y `rails s` .

Al haber ejecutado todos estos comandos se puede acceder a la interfaz en la URL localhost:3000.

4.1.3. Lógica de negocio

Como se dijo anteriormente, el prototipo solo cubre algunas de todas las funcionalidades propuestas anteriormente. Éstas son:

- Solo existe el tipo de usuario administrador, por lo tanto, cualquier usuario que ingrese a la plataforma puede crear una petición para realizar una subida de datos sin haber iniciado sesión.

- Se considera únicamente el caso de la ingestión de archivos en formato FITS (*binary* o ASCII *table*).
- Solamente se consideran a los catálogos como productos de datos a subir.
- Solamente está disponible el uso del protocolo SCS, además de ADQQL, como protocolos de acceso a los datos.
- No se ingresan los datos de cobertura espacial, espectral ni temporal.

Además, existen archivos de texto usados para configurar el funcionamiento del prototipo:

- En el archivo dachs_directory se define la carpeta en la que se guardan los archivos FITS al ser subidos a la interfaz web. Por defecto se usa el directorio /var/gavo/inputs.
- En el directorio *vocabulary* se encuentra los archivos con los vocabularios controlados usados a la hora de elegir un tipo en los metadatos (*type*), elegir un tema en los metadatos (*subject*), elegir la banda de onda de cobertura (*coverage waveband*) y por último el tipo (*type*) de las columnas.

A continuación en la figura 4.1 se ve una vista de las funcionalidades que tiene el administrador como generar el *resource descriptor* y publicar los datos en el OV ejecutando comandos DaCHS desde la interfaz misma.

4.2. Requerimientos Adicionales Derivados de la Validación

El día 11 de Abril del 2018 se llevó a cabo una reunión con la Dra. Amelia Bayo (PhD en física de la Universidad Autónoma de Madrid), con el propósito de presentar el prototipo funcional de la interfaz propuesta y obtener apreciaciones tanto sobre éste, como del protocolo.

Esta actividad resultó en la proposición de 3 grandes sugerencias, una para la interfaz y 2 para el protocolo propuesto.

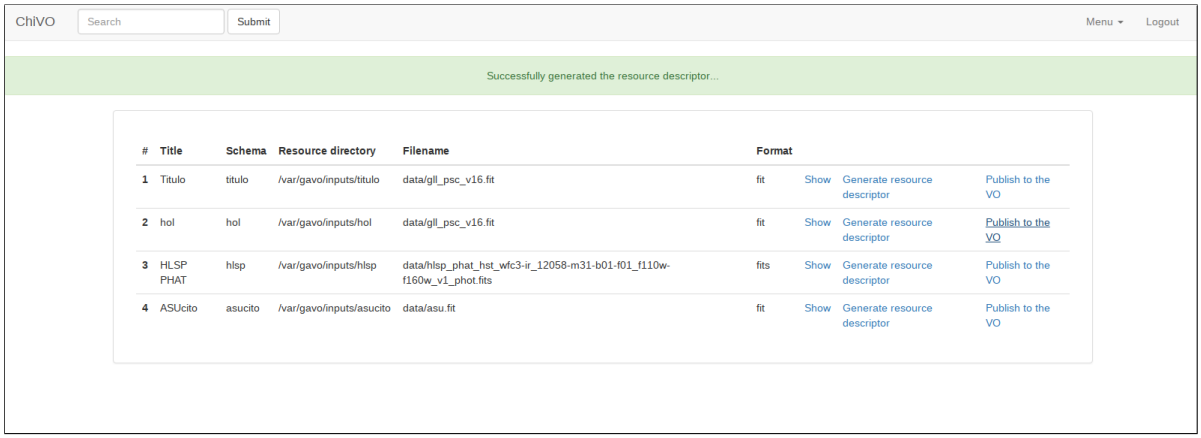


Figura 4.1: Vista del prototipo creado.
Fuente: Elaboración propia.

4.2.1. Requerimientos para la interfaz

Después de mostrar la interfaz del prototipo, se evidenciaron problemas de interpretación, por lo que se determinó que ésta tiene que ser sumamente clara y transparente para que los astrónomos sepan cuales etapas existen en el protocolo y qué datos son solicitados en cada una de ellas, de forma clara y concisa. Así se ahorran mal interpretaciones y se reduce enormemente la complejidad que un astrónomo tiene a la hora de usar la interfaz. En caso contrario incluso puede provocar que los astrónomos prefieran seguir usando los servicios actuales o que simplemente no publiquen sus datos en el Observatorio Virtual, lo cual no es positivo para el campo de la astronomía.

Para este fin se determinaron varios puntos que se deberán cumplir la interfaz propuesta:

- Se deberán usar nombres descriptivos y autoexplicativos para todos los campos de todos los formularios, además de textos de ayuda.
- Se debe contar con una barra de progreso que le indique al usuario en cada momento la etapa del proceso de subida de datos en la que se encuentra.
- Se deberá proveer varias secciones de ayuda, como FAQ (preguntas frecuentes), documentación que describa cada una de las tareas que se pueden realizar y guías mostrando, por medio de

ejemplos, como subir los datos de alto nivel a ChiVO.

Además, se deben llevar a cabo pruebas de usabilidad como se detalla en la sección 4.3.2 para detectar otros posibles problemas respecto a la facilidad de uso del sistema.

4.2.2. Requerimientos para el protocolo

Respecto al protocolo se determinaron dos cambios principales, la forma en que el astrónomo selecciona los datos a subir y el modo en que el astrónomo define el campo UCD de una columna.

Se determinó que la manera en que se propone que se seleccionen los datos a subir (ver en la sección 3.4.1.4) puede ser engorroso y tedioso para el astrónomo promedio. Es por esta razón que se propone que esta etapa se separe en las siguientes fases:

1. El astrónomo determinará las tablas que se publicarán (creándolas y asignándoles un nombre) y seleccionara los datos a subir en cada tabla, sin especificar sus metadatos.

2. El astrónomo especificará el resto de los metadatos (descripción, UCD, unidad, etc.) de cada una de las columnas seleccionadas de la etapa anterior.

De esta forma el astrónomo no se sentirá tan sobrecargado de información y formularios que rellenar de una sola vez.

Por otro lado, el modo en que el astrónomo define el campo UCD de una columna depende de dos casos:

- Cuando el dato proviene de un archivo FITS o VOTable, probablemente el UCD se encuentre en el mismo archivo, por lo que se podrá extraer y seleccionar de forma automática cuando el astrónomo especifique las columnas a publicar.

- En caso de que el archivo no sea de uno de estos formatos o aunque lo sea, no se encuentra el UCD, en vez de pedirle a éste que la escriba de forma directa (existe una cantidad extremadamente grande de posibles valores) se construirá el UCD mediante los siguientes pasos:

  1. Primero se seleccionará una de las posibles 12 categorías a las que pertenecen los UCD (IVOA, 2018d), a través de un *combobox*. Por ejemplo, *positional data*.
  2. Posteriormente se desplegarán en otro *combobox* los *tokens* o palabras válidas para esa categoría. Por ejemplo, *Longitude in galactic coordinates* [pos.galactic.lon]. Es importante que se muestre la descripción de estos valores, para la facilidad de uso del astrónomo.

  Con este proceso habremos elegido una palabra, pero es necesario tener en cuenta que un UCD puede estar compuesto de una o varias palabras separadas por punto y coma, por lo que se le tendrá que permitir al astrónomo repetir este proceso varias veces. Esto resultará en una o varias palabras unidas por el carácter, punto y coma, conformando nuestro UCD para la columna. Por ejemplo, stat.error;phot.mag;em.IR.H.

  Se debe tener en cuenta que existen restricciones a la hora de construir un UCD, por ejemplo, sólo algunas palabras pueden ser la primera, otras pueden ser solamente la segunda, etc.

4.3. Esquema de Validación Sistemática

En este capítulo se definen los criterios que se consideran para validar que el sistema fue correctamente desarrollado. Para este fin se realizarán pruebas funcionales y pruebas de usabilidad para asegurar las funcionalidades y la facilidad de uso del sistema.

4.3.1. Pruebas funcionales

Las pruebas funcionales buscan corroborar que el sistema haga lo que debiese hacer, para esto se definirán casos de prueba con diferentes requerimientos para validar las distintas funcionalidades del sistema.

Considerando lo anterior, se propone los siguientes pasos para la validación de la publicación de datos:

1. Conseguir al menos 6 conjuntos de datos, en donde como mínimo se tengan 2 por cada tipo de producto de dato (catálogo, imagen y espectro) y que por lo menos exista un archivo de cada formato aceptable (FITS, VOTable, CSV y TSV).

2. Realizar los procesos de subida y publicación de datos para cada conjunto de dato. Se tiene que probar al menos una vez cada uno de los servicios/protocolos seleccionables (SCS, ObsCore, ADQL, SSAP y SIAP).

3. Verificar las peticiones y publicar los datos por parte del administrador.

4. Por último, llevar un completo registro por cada caso de prueba, es decir:

   - Nombre, formato y enlaces de los archivos.
   - Número de registros que tienen los archivos [21].
   - Tipo de producto de datos y servicios/protocolos habilitados.
   - Columnas publicadas.
   - Columnas del ObsCore publicadas.
   - Comentarios y críticas del caso de prueba.

También se deben considerar las siguientes validaciones:

[21] Para determinar específicamente que datos se usaron para la prueba.

1. El astrónomo debiese poder salir de la interfaz y reanudar el proceso de publicación de datos (precargando los datos ya ingresados en la sección 3.4.1) por medio de la URL enviada a su correo.

2. Probar las funciones de mostrar, editar y eliminar una petición de publicación por parte de un administrador.

En caso de cubrir todas las expectativas y no encontrar errores, aceptar las pruebas de funcionalidad. En caso contrario, resolver la situación haciendo las mejoras y/o correcciones pertinentes.

4.3.2. Pruebas de usabilidad

Las pruebas de usabilidad tienen un enfoque de evaluar la facilidad de uso respecto a la interfaz visual de un sistema. Esto es un punto importante a considerar, ya que esta interfaz tiene que ser bastante intuitiva para que se tenga un valor agregado respecto a los otros servicios de publicación ya existentes.

Se propone el método de *expert review* que consiste en que gente con experiencia en el dominio realicen pequeñas tareas en el sistema, dándoles un intervalo de tiempo razonable para completarlas. Estas tareas no tienen que ser muy específicas ni se tiene que ayudar al usuario si este se encuentra en problemas para completarla, ya que el fin es determinar si una persona sin experiencia previa se desenvuelve de forma fluida en el sistema.

Al terminar el plazo para completar las tareas los astrónomos deberán evaluar el sistema llenando un formulario especificando si se cumplan las heurísticas de Nielsen, los cuales son principios de diseño que han sido usados desde hace varios años para medir la facilidad de uso de los sistemas informáticos (Nielsen, 1995).

Una propuesta para el cuestionario se muestra a continuación:

1. **Visibilidad del estado del sistema**: Después de realizar una acción, ¿el sistema informa lo

que ocurre?

Ejemplo: Al subir archivos se debería mostrar una barra de progreso.

Respuesta: _____

2. **Coincidencia entre el sistema y el mundo real**: ¿El sistema usa palabras, frases y conceptos familiares para el usuario, en lugar de términos técnicos?

   Ejemplo: El sistema debería usar iconos para ejecutar las acciones comunes, como el típico icono de un basurero para representar la eliminación de un recurso.

   Respuesta: _____

3. **Control de usuario y libertad**: ¿Accedió a funciones del sistema por error y tuvo que retroceder a menudo?

   Ejemplo: Si un usuario sube un archivo muy pesado por error, se debiese poder cancelar la subida.

   Respuesta: _____

4. **Consistencia y estándares**: ¿Se preguntó en algún momento si diferentes palabras o botones significaban lo mismo?

   Ejemplo: Cuando existe un botón para retroceder en una aplicación web puede crear confusión sobre si existe alguna diferencia con el botón para retroceder propio del *browser*.

   Respuesta: _____

5. **Prevención de errores**: ¿Existe un diseño cuidadoso que evite que ocurran errores?

   Ejemplo: Al llenar formularios se debiese advertir sobre campos obligatorios que no han sido llenados.

   Respuesta: _____

6. **Reconocimiento en lugar de recordar**: ¿El sistema minimiza la carga de memoria del usuario haciendo visibles objetos, acciones y opciones?

   Ejemplo: Usar un *combobox* con autocompletado en los formularios donde se ingresan opciones, en vez de apelar a la memoria del usuario.

   Respuesta: _____

7. Entregue sus propias apreciaciones, sugerencias o críticas respecto a la facilidad de uso de la interfaz web.

   Respuesta: _____

4.4. Estimación de Costos

Para la estimación del desarrollo de la interfaz propuesta se usará el método WBS, el cual está fuertemente relacionado al área de administración de proyectos y trata de dividir el trabajo completo en *items* cada vez más pequeños hasta poder estimar el trabajo realizado para cada *item*. Finalmente se calcula el trabajo total para realizar el proyecto sumando el trabajo necesario para realizar cada partición (U.S. Department of Defense Draft Military Standard, 2013).

Al aplicar WBS en un proyecto es necesario tener en cuenta un par de reglas:

- **La regla del 100 %**: Especifica que el 100 % del trabajo a realizar en el proyecto debe estar totalmente capturado en sus descomposiciones.
- **Elementos mutuamente excluyentes**: Es importante que cuando una tarea se divide, esta tenga componentes mutuamente excluyentes, en caso contrario al sumar los *items* más pequeños se estaría considerando más del 100 % del trabajo a realizar.

En nuestro caso lo más recomendable es usar el WBS para descomponer cada etapa del *System development life cycle* (SDLC), ya que esta división se produce de forma natural para los proyectos de desarrollo de *software*.

Se presenta a continuación la descomposición del trabajo.

1. Proyecto Interfaz web para la publicación de datos al OV
    1.1. Estudio preliminar
        1.1.1. Estudio del Observatorio Virtual
        1.1.2. Estudio de DaCHS
    1.2. Planificación
        1.2.1. Elaboración de la planificación para el desarrollo.
        1.2.2. Estudio de los diferentes *frameworks* para desarrollar la interfaz web
        1.2.3. Evaluación y elección del *framework*
    1.3. Análisis
        1.3.1. Definición de los requerimientos funcionales
        1.3.2. Definición de los requerimientos técnicos (SO, capacidad de Disco duro, etc.)
    1.4. Diseño
        1.4.1. Definición de la arquitectura del software
            1.4.1.1. Definir modelo de la base de datos
            1.4.1.2. Definir lógica de negocio
            1.4.1.3. Diseño del *layout* (*website wireframe*)
    1.5. Desarrollo de un prototipo funcional
    1.6. Implementación
        1.6.1. Desarrollo del registro e inicio de sesión
            1.6.1.1. Administrador
            1.6.1.2. Astrónomo
        1.6.2. Desarrollo del proceso de subida de datos por parte del astrónomo

1.6.2.1. Formulario para el llenado de metadatos y subida de archivos

1.6.2.2. Selección de los servicios disponibles

1.6.2.3. Selección de las columnas a mostrar

1.6.3. Desarrollo del proceso de publicación de datos por parte del administrador

1.6.3.1. Generación de RD

1.6.3.2. Verificación final

1.6.4. Interfaz para mostrar, editar y eliminar peticiones de publicación de datos

1.6.4.1. Interfaz para mostrar, editar y eliminar productos de datos

1.6.4.2. Interfaz para mostrar, editar y eliminar metadatos

1.6.4.3. Interfaz para mostrar, editar y eliminar columnas

1.6.5. *Dashboard* del astrónomo

1.6.5.1. Mostrar listado de peticiones de publicación de datos del usuario

1.6.6. *Dashboard* del administrador

1.6.6.1. Interfaz para mostrar y eliminar usuarios

1.6.6.2. Interfaz para aprobar y rechazar peticiones de publicación de datos

1.6.6.3. Visualizar y editar datos de la cuenta

1.7. Testing

1.7.1. Correcta visualización desde diferentes dispositivos y *browsers*

1.7.2. Pruebas de funcionalidad

1.7.3. Pruebas de usabilidad

1.7.4. Realizar mejoras y correcciones con el *feedback* de las pruebas

1.8. Deployment

1.8.1. Estudiar estructura de ChiVO

1.8.2. Realizar el *deployment*

A continuación, se presenta la estimación de las horas hombres para realizar las tareas en el nivel más bajo de descomposición, pero es necesario tener en cuenta que, parra esta estimación, se consideró que la persona a desarrollar el sistema ya tiene conocimientos básicos sobre el OV, los protocolos de la IVOA, los diferentes formatos de archivos usados comúnmente en la astronomía y el uso de DaCHS.

Cabe mencionar que se consideran como horas realizadas varias tareas que se hicieron al desarrollar esta memoria y que signifique directamente un ahorro de trabajo en el SDLC de la interfaz.

Tabla 4.1: WBS del proyecto de desarrollo de la interfaz.
Fuente: Elaboración propia.

Tareas	**HH estimadas**	**HH realizadas**	**HH restantes**
1.1.1. Estudio del Observatorio Virtual	24	0	24
1.1.2. Estudio de DaCHS	24	0	24
1.2.1. Elaboración de la planificación para el desarrollo	6	3	3
1.2.2. Estudio de los diferentes *frameworks* para desarrollar la interfaz web	6	3	3
1.2.3. Evaluación y elección del *framework*	1	0	1
1.3.1. Definición de los requerimientos funcionales	24	16	8
1.3.2. Definición de los requerimientos técnicos (SO, capacidad de Disco duro, etc.)	6	0	6
1.4.1.1. Definir modelo de la base de datos	4	3	1
1.4.1.2. Definir lógica de negocio	40	24	16
1.4.1.3. Diseño del *layout* (*website wireframe*)	6	0	6
1.5. Desarrollo de un prototipo funcional	90	90	0
1.6.1.1. Administrador	4	0	4
1.6.1.2. Astrónomo	12	0	12
1.6.2.1. Formulario para el llenado de metadatos y subida de archivos	6	0	6
1.6.2.2. Selección de los servicios disponibles	1	0	1
1.6.2.3. Selección de las columnas a mostrar	56	0	56
1.6.3.1. Generación del RD	60	0	60
1.6.3.2. Verificación final	4	0	4
1.6.4.1. Interfaz para mostrar, editar y eliminar productos de datos	3	0	3
1.6.4.2. Interfaz para mostrar, editar y eliminar metadatos	3	0	3
1.6.4.3. Interfaz para mostrar, editar y eliminar columnas	3	0	3
1.6.5.1. Mostrar listado de peticiones de publicación de datos del usuario	1	0	1
1.6.6.1. Interfaz para mostrar y eliminar usuarios	4	0	4
1.6.6.2. Interfaz para aprobar y rechazar peticiones de publicación de datos	4	0	4
1.6.6.3. Visualizar y editar datos de la cuenta	3	0	3
1.7.1. Correcta visualización desde diferentes dispositivos y *browsers*	1	0	1
1.7.2. Definición y ejecución de las pruebas de funcionalidad	24	4	20
1.7.3. Definición y ejecución de las pruebas de usabilidad	12	4	8
1.7.4. Realizar mejoras y correcciones con el *feedback* de las pruebas	16	0	16
1.8.1. Estudiar estructura de ChiVO	16	8	8
1.8.2. Realizar el *deployment*	8	0	8
Total	472	155	317

Capítulo 5 : Conclusiones

En el presente trabajo, se pudo formular un sistema para la ingestión de datos astronómicos de alto nivel a ChiVO y su publicación al Observatorio Virtual. Para la realización de esta tarea, se tuvo que considerar cuales eran los tipos de datos más comunes en el ámbito de la astronomía y considerar los diferentes casos en la construcción automática del *resource descriptor*, lo cual dependía de los tipos de datos a publicar. Esto no es una tarea sencilla, sobre todo por el grado de conocimiento requerido, tanto en el ámbito de la astronomía, como en en el dominio de la IVOA, además de esto se suma el hecho de que la documentación, tutoriales y ejemplos del uso de DaCHS en la Internet es limitada. Pero el desarrollo de esta interfaz significa un gran beneficio tanto a la comunidad de astrónomos, como para el Observatorio Virtual, ya que implica el desarrollar una forma en que astrónomos puedan publicar datos astronómicos de alto nivel, de forma rápida y sencilla.

Para la validación de esta propuesta se construyó un prototipo considerando un subconjunto de los requerimientos del protocolo, en otras palabras, considerando la aceptación de una menor cantidad de tipos de datos. Este prototipo fue enseñado a una astrónoma, con lo cual se tuvo la retroalimentación necesaria para la formulación de requerimientos adicionales que faciliten la tarea de la ingestión de los datos por parte del astrónomo, agregándole valor a la propuesta planteada.

Respecto a los posibles trabajos futuros, una clara posibilidad es la escalabilidad del sistema propuesto, contemplando la aceptación de otros tipos de productos de datos o formatos de archivos durante la ingestión para aumentar la generalidad de la propuesta, aunque el intentar una generalización desmedida de los casos de uso puede acarrear una complejidad excesiva de la interfaz y el

protocolo, ya que podría significar aumentar el número de comprobaciones que tenga que realizar el administrador o un aumento cantidad de elementos visuales en la interfaz que confundan al astrónomo.

Además de las dificultades técnicas que se puedan tener, también se hizo visible un problema respecto a los pocos incentivos que tienen los astrónomos para dejar sus datos a disposición de la comunidad científica. Ya que muy poca información se le entrega a los astrónomos sobre el uso de sus datos, por lo que seria ideal poder capturar más información sobre la utilización de estos y mostrar esta información en la interfaz propuesta, visibilizando la importancia que tuvo su publicación y como esta es accedida por la comunidad científica.

Bibliografía

ADS (n.d.). The ads data. http://adsabs.harvard.edu/abs_doc/help_pages/data.html. Acceso: 31-05-2018.

Askins, B. and Green, A. (2006). A rails / django comparison. pages 50 – 59.

Astropy Collaboration (2018). Astropy. https://github.com/astropy/astropy. Acceso: 03-01-2017.

Bonnarel, F., Fernique, P., Bienaymé, O., Egret, D., Genova, F., Louys, M., Ochsenbein, F., Wenger, M., and Bartlett, J. G. (2000). The aladin interactive sky atlas. *Astron. Astrophys. Suppl. Ser.*, 143:33 – 40.

ChiVO (2018). Chivo. https://vo.chivo.cl. Acceso: 27-05-2018.

Collaboration, A., Robitaille, T. P., Tollerud, E. J., p. Greenfield, Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A. M., Kerzendorf, W. E., Conley, A., Crighton, N., Barbary, K., Muna, D., Ferguson, H., Grollier, F., Parikh, M. M., Nair, P. H., Unther, H. M., Deil, C., Woillez, J., Conseil, S., Kramer, R., Turner, J. E. H., Singer, L., Fox, R., Weaver, B. A., Zabalza, V., Edwards, Z. I., Bostroem, K. A., Burke, D. J., Casey, A. R., Crawford, S. M., Dencheva, N., Ely, J., Jenness, T., Labrie, K., Lim, P. L., Pierfederici, F., Pontzen, A., Ptak, A., Refsdal, B., Servillat, M., and Streicher, O. (2013). Astropy: A community python package for astronomy. 558:9.

Demleitner, M. (2018a). The anatomy of the rd. http://docs.g-vo.org/DaCHS/tutorial.html#the-anatomy-of-the-rd. Acceso: 07-05-2018.

Demleitner, M. (2018b). *GAVO DC Software Reference Documentation*.

Demleitner, M., Neves, M. C., Rothmaier, F., and Wambsganss, J. (2014). Virtual observatory publishing with dachs. 7:27 – 36.

DSF (2018). Django. https://www.djangoproject.com. Acceso: 31-03-2018.

GAVO (2018). German astrophysical virtual observatory. http://www.g-vo.org. Acceso: 28-05-2018.

Hartl, M. (2012). *Ruby on Rails Tutorial: Learn Web Development with Rails*. Pearson Education, 2 edition.

HEASARC (2018). Cfitsio - a fits file subroutine library. https://heasarc.gsfc.nasa.gov/docs/software/fitsio/fitsio.html. Acceso: 27-05-2018.

Hubble European Space Agency Information Centre (2009). A short introduction to astronomical image processing. https://www.spacetelescope.org/projects/fits_liberator/improc. Acceso: 12-03-2018.

IAU FITS Working Group (2017). Flexible image transport system. https://fits.gsfc.nasa.gov. Acceso: 17-10-2016.

INAF (2014). Vo-dance. http://ia2.oats.inaf.it/vo-services/vo-dance. Acceso: 27-05-2018.

ISO (1988). Information processing – volume and file structure of cd-rom for information interchange. https://www.iso.org/standard/17505.html. Acceso: 23-03-2018.

IVOA (2009). Ivoat thesaurus. http://www.ivoa.net/rdf/Vocabularies/vocabularies-20091007/IVOAT/IVOAT.html. Acceso: 25-02-2018.

IVOA (2011). Simple spectral access protocol. http://www.ivoa.net/documents/SSA. Acceso: 27-05-2018.

IVOA (2013). Votable format definition. http://www.ivoa.net/documents/VOTable/20130920/REC-VOTable-1.3-20130920.html. Acceso: 21-03-2018.

IVOA (2014). Member organizations. http://www.ivoa.net/about/member-organizations.html. Acceso: 27-05-2018.

IVOA (2015). Ivoa simple image access. http://www.ivoa.net/documents/SIA. Acceso: 27-05-2018.

IVOA (2017). Observation data model core components and its implementation in the table access protocol. http://www.ivoa.net/documents/ObsCore/. Acceso: 25-04-2018.

IVOA (2018a). Astronomical data query language. http://www.ivoa.net/documents/latest/ADQL. Acceso: 27-05-2018.

IVOA (2018b). Simple cone search. http://www.ivoa.net/documents/latest/ConeSearch. Acceso: 27-05-2018.

IVOA (2018c). Simple cone search. http://www.ivoa.net/documents/latest/ConeSearch. Acceso: 27-05-2018.

IVOA (2018d). The ucd1+ controlled vocabulary. http://www.ivoa.net/documents/REC/UCD/UCDlist-20070402.html. Acceso: 14-05-2018.

Murray, C. A. (1989). The transformation of coordinates between the systems of b1950.0 and j2000.0, and the principal galactic axes referred to j2000.0. 218:325 – 329.

Nielsen, J. (1995). 10 usability heuristics for user interface design. https://www.nngroup.com/articles/ten-usability-heuristics. Acceso: 28-03-2018.

Peter, J., David, G., Istvan, C., Chenzhou, C., Francoise, G., Bob, H., Ajit, K., Chul, K. S., Andrew, L., Oleg, M., Masatoshi, O., Fabio, P., David, S., and Wolfgang, V. (2004). The international virtual observatory alliance: recent technical developments and the road ahead.

Python Software Foundation (2017). Pypi - the python package index. https://pypi.python.org/pypi. Acceso: 08-11-2017.

Reza, M. and Mardiyanto, M. (2014). Design of application framework for multi-tenant asset management system.

RubyGems Collaboration (2017). Stats. https://rubygems.org/stats. Acceso: 08-11-2017.

RVM Collaboration (2018). Installing rvm. https://rvm.io/rvm/install. Acceso: 14-04-2018.

Schwartz, E., Stephen, G., and Erez, N. (2012). Enhanced interferometric identification of spectra in habitable extrasolar planets. *The Astronomical Journal*, 144:71.

Shafranovich, Y. (2005). Common format and mime type for comma-separated values (csv) files. https://www.ietf.org/rfc/rfc4180.txt. Acceso: 09-10-2017.

Solar, M., Araya, M., Arévalo, L., Parada, V., Contreras, R., and Mardones, D. (2015). Chilean virtual observatory. pages 1 – 7.

SVO (2016). Svocat documentation. http://svo2.cab.inta-csic.es/vocats/SVOCat-doc/?what=intro. Acceso: 27-05-2018.

Taylor, M. (2014). Visualizing large datasets in topcat v4. 485:257 – 260.

Taylor, M. (2018). Starlink tables infrastructure library. http://www.star.bristol.ac.uk/~mbt/stil/. Acceso: 27-05-2018.

U.S. Department of Defense Draft Military Standard (2013). *Work Breakdown Structures for Defense Materiel Items*.

Wielen, R., Schwan, H., Dettbarn, C., and et al (2010). ARIHIP astrometric catalogue. VO resource provided by the GAVO Data Center.

A : Anexos

A.1. Tablas

A.1.1. Unidades en el OV

Tabla A.1: Unidades en el VO y sus sintaxis.
Fuente: M. Demleitner et al. (2005). *Units in the VO*. Retrieved from
http://www.ivoa.net/documents/VOUnits/index.html

unit	*description*	*fits*	*ogip*	*cds*	*vou*	*unit*	*description*	*fits*	*ogip*	*cds*	*vou*
%	percent			·		Jy	jansky	s	s	s	s
A	ampere	s	s	s	s	K	kelvin	s	s	s	s
a	julian year	s		s	s	lm	lumen	s	s	s	s
adu	ADU	·			s	lx	lux	s	s	s	s
Angstrom	angstrom	d		·	dp	lyr	light year	·	·		s
angstrom	angstrom		·		d	m	meter	s	s	s	s
arcmin	arc minute	·	·	·	s	mag	magnitudes	s	·	s	s
arcsec	arc second	·	·	s	s	mas	milliarcsecond	·		·	·
AU	astronomical unit	·	·	·	p	min	minute (time)	·	·	·	s
au	astronomical unit				·	mol	mole	s	s	s	s
Ba	besselian year	d			d	N	newton	s	s	s	s
barn	barn	sd	·	s	sd	Ohm	ohm	s		s	s

beam	beam	·			s	ohm	ohm		s		
bin	bin	·	·		s	Pa	s	s	s	s	s
bit	bit	s		s	sb	pc	s	s	s	s	s
byte	byte	s	·	s	sbp	ph	·				s
B	byte				sb	photon	photon	p	·		sp
C	coulumb	s	s	s	s	pix	pixel	·		·	s
cd	candela	s	s	s	s	pixel	pixel	p	·		sp
chan	channel	·	·		s	R	rayleigh	s			s
count	number	·	·		sp	rad	radian	s	s	s	s
Crab	crab		s			Ry	rydberg	·		s	s
ct	number	·		·	s	s	second (time)	s	s	s	s
cy	julian century	·				S	siemens	s	s	s	s
d	day	·	·	·	s	solLum	luminosity	·		·	s
dB					·	solMass	solar mass	·		·	s
D	debye	·		·	s	solRad	solar radius	·		·	s
deg	deg					sr	steradian	s	s	s	s
erg	erg	d	·		sd	T	tesla	s	s	s	s
eV	electron volt	s	s	s	s	ta	year tropical	d			d
F	farad	s	s	s	s	u	AMU	·			s
g	gramme	s	s	s	s	V	volt	·	·	·	·
G	gauss	sd	·		sd	voxel	voxel	·	·		s
H	henry	s	s	s	s	W	watt	s	s	s	s
h	hour	·	·	·	s	Wb	weber	s	s	s	s
Hz	hertz	s	s	s	s	yr	julian year	sp	·	sp	sp
j	joule	s	s	s	s						

A.1.2. Campos del modelo de datos ObsCore

Tabla A.3: Campos obligatorios del modelo de datos del *Observation Core Components*.
Fuente: Mireille Louys et al. (2016). *Observation Data Model Core Components and its Implementation in the Table Access Protocol*. Retrieved from http://www.ivoa.net/documents/ObsCore/

Column Name	Unit	Type	Description
dataproduct_type	unitless	String	Logical data product type (images, etc.)
calib_level	unitless	enum integer	Calibration level 0, 1, 2, 3, 4
obs_collection	unitless	String	Name of the data collection
obs_id	unitless	String	Observation ID
obs_publisher_did	unitless	String	Dataset identifier given by the publisher
access_url	unitless	String	URL used to access (download) dataset
access_format	unitless	String	File content format
access_estsize	kbyte	integer	Estimated size of dataset in kilo bytes
target_name	unitless	String	Astronomical object observed, if any
s_ra	deg	double	Central right ascension, ICRS
s_dec	deg	double	Central declination, ICRS
s_fov	deg	double	Diameter (bounds) of the covered region
s_region	unitless	String	Sky region covered by the data product (expressed in ICRS frame)
s_xel1	unitless	integer	Number of elements along the first spatial axis
s_xel2	unitless	integer	Number of elements along the second spatial axis
s_resolution	arcsec	double	Spatial resolution of data as FWHM
t_min	d	double	Start time in MJD
t_max	d	double	Stop time in MJD
t_exptime	s	double	Total exposure time.
t_resolution	s	double	Temporal resolution FWHM
t_xel	unitless	integer	Number of elements along the time axis
em_min	m	double	Start in spectral coordinates
em_max	m	double	Stop in spectral coordinates
em_res_power	unitless	double	Spectral resolving power
em_xel	unitless	integer	Number of elements along the spectral axis
o_ucd	unitless	String	UCD of observable (e.g. phot.flux.density, phot.count, etc.)
pol_states	unitless	String	List of polarization states or NULL if not applicable
pol_xel	unitless	integer	Number of polarization samples
facility_name	unitless	String	Name of the facility used for this observation
instrument_name	unitless	String	Name of the instrument used for this observation

A.1.3. Prefijos de las unidades físicas

Tabla A.4: Posibles prefijos de las unidades físicas mostradas.
Fuente: Elaboración propia.

da deca, 10^{1}	d deci, 10^{-1}	Ki kibi, 2^{10}
h hecto, 10^{2}	c centi, 10^{-2}	Mi mebi, 2^{20}
k kilo, 10^{3}	m mili, 10^{-3}	Gi gibi, 2^{30}
M mega, 10^{6}	u micro, 10^{-6}	Ti tebi, 2^{40}
G giga, 10^{9}	n nano, 10^{-9}	Pi pebi, 2^{50}
T tera, 10^{12}	p pico, 10^{-12}	Ei exbi, 2^{60}
P peta, 10^{15}	f femto, 10^{-15}	Zi zebi, 2^{70}
E hexa, 10^{18}	a atto, 10^{-18}	Yi yobi, 2^{80}
Z zetta, 10^{21}	z zepto, 10^{-21}	
Y yotta, 10^{24}	y yocto, 10^{-24}	

A.2. Listas

A.2.1. Posibles valores para el campo type de los metadatos

Lista A.1: Valores posibles en el campo *type* de los metadatos.
Fuente: M. Demleitner et al. (2005). *GAVO DC Software Reference Documentation*. Retrieved from http://docs.g-vo.org/DaCHS/ref.html

- Archive
- Bibliography
- Catalog
- Journal
- Library
- Simulation
- Survey
- Transformation
- Education
- Outreach

- EPOResource
- Animation
- Artwork
- Background
- BasicData
- Historical
- Photographic
- Press
- Organisation
- Project
- Registry

A.2.2. Parámetros de los mixins del obscore

Lista A.2: Parámetros del mixin obscore.
Fuente: M. Demleitner et al. (2018). *GAVO DC Software Reference Documentation*. Retrieved from http://docs.g-vo.org/DaCHS/ref.html

- **accessURL**: defaults to accref; URL at which the product can be obtained. Leave as is for tables mixing in products.
- **calibLevel**: defaults to 0; Calibration level of data, a number between 0 and 3; for details, see http://dc.g-vo.org/tableinfo/ivoa.obscore
- **collectionName**: defaults to 'unnamed'; A human-readable name for this collection. This should be short, so don't just use the resource title.
- **coverage**: defaults to NULL; A polygon giving the spatial coverage of the data set; this must always be in ICRS. This is cast to an pgsphere spoly, which currently means that you have to provide an spoly (reference), too.
- **creatorDID**: defaults to NULL; Global identifier of the data set assigned by the creator. Leave

NULL unless the creator actually assigned an IVO id herself.

- **dec**: defaults to NULL; Center Dec.
- **did**: defaults to $COMPUTE; Global identifier of the data set. Leave $COMPUTE for tables mixing in products.
- **emMax**: defaults to NULL; Upper bound of wavelengths represented in the data set, in meters.
- **emMin**: defaults to NULL; Lower bound of wavelengths represented in the data set, in meters.
- **emResPower**: defaults to NULL; Spectral resolution as lambda/delta lambda.
- **emUCD**: defaults to NULL; UCD of the spectral axis as defined by the spectrum DM, plus a few values defined in obscore 1.1 for Doppler axes.
- **emXel**: defaults to NULL; Number of samples along the spectral axis
- **expTime**: defaults to NULL; Total time of event counting. This simply is tMax-tMin for simple exposures.
- **facilityName**: defaults to NULL; The institute or observatory at which the data was produced.
- **fov**: defaults to NULL; Approximate diameter of region covered.
- **instrumentName**: defaults to NULL; The instrument that produced the data.
- **mime**: defaults to mime; The MIME type of the product file. Only touch if you do not mix in products.
- **oUCD**: defaults to NULL; UCD of the observable quantity, e.g., em.opt for wide-band optical frames.
- **obsId**: defaults to accref; Identifier of the data set. Only change this when you do not mix in products.
- **polStates**: defaults to NULL; List of polarization states present in the data; if you give something, use the convention of choosing the appropriate from I Q U V RR LL RL LR XX YY XY YX POLI POLA and write them in alphabetical order with / separators, e.g. /I/Q/XX/.

- **polXel**: defaults to NULL; Number of polarisation states in this product.
- **productSubtype**: defaults to NULL; File subtype. Details pending.
- **productType**: Data product type; one of image, cube, spectrum, sed, timeseries, visibility, event, or NULL if None of the above.
- **ra**: defaults to NULL; Center RA.
- **sPixelScale**: defaults to NULL; Size of a spatial pixel (in arcsec).
- **sResolution**: defaults to NULL; The (best) angular resolution within the data set, in arcsecs.
- **sXel1**: defaults to NULL; Number of pixels along the first spatial axis.
- **sXel2**: defaults to NULL; Number of pixels along the second spatial axis.
- **size**: defaults to accsize/1024; The estimated size of the product in kilobytes. Only touch when you do not mix in products#table.
- **tMax**: defaults to NULL; MJD for the upper bound of times covered in the data set. See tMin.
- **tMin**: defaults to NULL; MJD for the lower bound of times covered in the data set (e.g. start of exposure). Use ts_to_mjd(ts) to get this from a postgres timestamp.
- **tResolution**: defaults to NULL; Temporal resolution
- **tXel**: defaults to NULL; Number of samples along the time axis
- **targetClass**: defaults to NULL; Class of target object(s). You should take whatever you put here from http://simbad.u-strasbg.fr/guide/chF.htx.
- **targetName**: defaults to NULL; Name of the target object.
- **title**: defaults to NULL; A human-readable title of the data set.

A.2.3. Atributos del mixin ssap#mixc

Lista A.3: Atributos del mixin ssap#mixc
Fuente: M. Demleitner et al. (2018). *GAVO DC Software Reference Documentation*. Retrieved from http://docs.g-vo.org/DaCHS/ref.html

- **fluxSI**: defaults to __NULL__; SI conversion factor for fluxes in the spectrum instance (not the SSA metadata) in Osuna-Salgado convention; ssa:Dataset.FluxSI (you probably want to leave this empty)
- **fluxUCD**: defaults to phot.flux.density;em.wl; ucd of the flux column, like phot.count, phot.flux.density, etc. Default is for flux over wavelength; ssa:Char.FluxAxis.Ucd
- **fluxUnit**: Flux unit used by the spectra and in SSA char metadata. This must be a VOUnit string (use a single blank if your spectrum is not calibrated).
- **spectralSI**: defaults to __NULL__; SI conversion factor of frequency or wavelength in the spectrum instance (not the SSA metadata, they are all in meters); ssa:Dataset.SpectralSI (you probably want to leave this empty)
- **spectralUCD**: defaults to em.wl; ucd of the spectral column, like em.freq or em.energy; default is wavelength; ssa:Char.SpectralAxis.Ucd
- **spectralUnit**: Spectral unit used by the spectra (SSA char metadata always is wavelength in meters). This must be a VOUnit string (use a single blank if your spectrum is not calibrated).
- **timeSI**: defaults to __NULL__; SI conversion factor for times in Osuna-Salgado convention; ssa:DataSet.TimeSI (you probably want to leave this empty)

A.2.4. Campos del procedimiento setMeta

Lista A.4: Campos del procedimiento setMeta.
Fuente: M. Demleitner et al. (2018). *GAVO DC Software Reference Documentation*. Retrieved from http://docs.g-vo.org/DaCHS/ref.html

- **bandpassHi**: defaults to None; lower value of wavelength or frequency (you usually want to use //siap#getBandFromFilter to fill this).
- **bandpassId**: defaults to None; a rough indicator of the bandpass, like Johnson bands.
- **bandpassLo**: defaults to None; upper value of the wavelength or frequency (you usually want to use //siap#getBandFromFilter to fill this).

- **bandpassRefval**: defaults to None; characteristic frequency or wavelength of the exposure (you usually want to use //siap#getBandFromFilter to fill this).
- **bandpassUnit**: defaults to "m"; The unit of the bandpassRefval and friends (just don't touch this).
- **dateObs**: defaults to None; the midpoint of the observation; this can either be a datetime instance, or a float>1e6 (a julian date) or something else (which is then interpreted as an MJD).
- **instrument**: defaults to str(rd.getMeta(ïnstrument")); a short identifier for the instrument used.
- **pixflags**: defaults to None; processing flags (C atlas image or cutout, F resampled, X computed without interpolation, Z pixel flux calibrated, V unspecified visualisation for presentation only).
- **refFrame**: defaults to 'ICRS'; reference frame of the coordinates (change at your peril).
- **title**: defaults to None; image title. This should, in as few characters as possible, convey some idea what the image will show (e.g., instrument, object, bandpass).

A.2.5. Tipos de datos para una columna

Lista A.5: Tipo de dato de una columna.
Fuente: M. Demleitner et al. (2018). *GAVO DaCHS Tutorial*. Retrieved from http://docs.g-vo.org/DaCHS/tutorial.html

- **text**: Cadena de caracteres.
- **char**: Un solo carácter.
- **real**: Número real.
- **double precision**: Número de punto flotante.
- **integer**: Número entero de típicamente 32 bits.
- **bigint**: Número entero de típicamente 64 bits.
- **smallint**: Número entero de típicamente 16 bits.
- **timestamp**: Combinación de tiempo y fecha.
- **date**: Representa una fecha.

- **time**: Representa un tiempo.
- **box**: Representa un rectángulo.
- **spoint, scircle, sbox, spoly**: Objetos de geometría esférica.

A.3. Selección de Servicios/Protocolos

Dependiendo del tipo de producto de dato que represente un conjunto de datos, solo algunas opciones de servicios/protocolos estarán disponibles:

- Para un catálogo sólo se podrá habilitar SCS, ObsCore y ADQL.
- Para una imagen sólo se podrá habilitar SCS, ObsCore, ADQL y SIAP.
- Para un espectro sólo se podrá habilitar SCS, ObsCore, ADQL y SSAP.

Cada uno de estos protocolos tendrán los siguientes efectos en el conjunto de datos:

- **SCS** habilitará las consultas *Simple Cone Search*, lo que permite realizar búsquedas de registros mediante la definición de un cono en el cielo y los parámetros RA y DEC de los datos. Para proveer este servicio es necesario que el respectivo conjunto de datos contengan los campos RA y DEC en estándar ICRS o al menos J2000.
- **ObsCore** define que varios datos de los que serán publicados pertenecerán al ObsCore.
- **ADQL** permitirá realizar *queries* a los datos por medio del lenguaje ADQL.
- **SSAP** permitirá el acceso a espectros.
- **SIAP** permitirá el acceso a imágenes.

A.4. Procesamiento de los Datos Astronómicos

Considerar que para que el astrónomo pueda seleccionar las columnas de los archivos subidos, el sistema deberá analizar el contenido de estos y mostrarlas en la vista, permitiéndole al astrónomo elegir los campos a publicar.

Dependiendo del formato del archivo esto se puede hacer de distintas formas:

- Para archivos en formato VOTable, al tratarse de archivos en XML, se tendrá que retornar las columnas por medio de XPath [22].
- Para archivos en formato FITS se tendrá que retornar las *keys* tanto de las cabeceras como de los datos de los HDU especificados en 3.4.1.2.
- Para archivos en formato CSV y TSV el proceso es un poco distinto ya que existe un paso adicional antes de seleccionar los datos a publicar. Esta etapa consta de que el astrónomo debiese ingresar el nombre de todas las columnas de cada archivo CSV o TSV, considerando también el orden correcto en que aparecen estas las columnas y posteriormente el astrónomo podrá seleccionar las columnas que desea publicar. Esta forma de realizar el proceso se debe a que no existe un formato estándar de CSV o TSV para manejar datos astronómicos, por lo que no se puede crear un programa que retorne el nombre de las columnas en un archivo con estos formatos.

A.5. Función y Estructura General de un *resource descriptor*

Para la publicación de datos al Observatorio Virtual mediante DaCHS es necesario la creación de un archivo de configuración llamado *resource descriptor*. Este archivo que generalmente tiene el nombre de q.rd define todo el proceso de ingestión y publicación para un conjunto de datos en

[22] Lenguaje para acceder a los elementos de un documento XML

específico. La ingestión y publicación, en resumen, es el procedimiento en que DaCHS ingresa a su base de datos los datos astronómicos que se pueden encontrar en archivos de texto, FITS, VOTable, etc., y que estos sean publicados en el OV.

Generalmente los *resource descriptors* tienen la siguiente estructura:

- `resource` : Esta es el elemento raíz de un q.rd. Su función principal es definir el esquema usado por la base de datos de DaCHS para la ingestión del conjunto de datos.

  Dentro del elemento `resource` se tienen los siguientes contenidos:

  - **Definición de metadatos**: Los metadatos se declaran en los elementos `meta` y describen al conjunto de datos en si. Por ejemplo, en ellos se definen los nombres de los autores, instrumento utilizado, observatorio en el que se obtuvieron los datos, etc.
  - **Definición de tablas**: Se puede observar por los elementos `table` . Estos detallan la creación de las tablas en la base de datos de DaCHS para la ingestión de los datos que se encuentran en los archivos FITS, VOtable, etc. Cabe destacar que dentro de un q.rd puede existir la definición de un número indeterminado de tablas.
  - **Mapeo de datos**: Esta parte a la definición de los elementos `data` , en los cuales se especifican como realizar la ingestión de los datos astronómicos, que se encuentran en los archivos FITS, VOTable, etc., hacia las tablas definidas con los elementos `table` .
  - **Definición de servicios**: Detalla el como se publicaran las tablas después de realizar la ingestión y publicación de los datos, por medio de los elementos `service` . Específicamente estos elementos indican los protocolos que soportan, *inputs*, *outputs*, etc.

La definición explicada anteriormente se puede ver representada en el código 18, en donde se visualiza que la definición de los metadatos, tablas, mapeos y servicios se realizan mediante los elementos hijos del elemento raíz, `resource` .

En base a lo anteriormente mencionado, se puede observar el código 18, el cual representa, a grandes rasgos, la estructura general/básica de un *resource descriptor*.

Código 18: Estructura de un *resource descriptor*

```
<resource>
        <table>
        .
        .
        .
        </table>

        <data>
        .
        .
        .
        </data>

        <service>
        .
        .
        .
        </service>
</resource>
```

A.6. Ejemplo de Construcción de un *resource descriptor*

Para mostrar el potencial de DaCHS, se realizará el siguiente ejemplo, en donde se genera el RD para la publicación de un conjunto de datos. Tener en cuenta que este ejemplo representa en lineas generales la construcción de este archivo, ya que se omiten muchos detalles.

Consideremos el caso en que un astrónomo desea ingresar los siguientes datos que se tienen en un archivo llamado asu.txt en formato CSV, donde se tienen los datos de 6 objetos, separados por las

columnas, ID (identificador), _RAJ2000 (ascensión recta) Y _DEJ2000 (declinación).

```
ID,_RAJ2000,_DEJ2000
----,----------,----------
1047,323.354001,-00.879194
1221,323.341199,-00.877481
1249,323.353065,-00.877276
1921,323.345946,-00.871682
1927,323.361530,-00.871631
2288,323.356795,-00.868873
```

Para que este ejemplo no sea tan extenso se omite el código correspondiente a la definición de los metadatos y se comienza con la definición de la tabla que se usará para almacenar los datos a publicar. Esto se declara con el elemento `table` , en el cual se definen columnas dependiendo de los datos que uno desea ingresa. Como en nuestro caso publicaremos todas las columnas del archivo definiremos 3 columnas dentro de la tabla, todo esto se puede visualizar en el código 19.

Notar que se proveen datos descriptivos (metadatos) para cada una de las columnas, como el nombre, descripción, tipo de dato, unidad, ucd, etc.

Código 19: Elementos columns

```
<table id="main" onDisk="True" adql="True">
  <column name="hr" description="Resource id" type="integer" ucd="meta.id;meta.main"
      verbLevel="30" required="True"/>
  <column name="_ra" description="Right ascencion" type="double precision" ucd="pos.eq.
      ra;meta.main" unit="deg" verbLevel="1" required="True"/>
  <column name="_de" description="Declination" type="double precision" ucd="pos.eq.dec;
      meta.main" unit="deg" verbLevel="1" required="True"/>
</table>
```

Y a continuación viene la definición de la ingestión de los datos a las columnas definidas anteriormente, para esto se utiliza el elemento `data` que se ve en el código 20.

Código 20: Elemento data

```
<data id="import">
  <sources>data/asu.txt</sources>
  <csvGrammar strip="True" topIgnoredLines="3">
    <names>id,raj2000,dej2000</names>
  </csvGrammar>
  <make table="main">
    <rowmaker>
      <map dest="_ra">@raj2000</map>
      <map dest="_de">@dej2000</map>
      <map dest="hr">@id</map>
    </rowmaker>
  </make>
</data>
```

El elemento `sources` define los archivos usados en la ingestión, generalmente estos se encuentran dentro de la carpeta *data* del *resource directory*. En `csvGrammar` se especifica el nombre las columnas que tiene el archivo de texto, respetando su orden (en el caso de usar archivos FITS o VOTable se usan otros *grammars*, en los cuales no es necesario especificar las columnas que tienen los archivos). En `make` se especifica la tabla en la que se realiza la ingestión (en este caso es la tabla *main*) y en `rowmake` se definen como se ingresaran los datos (desde el archivo de texto asu.txt) a las columnas de la tabla. Las columnas objetivo se especifican con el atributo `dest` del elemento `map` , y con el carácter @ se hace una referencia a las columnas del archivo. En otras palabras:

- Los datos de la columna raj2000 serán ingresados a la tabla _ra.
- Los datos de la columna dej2000 serán ingresados a la tabla _de.

- Los datos de la columna id serán ingresados a la tabla hr.

Por último, se deben definir los servicios para poder acceder a los datos ingresados a la base de dato de DaCHS, lo cual se hace con el elemento `service` . Dentro de este existen muchos elementos hijos y atributos, pero básicamente todos ellos apuntan a detallar que tipos de protocolos soportarán las tablas, que *inputs* u *outputs* tendrán, etc.

Continuando con el ejemplo en el código 21 se define, por medio del elemento `inputKey` , que el servicio tendrá como *input* la columna `hr` , además con `defaultRenderer` se define que, por defecto, se usara un formulario web para acceder a los datos.

Código 21: service

```
<service id="get_data" defaultRenderer="form" allowed="form,static">
  <meta name="title">Get Data</meta>
  <meta name="shortname">gd</meta>
  <dbCore queriedTable="main">
  <condDesc>
    <inputKey original="hr" required="True"/>
  </condDesc>
  <outputTable verbLevel="15"/>
  </dbCore>
</service>
```

Entonces al ejecutar los comandos DaCHS para realizar la ingestión y publicación, se podrá acceder al servicio web de DaCHS para interactuar con los datos ingresados. Como se puede ver en la figura A.1 y de acuerdo al código del servicio, se creo un formulario web en el que se espera que el usuario ingrese un valor para la columna `hr` .

En la figura A.2 se puede ver que al ingresar el valor de 1047 se obtiene como resultado los datos RA y DEC para el registro con ID = 1047 (recordar que la columna `hr` almacena el valor de la columna ID en el archivo de texto).

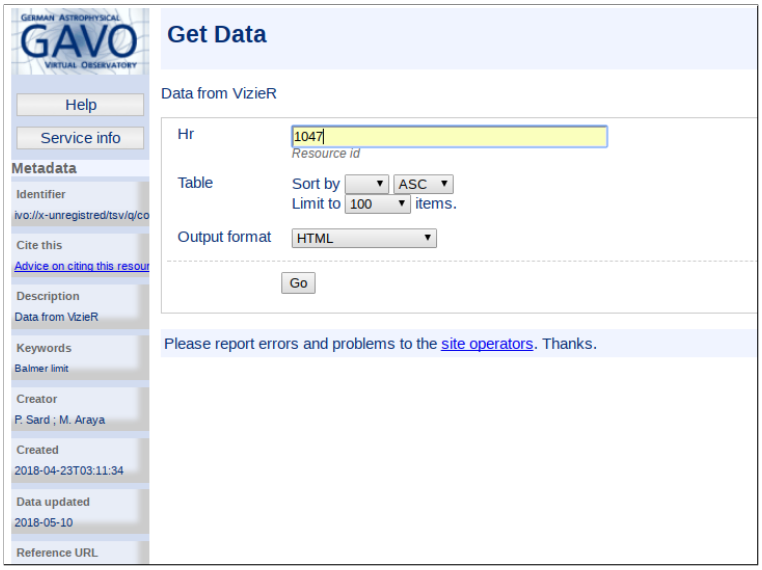


Figura A.1: Formulario creado por DaCHS.
Fuente: Retrieved from https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html

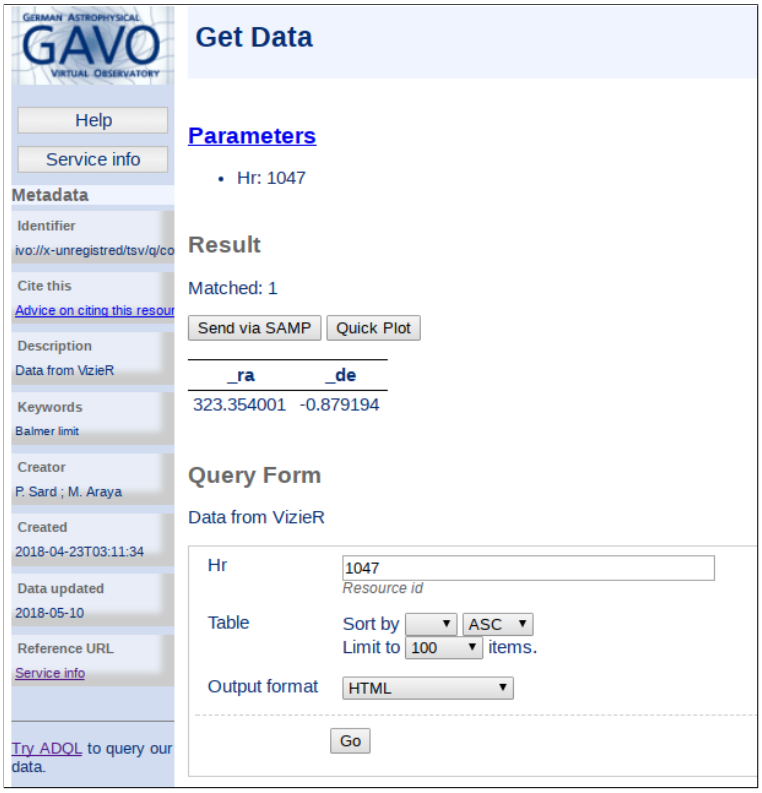


Figura A.2: Resultados de la consulta realizada a los datos.
Fuente: Retrieved from https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html