FONDEF D11I1060

Primera Etapa de Implementación

Mauricio Solar, Marcelo Mendoza, Jonathan Antognini, Diego Mardones,
Karim Pichara, Ricardo Contreras, Victor Parada.

Valparaíso, 30 de junio de 2014

Resumen

El presente documento detalla la primera etapa de implementación: Implementación de modelo de datos y capa de acceso de datos en aplicación web, según los estándares y protocolos de IVOA.

Palabras Claves: ChiVO, Capa de Datos, DaCHS, Saada, VODance, Tecnologías, Herramientas.

1. Introducción

En el marco de la creación de una "Plataforma astroinformática para la administración y análsis inteligente de datos a gran escala", se dio comienzo al desarrollo del Chilean Virtual Observatory (ChiVO).

En base a los casos de uso y requerimientos capturados para el primer hito, las universidades participantes del proyecto han estado trabajando en la implementación de estos.

El presente documento detalla:

- Base de datos: Tecnologías, modelos de datos, formatos y desarrollo actual de la capa de base de datos de la arquitectura de ChiVO.
- Aplicación: Tecnologías estudiadas y resultados actuales. Además una subsección dedicada al estudio y comparativa entre los distintos toolkits de data access layer recomendados por IVOA.
- Interfaz de usuario: Implementaciones actuales y capturas de pantalla del actual sistema.

2. Base de datos

Existen distintos motores de base de datos (RMDBS), sin embargo casi todos los Toolkits de acceso a datos usan como motor PostgreSQL, debido a que las implementaciones de los protocolos DAL se basan en el uso de pgSphere. Por ende no hubo posibilidad de elegir el RMDBS. Uno de los toolkits recomendados por IVOA permite trabajar con MySQL, sin embargo aún no cuenta con version estable.

2.1. Modelos de datos y formatos

ALMA posee sus propios formatos de datos, según lo que en un principio fueron sus necesidades. Entre ellos se encuentran:

- **ALMA Science Data Model**: define una colección de información almacenada durante la observación, la cual es necesaria para realizar análsis científicos. Está compuesto por un core de 16 tablas que son usadas en todos los modos de observación; y otras 23 tablas disponibles dependiendo el contexto de observación. El subsistema de calibración del telescopio define sus propias tablas adicionales (una por tipo de calibración). En términos prácticos, un ASDM define metadata en archivos XML según un esquema de dependencias[1], la cual anexa links a la data binaria.
- **Measurement Set**: mediante una función del paquete de software CASA se obtiene a partir de un ASDM, y define una tabla principal y un set de tablas secundarias. Este formato de dato es utilizado para procesamiento de data multidimensional.
- **FITS**: uno de los productos del procesamiento de datos son los archivos FITS, sin embargo el pipeline actual de ALMA, no provee ni metadata ni data binaria estándar como para trabajar con este formato.

En vista de los metadatos y datos binarios producidos por el observatorio ALMA se propuso realizar un mapping entre los campos mandatorios del modelo de datos recomendado por IVOA (Observation Core Data Model), con el ASDM. Los resultados de este proceso fueron:

[1] http://csrg.cl/ jantogni/ASDMTableDependencies_v3.jpg

ObsCore	**ASDM**
dataproduct_type	visibility
calib_level	1
obs_collection	ALMA
obs_id	ExecBlock.execBlockUID
obs_publisher_did	DaCHS automáticamente lo asigna con products#define
access_url	DaCHS automáticamente lo asigna con products#define
access_format	DaCHS automáticamente lo asigna con products#define
access_estsize	DaCHS automáticamente lo asigna con products#define
target_name	Scan.sourceName -¿Source.sourceName o Scan.fieldName -¿Field.sourceId -¿Source.sourceName
s_ra	Scan.fieldName -¿Field.sourceId -¿Source.direction[0] ó Scan.sourceName -¿Source.direction[0]
s_dec	Scan.fieldName -¿Field.sourceId -¿Source.direction[1] ó Scan.sourceName -¿Source.direction[1]
s_fov	Pendiente
s_region	circle
s_resolution	1,2 * $\frac{lambda}{ExecBlock.baseRangeMax}$
t_min	Scan.execBlockId -¿ExecBlock.startTime
t_max	Scan.execBlockId -¿ExecBlock.endTime
t_exptime	Para todos los SubScan.subscanIntent = ON_SOURCE sumar (SubScan.endTime - SubScan.startTime) el de cada uno
t_resolution	Pendiente
em_min	Scan.execBlockId -¿ExecBlock.baseRangeMini
em_max	Scan.execBlockId -¿ExecBlock.baseRangeMax
em_res_power	Spectral_Window.name = ...#FULL_RES Spectral_Window.resolution
o_ucd	em.mm
pol_states	Pendiente
facility_name	ALMA
instrument_name	ALMA
freqency	SBSummary.frequency

Para obtener cada atributo se realizó un programa en Java usando la biblioteca creada por la NRAO [2]. Este paquete permite parsear un ASDM en estructura de datos fácil de manejar desde el punto de vista de programación y respeta las relaciones del esquema de dependencias.

[2]http://csrg.cl/ jantogni/ASDM-standalone.tar.bz2

3. Aplicación

3.1. Endpoint

Los frameworks que se evaluaron para la implementación del endpoint fueron:

- Ruby on Rails: RoR es uno de los frameworks de desarrollo web más usados actualmente, su uso es mediante el concepto Modelo-Vista-Controlador. El endpoint es un webservice, por lo que RoR como framework completo MVC, poseía características que no serían utilizadas, como por ejemplo las vistas usuarios. Por esta razón no se utilizó este framework.
- Python/Flask: Flask es un microframework diseñado especialmente para hacer webservices y herramientas web pequeñas. Lo que provee esta biblioteca es un marco de trabajo para la creación de aplicaciones web que puedan ser accedidas mediante distintos métodos HTTP. Existe mucha documentación y comunidad activa que permite implementar y solucionar problemas de forma rápida. Finalmente se inclinó por esta alternativa por ser un framework más a la medida de lo necesario por implementar.

Los avances actuales del Endpoint contemplan:

- Capa de acceso al registro VO-Paris, el cual es un registro de catálogos disponibles para los usuarios, y que permiten consultas mediante SCS/SSA/SIA/TAP. Este sistema será utilizado por la interfaz usuario para realizar búsquedas en distintos catálogos astronómicos.
- Se comenzó a desarrollar un SESAME ChiVO, sistema con el cual se puede identificar las coordenadas espaciales de un objeto astronómico mediante su nombre. Actualmente en la plataforma se está usando el SESAME de AstroGrid.

3.2. ALMA Resource

Dentro de los toolkits de DAL recomendados por IVOA, se testearon los siguientes:

- SAADA: Desarrollado por el VO Frances, es una herramienta bastante útil del punto de vista usuario, posee excelente documentación y manual de instalación, incluso la instalación se mediante GUI. Está desarrollado en Java y su correspondiente deployment se hace usando Tomcat. Tienen una versión abierta del código en googlecode. Es posible configurar servicios SCS/SIA/SSA/TAP.
- VO-Dance: Desarrollado por el VO Italiano, es una herramienta en Java en su Backend, y Python en su Frontend (Framework Django). Lo destacable de esta herramienta es que trabaja usando MySQL como motor de base de datos principal, y según lo conversado con los desarrolladores están probando PostgreSQL actualmente. La herramienta no es OpenSource y la documentación de instalación y configuración es básica, ya que aún continúa en desarrollo. SCS/SIA/SSA/TAP.
- openCADC: Desarrollado por el VO Canadiense, es una herramienta OpenSource escrita en Java, utilizada actualmente en el ALMA Science Archive. Este toolkit es uno de los más robustos, contiene distintos paquetes con servicios a ser utilizados en el webservice, sin embargo no existe documentación de instalación y configuración, y para poder probarlo fue necesario contactar directamente al desarrollador principal. Es posible configurar servicios TAP.
- DaCHS: Desarrollado por el VO Alemán, es una herramienta OpenSource escrita en Python. Es uno de los toolkits DAL más usados por los VO, ya que posee una amplia documentación de instalación y configuración. Es posible configurar servicios SCS/SIA/SSA/TAP.

El resumen de los toolkits en una tabla comparativa es:

Toolkits	Lenguaje	OpenSource	Documentación	Servicios	Último update
SAADA	Java	Si	Si	SCS/SIA/SSA/TAP	Mayo 2014
VO-Dance	Java/Python	No	No	SCS/SIA/SSA/TAP	Diciembre 2012
openCADC	Java	Si	No	TAP	—
DaCHS	Python	Si	Si	SCS/SIA/SSA/TAP	Junio 2013

En base a estos factores, el Toolkits DAL elegido fue DaCHS. En base a lo planteado en la base de datos, y el funcionamiento de DaCHS, los archivos de configuración usados se encuentran en github[3]. Por ejemplo un Simple Image Access Protocol sería como:

```
<?xml version="1.0" encoding="iso-8859-1"?>

<resource schema="siapobsexample">
  <meta name="title">SIAP ObcCoreDM Examples</meta>
  <meta name="creationDate">2013-11-26T14:59:00Z</meta>
  <meta name="description" format="plain">
    LIRAE GROUP
    SIAP ObsCoreDM Ejemplo
    Probar implementar una tabla Simple Image Access Protocol, junto con una tabla ObsCore
  </meta>
  <meta name="copyright">Free to use.</meta>
  <meta name="creator.name">Patricio Ramirez</meta>
  <meta name="subject">Ejemplo</meta>
  <meta name="facility">The Dispair Telescope</meta>

  <meta name="coverage">
    <meta name="waveband">Optical</meta>
  </meta>
  <meta name="content.type">Catalog</meta>

    <table id="spe" onDisk="True" adql="True">
        <mixin>
            //siap#pgs
        </mixin>
        <mixin
            collectionName="'LIRAE GROUP'"
            ra="centerAlpha"
            dec="centerDelta"
            title="imageTitle"
            sResolution="0.5"
            facilityName="'ALMA'">
            //obscore#publishSIAP
        </mixin>

        <meta name="description">
          Ejemplo SIAP+ObsCore.
        </meta>

        <!--DESCRIPTION-->
        <!--SIAP columns-->
```

[3]https://github.com/paramire/LIRAE-DaCHS.git

```
      <column name="centerAlpha"
        ucd="pos.eq.ra;meta.main"
        tablehead="RA"
        description="Area center RA ICRS"
        verbLevel="10"/>
      <column name="centerDelta"
        ucd="pos.eq.dec;meta.main"
        tablehead="Dec"
        description="Area center Declination ICRS"
        verbLevel="10"/>

  </table>

<rowmaker id="build_spe">
  <var name="imageTitle">
      "%s %s"%(@TELESCOP,@DATE_OBS)
  </var>
  <apply procDef="//siap#computePGS"/>
      <apply procDef="//siap#setMeta">
          <!-- Falta como completar info -->
          <bind name="title">vars["imageTitle"]</bind>
          <bind name="instrument">vars["TELESCOP"]</bind>
  </apply>
  <map dest="centerDelta">@OBSDEC</map>
  <map dest="centerAlpha">@OBSRA</map>
</rowmaker>


  <data id="import_content">
          <!--RECURSOS FITs-->
          <sources recurse="True" pattern="res/*.fits">
          </sources>

          <!-- qnd: Hack para hacer fitsProdGrammar mas rapido-->
          <fitsProdGrammar qnd="True">

              <!-- Al utilizar productos es necesario //products#define, nos
              agrega ciertas culmanas a la tabla-->
              <rowfilter procDef="__system__/products#define">
                      <bind key="table">"siapobsexample.spe"</bind>
              </rowfilter>
          </fitsProdGrammar>

      <!--SERVICIO TAP-->
      <register services="__system__/tap#run"/>
      <!--Crea tabla en la DB -->
      <make table="spe" rowmaker="build_spe"/>
  </data>
```

```
    <!--Servicio, Publicación-->
    <service id="sia" allowed="form,siap.xml">
        <meta name="shortName">SIAP EXAMPLE</meta>
        <meta name="title">"Sample image access"</meta>
        <meta name="testQuery.pos.ra">230.444</meta>
        <meta name="testQuery.pos.dec">52.929</meta>
        <meta name="testQuery.size.ra">0.1</meta>
        <meta name="testQuery.size.dec">0.1</meta>
        <meta name="sia.type">Pointed</meta>
        <publish render="siap.xml" sets="local"/>
        <publish render="form" sets="local" />

        <!-- CORES -->
            <dbCore id="query_images" queriedTable="spe">
              <condDesc original="//siap#protoInput"/>
              <condDesc original="//siap#humanInput"/>
            </dbCore>
    </service>
</resource>
```

4. Interfaz Usuario

Inicialmente la interfaz usuario o frontend iba a contener solo vistas, por lo que el desarrollo podía ser en prácticamente cualquier lenguaje o framework, como por ejemplo HTML, PHP, Django o RoR. Sin embargo con los requerimientos de la plataforma, especialmente el de capa de usuarios, se decidió inclinarse por un framework MVC que fuese lo suficientemente ágil y compatible con el resto de servicios, por lo que se eligió RoR.

Por el momento se están realizando pruebas conceptuales de:

- Integración con el SESAME de AstroGrid.
- Consulta por múltiples fuentes.
- Consultas a distintos catálogos, desplegando los resultados en paralelo.

Los avances están disponibles en la plataforma beta.chivo.cl
Algunos screenshots:

5. Siguientes Pasos

Durante los próximos meses se espera desarrollar:

- Base de datos: como ya fueron identificados los metadatos, ahora es necesario poblar la base de datos tanto con metadatos como binarios, lo que significa: elegir el tipo de data que se va a publicar y crear los Resource Descriptor para DaCHS.
- Aplicación: Terminar el SESAME ChiVO; Integración entre DaCHS y Flask para proveer serivicios a la interfaz de usuarios en la consulta de catálogos múltiples.
- Interfaz de Usuario: crear vistas según los requerimientos de los astrónomos y crear una forma inteligente de consultar la base de datos directamente a través del Table Access Protocol.

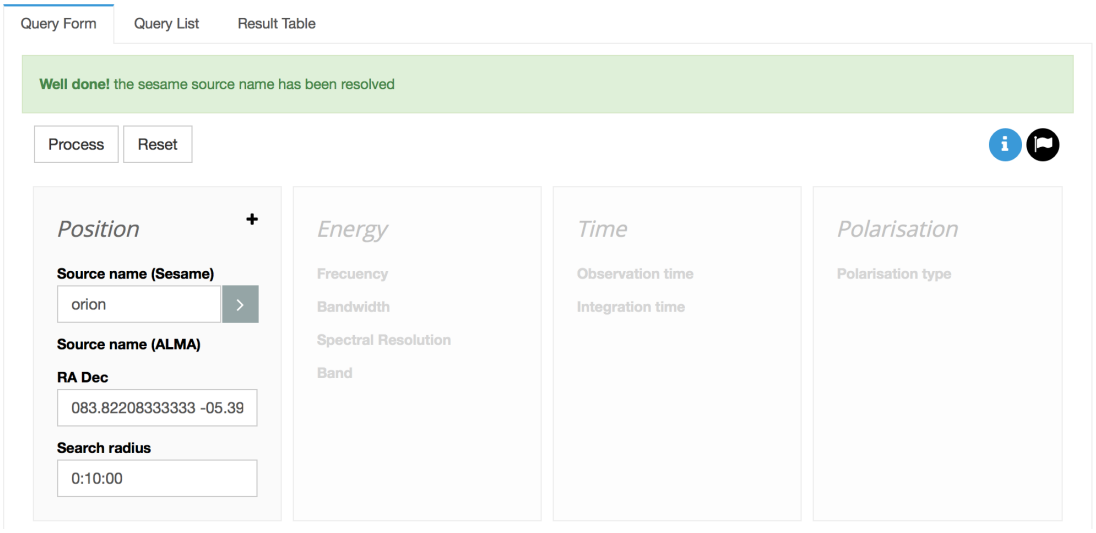

Figura 1: Resolvedor de nombres

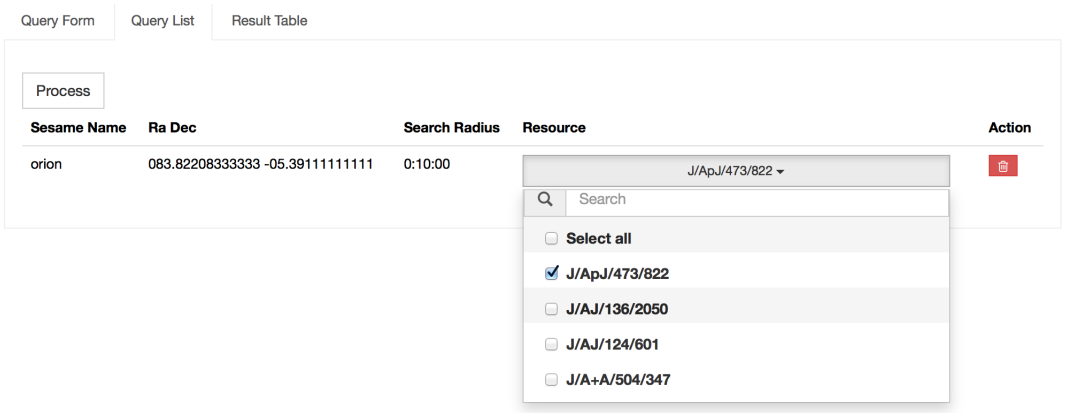

Figura 2: Búsqueda Múltiples Catálogos

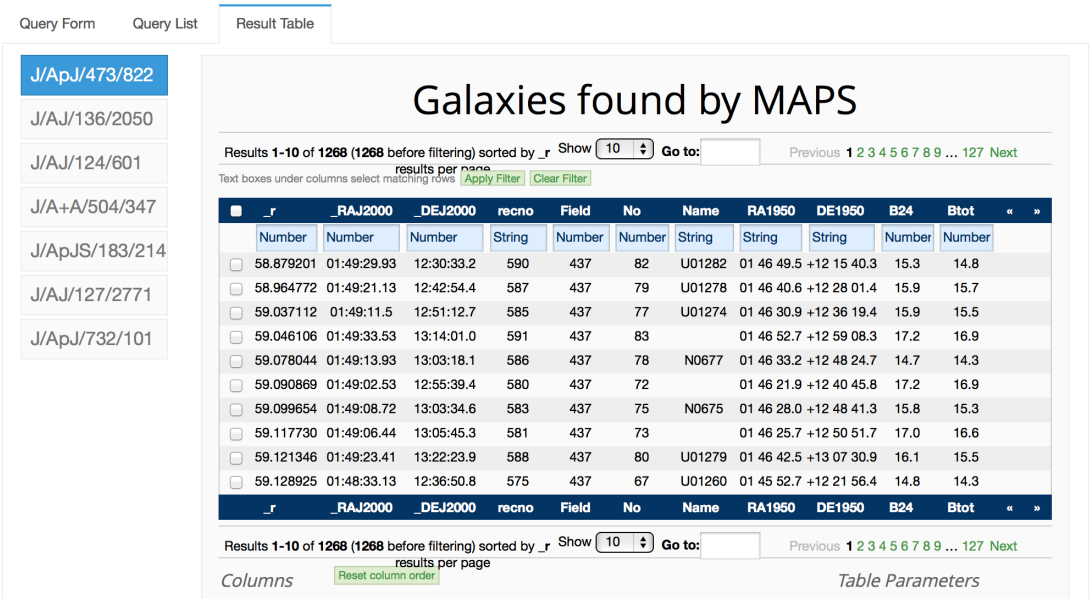

Figura 3: Múltiples Resultados