UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Integrando servicios mediante la automatización de procesos para la selección y adquisición de datos (TDAQ) en el CERN

Autores: Alejandro Santos
Director: Mg. Lía Hebe Molinari
Codirector: Dra. Giovanna Lehmann Miotto
Asesor profesional: ---
Carrera: Licenciatura en Informática

Resumen

Shifter Assistant (SA) es un software desarrollado en CERN, la Organización Europea para la Investigación Nuclear, el cual permite ejecutar procesando datos operacionales de ATLAS y disparando alertas en condiciones específicas de eventos. Acompaña a los operadores en sus tareas diarias en la sala de control de ATLAS, permitiendo diagnosticar situaciones problemáticas y asistir en la solución de problemas, con algunas especiales indicaciones on qué pasos a seguir.

Funciona mediante directivas que ofrecen validaciones, evaluando condiciones en tiempo real, procesando, analizando, y correlacionando fuentes de información heterogénea. Una vez que una condición se cumple, propaga el mensaje apropiado hacia el shifter en la sala de control, o, por ejemplo, enviando un SMS o un correo electrónico hacia el experto de guardia.

El proyecto propuesto consiste en diseñar e implementar un flujo de trabajo que permita a diferentes usuarios insertar o modificar sus propias reglas de SA.

Palabras Claves

ESPER. CEP. CERN. Testing. Pruebas. Workflow. Documentación.

Conclusiones

En este trabajo se ha presentado un completo mecanismo de pruebas para el Shifter Assistant de ATLAS que permite a los usuarios de diferentes dominios insertar sus conocimientos en la aplicación y verificar la correctitud de sus directivas. La reproducibilidad de las pruebas permite colaborar entre colegas en la etapa de desarrollo de software y también permite introducir una procedimiento formal de aprobación de código listo para ser puesto en producción.

Trabajos Realizados

Una propuesta de workflow que los expertos puedan seguir a fin de poder escribir y validar sus propias directivas. Documentación de las clases, objetos, tipos de datos y eventos propios de Shifter Assistant (SA) disponibles desde las directivas. Descripción de la configuración de SA. La aplicación OSIRIS con una interfaz estándar que permita leer y reconstruir eventos históricos almacenados en medios de almacenamiento secundarios y bases de datos. Modificaciones a SA a fin de aceptar la inyección de eventos pasados.

Trabajos Futuros

El trabajo promovió el desarrollo de nuevos proyectos: la creación de una interfaz web para facilitar su uso por parte de los usuarios, y la posibilidad de utilizar la API de OSIRIS desde Python y Java.

Fecha de la presentación: Julio 2015

Integrando servicios mediante la automatización de procesos para la selección y adquisición de datos (TDAQ) en el CERN

Alejandro Santos

Dirección: Mg. Lía Hebe Molinari
Co-dirección: Giovanna Lehmann Miotto, PhD.

Julio 2015

Índice general

Agradecimientos

Agradezco a mi mamá, papá y hermana. A mi familia y amigos por haberme ayudado y alentado a realizar este viaje de 18 meses a Ginebra, Suiza. Porque a pesar de la distancia los tuve más cerca que nunca.

Agradezco a mis amigos, compañeros y colegas de la UNLP. Al Profesor Javier Díaz y a la Profesora María Teresa Dova por su confianza y permitirme realizar esta hermosa experiencia. A la Profesora Lía Molinari por su criterio e invaluable esfuerzo en facilitar el desarrollo del presente trabajo de Licenciado en Informática. A las Profesoras Alejandra Schiavoni y Catalina Mostaccio por compartir su experiencia docente y fomentar mi desarrollo académico.

I thank my CERN advisors, colleagues, and friends. To my CERN supervisor Giovanna Lehmann Miotto, for her support and encouragement to become a better software professional. To Andrei, Gabriel, Giuseppe, and Igor for their continued patience to follow and advice my work.

Propuesta de tesina

El experimento ATLAS en el CERN adquiere datos de Física mediante un complejo sistema de selección y adquisición de datos, el cual consiste de procesos de software corriendo en 20000 núcleos interconectados mediante redes Ethernet de altas prestaciones.

Mientras que en los primeros años de operaciones los procesos de adquisición de datos fueron controlados principalmente por una serie de expertos, hoy en día cualquier miembro de la colaboración ATLAS (más de 3000 científicos autores) pueden registrarse como operador, luego de un rápido entrenamiento y unas horas de "sombra" de otro operador.

Por lo tanto, la sala de control se encuentra poblada por 10 "shifters" a cargo de diferentes aspectos del experimento (seguridad del detector, interacción con el acelerador, control de calidad de datos online, selección online de eventos) y quienes tienen muy poca información detallada de los componentes que operan.

A fin de mantener una alta eficiencia en la adquisición de datos del experimento ATLAS, el equipo TDAQ impulsó el desarrollo en dos direcciones: por un lado se automatizaron tantos procedimientos y recuperaciones como fue posible, y por el otro lado se desarrolló una aplicación llamada Shifter Assistant (SA), la cual alerta a los operadores en caso de anomalías y sugiere acciones a tomar.

El Shifter Assistant (SA) encontró un gran éxito y se lo ha expandido desde su alcance original del área de adquisición y selección de datos, también para incluir estado del detector y calidad de datos. Está basado en un motor de Procesamiento de Eventos Complejos (CEP[1]) y el conocimiento se encuentra codificado mediante el uso del Lenguaje de Procesamiento de Eventos (EPL) modelado en base a SQL.

Con la expansión de la base de conocimientos hubo que enfrentarse con el problema general afectado por los sistemas experto: la administración de la base de conocimientos, en términos de expansibilidad y pruebas. Dado que SA fue desarrollado durante la fase de toma de datos (run 2010-2012), se decidió adoptar un enfoque centralizado, con las directivas siendo desarrolladas y probadas por dos desarrolladores solamente, a pedido de diferentes expertos.

El proyecto propuesto consiste en diseñar e implementar un flujo de trabajo

[1] `http://esper.codehaus.org/`

que permita a diferentes expertos insertar o modificar directivas EPL. Este flujo de trabajo debe permitir:

1. Que las directivas sean modificadas por las personas con el acceso debido.
2. Verificación de la validez sintáctica de las directivas antes de su envío.
3. Realizar pruebas de unidad automáticas en las directivas modificadas.
4. Solicitar la validación explicita y aceptación, por parte de un responsable, de la directiva antes de que sea puesta en producción.
5. Registrar en la bitácora electrónica cualquier cambio en las directivas que van en producción.

Capítulo 1

Introducción

Shifter Assistant (SA) es un software desarrollado en CERN, la Organización Europea para la Investigación Nuclear, el cual permite ejecutar *directivas* procesando datos operacionales de ATLAS y disparando alertas en condiciones específicas de eventos. Por ejemplo, disparar una alerta cuando un valor en un evento se encuentra fuera de un rango específico.

ATLAS es uno de los seis detectores primarios del LHC, el Gran Acelerador de Hadrones, el cual es el mayor acelerador de partículas del mundo. ATLAS es el más grande de los detectores construidos por el hombre hasta la fecha. El LHC es un anillo de 27 Km de circunferencia que se encuentra bajo tierra en la frontera entre Suiza y Francia en la localidad de Ginebra, Suiza.

El LHC no se encuentra en funcionamiento permanente. Su construcción comenzó en los años '90 y recién en el año 2009 comenzó su funcionamiento a pleno, produciendo colisiones de partículas y recolectando datos de física para su posterior análisis. En el año 2013 la ejecución del LHC se detuvo a fin de poder actualizar sus componentes y poder incrementar la energía entregada en las colisiones de partículas. El período de tiempo entre 2009 y 2013 donde el LHC estuvo en funcionamiento se lo conoce como *Run 1*. El próximo período comprendido entre 2015 y 2018 se lo conoce como *Run 2*.

SA es una herramienta que acompaña a los *shifters* en sus tareas diarias en la sala de control de ATLAS, permitiendo diagnosticar situaciones problemáticas y asistir en la solución de problemas, con algunas especiales indicaciones on qué pasos a seguir. Funciona ofreciendo validaciones y evaluando condiciones en tiempo real, procesando, analizando, y correlacionando fuentes de información heterogénea. Una vez que una condición se cumple, propaga el mensaje apropiado hacia el shifter en la sala de control, o, por ejemplo, enviando un SMS o un correo electrónico hacia el experto de guardia.

SA encontró un gran éxito y extendió su alcance y funciones desde el área de selección y adquisición de datos hacia el funcionamiento del detector y calidad de datos. Está basado en un motor de Procesamiento de Eventos Complejos (*Complex Event Processing*, CEP, en inglés) llamado Esper [12] y su base de

conocimientos (en la forma de directivas) es codificada mediante el uso de un Lenguaje de Procesamiento de Eventos (*Event Processing Language*, EPL, en inglés) modelado en base a SQL.

Shifter Assistant reacciona ante dos flujos de eventos y una base de datos de ATLAS, conocidos como *ERS* Capítulo 3.1, *IS* Capítulo 3.2, y *OKS* Capítulo 3.6. *ERS* es un sistema de mensajería de propósito general para el registro de mensajes de error e información, siendo éstos almacenados en servidor de base de datos SQL. *IS* es un sistema de comunicación inter-proceso utilizado por el software de ATLAS para la comunicación y control del detector y sus componentes. La intención de ERS es la de registrar pequeños mensajes de texto en bajas cantidades, mientras que la de IS es la de proveer un mencanismo de propósito general para compartir información entre aplicaciones. Ambos sistemas de mensajería y comunicación fueros desarrollados íntegramente en CERN. *OKS* es la base de datos de configuración de ATLAS, donde se detallan los componentes activos junto a sus parámetros de configuración del detector en su funcionamiento.

En el pasado y durante Run 1 del LHC, las directivas de Shifter Assistant fueron escritas por su equipo de desarrollo. Los expertos proponían un caso de uso en inglés, y las directivas propiamente en EPL eran escritas por el equipo de desarrolladores de Shifter Assistant. Este es aún el caso hoy en día, y los expertos que quieran delegar este trabajo al equipo de Shifter Assistant todavía pueden hacerlo.

Las pruebas de software son una etapa integral en el proceso de desarrollo de software y el único mecanismo que puede permitir una buena calidad en términos de funcionalidad y eficiencia. Un sistema complejo no puede ser puesto en funcionamiento sin tener extensivas pruebas de integración.

Como parte del presente trabajo fue desarrollado un servicio y un conjunto de herramientas que permita a los expertos escribir y validar sus propias directivas sin depender directamente de la disponibilidad humana del equipo de desarrollo de Shifter Assistant.

Un servicio fue desarrollado que permite probar las directivas de SA ejecutándolas en el sistema dentro de un *arenero* (sandbox, en inglés) mediante el uso de una réplica configurable de datos, eventos y configuraciónes pasadas y almacenadas en medios de almacenamiento secundarios. Los eventos pasados son reconstruidos, simulando la ejecución como si fuese lo más cercana a la realidad posible. Este enfoque permite la validación de directivas en un ambiente controlado e independiente, permitiendo mantener un historial de pruebas pasadas. El pocedimiento para las pruebas de directivas de SA se llama *Shifter Assistant Replay Testing Model*, o *SAReplay*.

Fue desarrollado durante el período de 17 meses, desde Febrero 2014 hasta Junio 2015, en el marco de una beca de estudios obtenida a través de la gestión del IFLP-UNLP[1] en en experimento ATLAS del CERN. Investigadores y estu-

[1]Instituto de Física de La Plata de la Universidad Nacional de La Plata

diantes del IFLP-UNLP poseen presencia permanente en ATLAS del CERN, y ante la posibilidad de incorporar un estudiante de Informática al experimento es que se produjo la oportunidad de desarrollar el trabajo.

Específicamente, el presente trabajo consiste de:

1. Un plan de acción (propuesta de workflow) que los expertos puedan seguir a fin de poder escribir y validar sus propias directivas. Capítulo 1.1
2. Documentación de las clases, objetos, tipos de datos y eventos propios de Shifter Assistant disponibles desde las directivas. Al momento de comenzar este trabajo no existía documentación de desarrollo de Shifter Assistant. Capítulo 4.6
3. Descripción de la configuración de Shifter Assistant. Capítulo 4
4. El desarrollo de la aplicación OSIRIS con una interfaz estándar que permita leer y reconstruir eventos históricos de IS y ERS almacenados en medios de almacenamiento secundarios y bases de datos. Capítulo 5
5. Modificaciones propias a Shifter Assistant a fin de aceptar la inyección de eventos históricos pasados. Capítulo 6
6. Un paper en inglés presentado en la conferencia CHEP 2015 [2] con una descripción del servicio [6].

Los desafíos de este trabajo consisten en la construcción de una herramienta que permita eficientemente la lectura de valores históricos y la reconstrucción de los eventos pasados, en una forma que Shifter Assistant acepte como válidos y permita reproducir eventos pasados lo más cercanos a la realidad posible. Las dos herramientas desarrolladas son:

1. OSIRIS, que permite leer valores históricos de IS con P-BEAST.
2. Un nuevo inyector de valores hacia Shifter Assistant, tomando los valores de OSIRIS y a lo largo de tiempo la construcción de los objetos de IS.

[2] International Conference on Computing in High Energy and Nuclear Physics (CHEP)

1.1. Propuesta de Workflow

La propuesta de workflow a fin que los usuarios puedan escribir y validar sus directivas a fin de poder incluirlas en el sistema de producción de Shiter Assistant es:

1. Estudiar y analizar el problema que se desea observar.
2. Escribir la directiva EPL que dispare la alerta deseada Capítulo 4.
3. Encontrar un rango de tiempo donde se encuentren datos almacenados interesantes Capítulo 3.1.1.
4. Ejecutar SA en el modo *replay* Capítulo 6.1.
5. Verificar la salida del modo *replay* Capítulo 6.1.
6. Si el resultado es aceptable, enviar la directiva para ser aprobada.
7. Caso contrario, volver a comenzar.

Figura 1.1: Propuesta de Workflow. Fuente: Presentación de SAReplay en CHEP 2015 [7]

Capítulo 2

CERN

2.1. CERN

El CERN es la Organización Europea para la Investigación Nuclear, y es el mayor laboratorio y centro de investigación de física de partículas en el mundo [1].

El principal experimento del CERN es el Gran Acelerador de Hadrones (LHC) que consiste de un acelerador de partículas de 27km de diámetro, ubicado bajo tierra a 150m de la superficie. Como parte del LHC existen seis grandes experimentos: ATLAS, CMS, ALICE, LHCb, LHCf y TOTEM. Estos seis experimentos poseen, cada uno, un detector ubicado en uno de los ocho puntos de acceso del LHC. Las partículas aceleradas colisionan en cada uno de los detectores.

El LHC es la etapa final de aceleración del haz de partículas. El proceso comienza obteniendo una muestra de hidrógeno. Los protones se consiguen removiendo los electrones de cada átomo de hidrógeno, y se inyecta el haz hacia el PSB (Particle Synchroton Booster) desde Linac2. Desde el PSB, los protones se inyectan hacia el PS, luego al SPS, para terminar en el LHC.

En ambos casos, las partículas se aceleran y agrupan en racimos o "*bunches*", a fin de maximizar la interacción al momento de la colisión. En el LHC existen dos caños por los que las partículas circulan en dirección opuesta, y en los puntos de colisión los dos haces que viajan cada uno por uno de los conductos se encuentran. A fin de maximizar la eficiencia de colisión de partículas, estos caños son tuberías al vacío. De otra manera, las partículas aceleradas colisionarán con el gas que exista dentro de estas tuberías.

Poderosos electroimanes superconductores generando un campo magnético se usan para orientar y dirigir el haz de partículas, mientras que cavidades resonantes se utilizan para aumentar su velocidad y energía. La combinación de los electroimanes superconductores, cavidades resonantes, caños conductores de los haces de partículas y los detectores conforman el LHC, el Gran Acelerador de Hadrones.

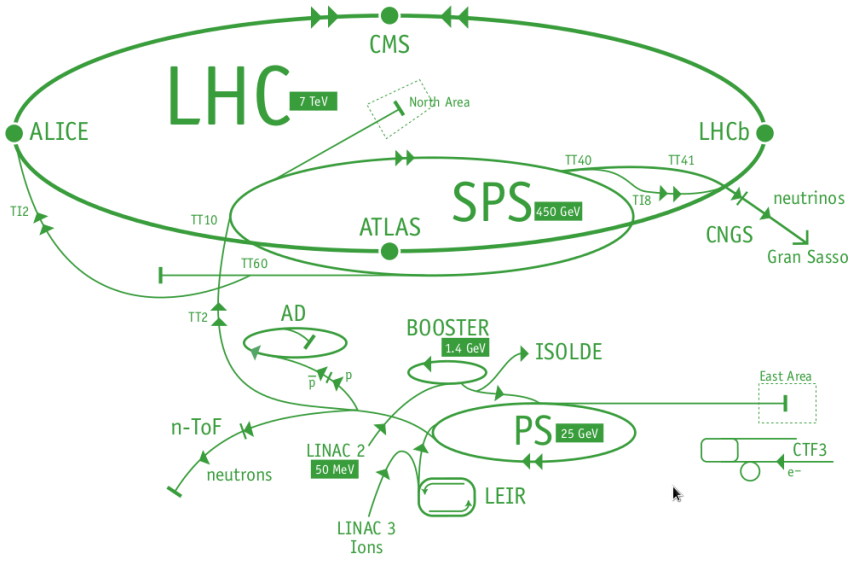

Figura 2.1: Serie de aceleradores de CERN. Fuente: *LHC, the guide* [1]

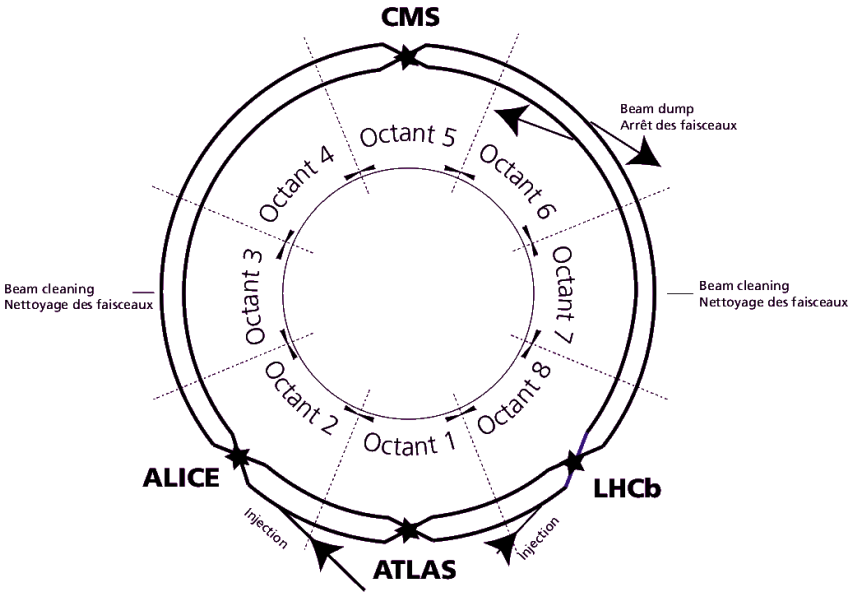

Figura 2.2: Gran Colisionador de Hadrones (LHC). Fuente: *LHC, the guide* [1]

2.2. ATLAS

ATLAS es uno de los experimentos del LHC. Es un detector de propósito general diseñado para cubrir el mayor espectro de física en el LHC, y el descubrimiento más popular por lo que se conoce a ATLAS (y CMS) es el Bosón de Higgs, la partícula fundamental responsable de darle masa a la materia. ATLAS posee ocho electroimanes superconductores de 25 metros que le dan la forma de rosquilla, orientados de forma de rodear y cubrir el caño del haz de partículas en el centro del detector. Es necesario que los imanes sean superconductores para obtener la mayor eficiencia de generación de campo magnético, que de otra forma no sería práctico ni posible obtener campos magnéticos de suficiente intensidad.

ATLAS es el mayor y más grande detector-colisionador construido. La colaboración consiste de más de 3000 miembros de 177 universidades en 38 países [2]. ATLAS posee varios sub detectores encargados de registrar eventos específicos, de forma de que los sub detectores funcionan en conjunto a fin de observar los eventos interesantes para el análisis de física. Los sub detectores son tres: *Pixel*, *Calorimeter*, y *Muon*.

2.2.1. Point 1

La red de comunicación de ATLAS es una red Ethernet Gigabit clásica, y se encuentra aislada a la red del CERN. La red de control privadas de ATLAS se llama *ATCN* [9], y por su obicación en el anillo del LHC también se la conoce como *Point 1*. La red de *Point 1* es de acceso limitado solo a los miembros de la colaboración ATLAS, los permisos de acceso deben ser solicitados y otorgados por los administradores de la red, y el acceso se realiza físicamente en la sala de control de ATLAS, o mediante una puerta de enlace de acceso limitado solo a los miembros de ATLAS.

Dentro de *Point 1* se encuentran los servidores del Trigger y Adquisición de Datos de ATLAS, y de las aplicaciones de infraestructura y control, incluyendo donde se ejecutan Shifter Assistant Capítulo 3.3 y P-BEAST Capítulo 3.4.

2.3. TDAQ

El detector ATLAS produce dos grandes conjuntos de datos. Por un lado se encuentran los datos de física, y por otro los datos operacionales, de control y monitorización. Los datos de física se toman desde cada uno de los sub detectores hacia el *High Level Trigger* (HLT) por medio de una red dedicada de 2000 hilos de fibra óptica de 10 GbE, conectando los dispositivos electrónicos de cada sub detector ROD (*Read Out Drivers*) hacia cada uno de los ROS (*Read Out System*). [4] [5]

Un ROS es una computadora de propósito general con una interfaz de red colectando los datos de física, y todos los ROSes en conjunto forman la interfaz

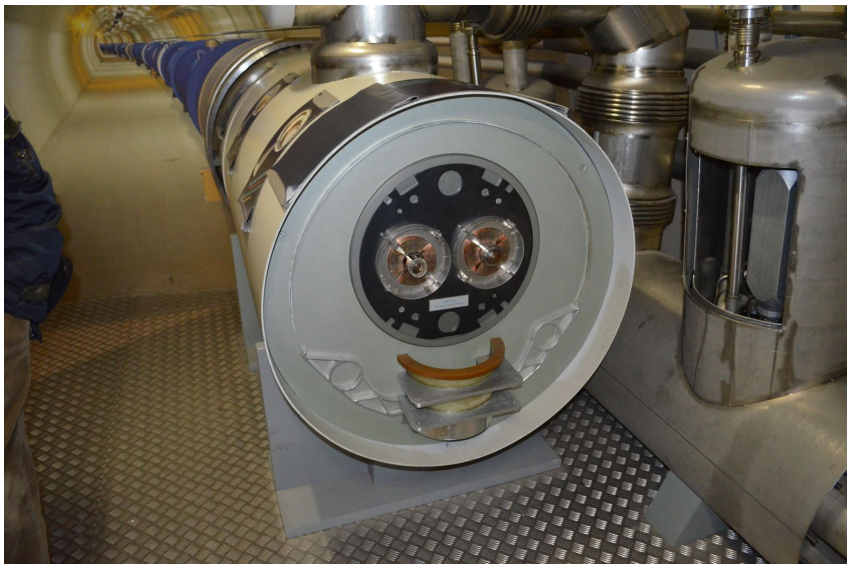

Figura 2.3: Corte de muestra de la tubería del LHC

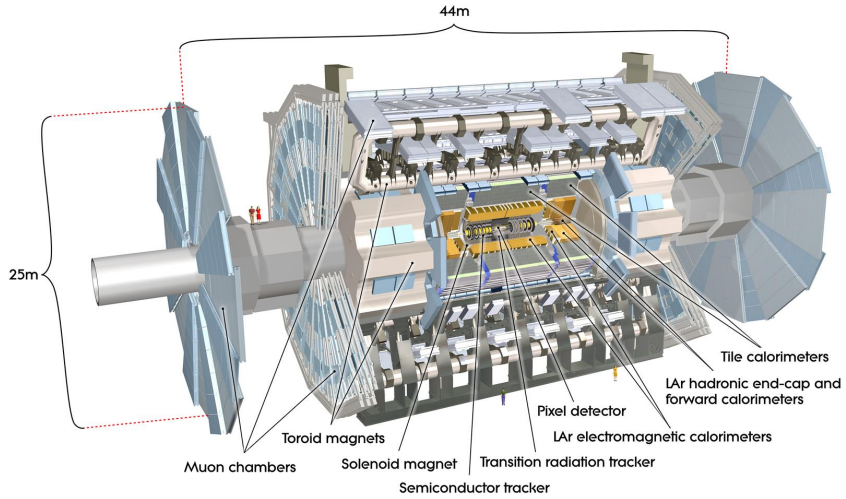

Figura 2.4: ATLAS. Fuente: ATLAS [2] *ATLAS Experiment © 2014 CERN*

Figura 2.5: Sala de control de ATLAS

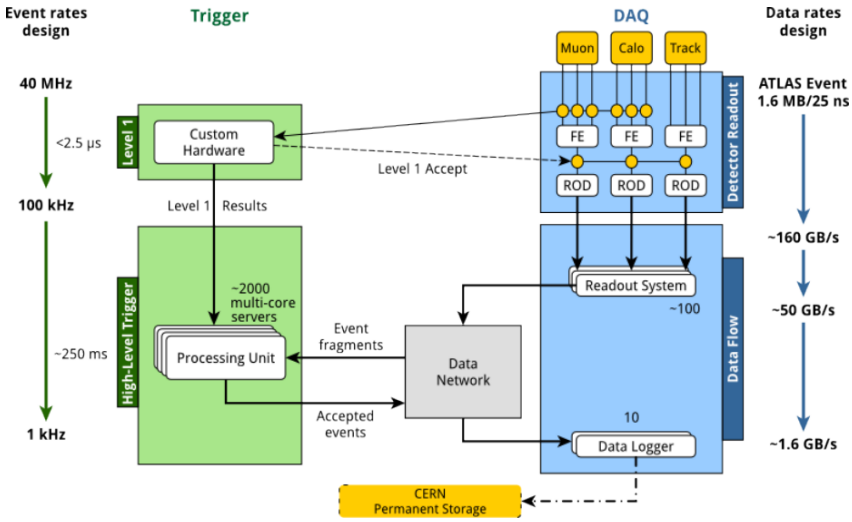

Figura 2.6: ATLAS TDAQ, Trigger y Adquisición de Datos. Fuente: *Evolution of the ReadOut System of the ATLAS experiment* [5]

de lectura de datos del HLT. El HLT es el encargado de seleccionar y filtrar los eventos entrantes. Existe una etapa previa intermedia entre el el ROD y HLT conocida como Trigger de Nivel 1, encargada de hacer un filtro y selección inicial y previa al HLT.

Los eventos obtenidos en cada sub detector se disparan con una frecuencia de 40 MHz. Esto significa que la colisión de los racimos de partículas se realiza 40 millones de veces por segundo. El Trigger de Nivel 1 reduce los eventos producidos de las colisiones a una frecuencia de 100 KHz, para luego el HLT terminar de seleccionar eventos interesantes a una frecuencia de 1 KHz. El resultado final se envía a medios de almacenamiento secundarios como lo son discos rígidos y cintas magnéticas.

El análisis de datos de física se realiza en forma *offline*. Los investigadores toman los datos almacenados en los medios de almacenamiento secundarios y realizan el análisis y estudios deseados, permitiendo repetir y variar las condiciones de análisis independiente del funcionamiento y estado actual del detector.

Por otro lado, el segundo conjunto de datos de ATLAS son los datos operacionales, de control, configuración y monitorización. Estos datos incluyen información de control y estado de los sub detectores y diferentes componentes. La red de conexión por la que viajan estos eventos es independiente a la de los datos de física, *ATCN*, y el protocolo de comunicación utilizado es CORBA a bajo nivel, aunque al día de hoy existen algunas abstracciones que permiten facilitar su uso. Entre ellas existen dos sistemas de mensajería y comunicación, ERS Capítulo 3.1 e IS Capítulo 3.2.

Capítulo 3

Recursos preexistentes al trabajo

3.1. ERS

El Servicio de Reporte de Errores (ERS) es un mecanismo general para el reporte de errores de las aplicaciones ATLAS de TDAQ. Es un conjunto de librerías de software en C++ y Java, donde los mensajes de error se almacenan de forma remota en una base de datos SQL. Esto permite consultar de forma rápida y sencilla los errores reportados por las aplicaciones. [22] [23] [24]

Estos mensajes son el resultado de haber ejecutado las aplicaciones de TDAQ de ATLAS, donde el tipo de mensajes es diverso. Los mensajes se encuentran organizados en una jerarquía, donde primero se toma el número de versión de *Release* de TDAQ, el cual indica la versión del software utilizada para realizar la toma de datos de ATLAS, luego el nombre de la partición Capítulo 3.2.1 donde se ejecutaron las aplicaciones de la toma de datos, luego el nombre de usuario utilizado para realizar la toma de datos, y por último el *Número de Run* Capítulo 3.1.1 asociado.

Los campos de un mensaje de tipo ERS son:

Message	Indica el texto del mensaje propiamente.
Application	Indica el nombre de la aplicación que disparó el mensaje.
Message ID	Indica el tipo de mensaje.
Host	Indica el nombre de la máquina donde se disparó el mensaje.
Param	Parámetros propios para cada tipo de mensaje.
Qualifiers	Conjunto de identificadores asociados a un mensaje, utilizados como etiquetas para proveer información adicional y facilitar su filtrado posterior.

Type Tipo de mensaje, el cual puede ser alguno de: *Fatal, Error, Warning, Information*, y se utilizan para indicar una prioridad de mensaje. Un mensaje de tipo *Fatal* es aquel que debe recibir la inmediata atención del responsable de la aplicación que generó el mensaje. Un mensaje de tipo *Error* es un error asociado a la aplicación. Uno de tipo *Warning* es una advertencia que dependiendo el caso puede indicar un problema alternativo, y uno de tipo *Information* es un mensaje informativo para mantener alguna clase de registro de un evento producido.

Timestamp Fecha y hora del momento en el que el mensaje fue producido.

3.1.1. Números de Run

Un *Número de Run* en ATLAS identifica unívocamente mediante un número, un intervalo de tiempo en donde el HLT se encuentra funcionando. Esto puede ser porque el LHC se encuentra en funcionamiento y las colisiones de partículas están ocurriendo, o porque se está realizando alguna ejecución de pruebas y se están produciendo datos falsos o de pruebas en el HLT. Un número de Run es único en ATLAS.

Por ejemplo, el *Número de Run* 251876 comenzó a regir en la partición Capítulo 3.2.1 `ATLAS` con el nombre de usuario `crrc` desde el 12 de Febrero 2015 a las 21:28:19, y terminó al momento de comenzar el siguiente, con el Número de Run 251880, teniendo el Run 251876 una duración de aproximadamente 22 horas y 30 minutos. Esta información se puede consultar mediante el uso de la aplicación *Log Manager* Capítulo 3.1.2.

3.1.2. Log Manager

La aplicación `log_manager` permite, mediante una interfaz gráfica, visualizar los mensajes almacenados en la base de datos de ERS. Capítulo 3.1 Parte de su funcionalidad es la de filtrar mensajes por cada uno de los campos de los mensajes asociados al *Número de Run* elegido.

3.2. IS

El Servicio de Información (IS) es un mecanismo para compartir la información de las aplicaciones de ATLAS ejecutándose en un ambiente distribuido [19]. IS mantiene un estado global de sus objetos y atributos. Cada vez que una aplicación actualiza un atributo de un objeto, este valor permanece en el sistema hasta que es explícitamente eliminado.

IS se encuentra implementado internamente a partir de la librería `IPC`, la cual a su vez se encuentra implementada en términos de `CORBA` [20] [21]. `CORBA` es un protocolo de comunicación estándar que permite a diferentes aplicaciones en diferentes plataformas y lenguajes de programación puedan comunicarse. En

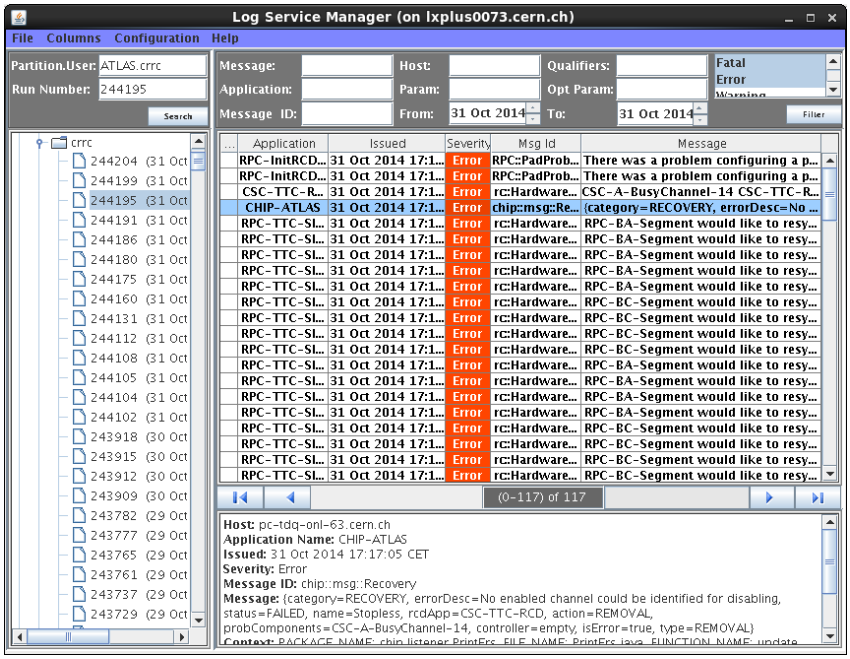

Figura 3.1: ERS Log Manager

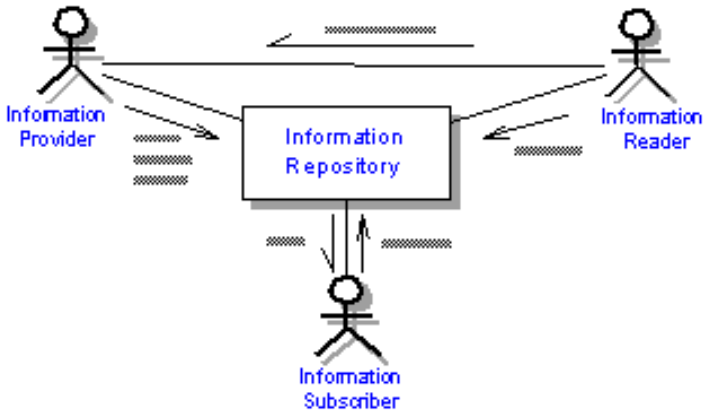

Figura 3.2: Information Service (IS). Fuente: IS User Manual [19]

ATLAS las aplicaciones CORBA son en su mayoría escritas en Java y C++, pero en lugar de utilizar CORBA directamente se utiliza IS y, en menor medida, IPC. Tanto la librería IPC como IS fueron desarrolladas en CERN.

3.2.1. Particiones, clases y servidores

IS se divide en *particiones*. Cada *partición* es un espacio de nombres global que posee varios miles de objetos. Cada *objeto* posee una *clase* que define la lista de atributos posibles para este objeto. Los *atributos* tienen un tipo de datos concretamente definido, y el valor de los atributos de cada objeto puede variar en el tiempo.

Los objetos se encuentran almacenados en *servidores*, y todo objeto debe tener un *servidor* asociado. Un *servidor* aporta un segundo nivel de indirección dentro del espacio de nombres global de la partición donde se encuentra el objeto, y la identificación unívoca de un objeto está dada por la tupla: *(partición, servidor, objeto)*, donde el nombre del *servidor* es parte del nombre del objeto.

Por ejemplo en la partición "ATLAS" se puede encontrar el objeto con el nombre "RunParams.RunInfo", donde el prefijo "RunParams" marcado por el punto indica el nombre del servidor, y el nombre del objeto propiamente es "RunInfo".

3.2.2. Publicación de valores en IS

Es posible crear objetos y modificar sus atributos en IS. Para leer los valores de los atributos hay dos formas, la primera es consultar el valor explícitamente, la segunda es, desde un programa C++ o Java, registrar un *listener* y recibir notificaciones al momento en que algún objeto u atributo cambia su valor.

La publicación de valores en IS permite que este dato sea accesible desde otra aplicación, siendo ejecutada potencialmente en otra máquina física. Es necesario publicar este valor a fin de que esto sea posible.

Un ejemplo en C++ para la publicación de valores simples se puede encontrar en el manual de usuario de IS, el cual se transcribe aquí:

```
int main( int ac, char ** av ) {
  // Inicializa la librería de comunicación
  IPCCore::init( ac, av );
  // Crear la instancia de la partición
  IPCPartition partition("part_hlt_asantos");
  // Crear el diccionario IS de la partición
  ISInfoDictionary dict(partition);

  // Crear las instancias de la información a publicar
  // e inicializarles un valor
  ISInfoInt voltage(220);
  ISInfoFloat temperature(22.3);
```

```
  // Publicar la información en el servidor IS DQM
  dict.checkin("DQM.DeviceVoltage", voltage);
  dict.checkin("DQM.DeviceTemperature", temperature);

  return 0;
}
```

En el anterior programa se inicializa la librería de comunicación IPC con el entorno invocado de la aplicación, se obtiene una referencia a la partición con nombre "`part_hlt_asantos`", y se obtiene una referencia al diccionario de nombres de la partición. Luego se inicializan dos objetos de tipo `ISInfoInt` e `ISInfoFloat` que permiten almacenar valores simples de tipo de datos `int` y `float`, y por último mediante el llamado al método "`checkin`" se publican estos dos objetos bajo los nombres "`DeviceVoltage`" y "`DeviceTemperature`" dentro del *servidor* "`DQM`".

3.2.3. Lectura de valores de IS

Hay dos maneras de leer valores de IS. La primera es consultarlos explícitamente. Esto se puede lograr mediante el uso del método `getValue` de un objeto `ISInfoDictionary`. Por ejemplo, siguiendo con el mismo programa de ejemplo anterior, en lugar de escribir un valor podemos leerlo desde C++ de la siguiente manera:

```
ISInfoDictionary dict(partition);
dict.getValue("DQM.DeviceVoltage", voltage);
dict.getValue("DQM.DeviceTemperature", temperature);
```

Asimismo también es posible leerlo en Java de la siguiente manera:

```
public static String readObject(String partition,
   String objName) {
  StringBuilder sb = new StringBuilder();
  Partition p = new Partition(partition);
  Repository r = new Repository(p);
  AnyInfo info = new AnyInfo();

  r.getValue(objName, info);
  InfoDocument doc = new InfoDocument(p, info);

  for(int i = 0; i < info.getAttributeCount(); ++i) {
    sb.append(doc.getAttribute(i).getName() + "=" +
       info.getAttribute(i));
    sb.append(", ");
  }
```

```
  return sb.toString();
}
```

Al momento de publicar la variable `voltage` de tipo `ISInfoInt` se publica un objeto de tipo "S32" con un único atributo, de nombre `value` y de tipo S32. El tipo de datos S32 representa un número entero con signo de 32 bits.

La segunda manera de leer valores de IS es mediante la subscripción de un listener, y esperar a recibir las notificaciones de cambios. Por ejemplo, para obtener una notificación cada vez que se actualiza un valor podemos tener:

```
public class Main implements InfoListener {
  public static final String PARTITION_NAME =
    "part_hlt_asantos";
  public static final String SERVER_NAME = "DQM";

  private Criteria criteria = null;
  private Partition part = null;
  private Repository isrepo = null;

  public Main() {}

  public void configure() {
    criteria = new Criteria(java.util.regex.Pattern.
       compile(".*")) ;
    part = new Partition(PARTITION_NAME);
    isrepo = new Repository(part);
    isrepo.subscribe(SERVER_NAME, criteria, this);
  }

  @Override
  public void infoCreated(InfoEvent arg0) { /* ... */
     }

  @Override
  public void infoUpdated(InfoEvent arg0) { /* ... */
     }

  @Override
  public void infoDeleted(InfoEvent arg0) { /* ... */
     }

  public static void main(String[] args) {
    Main m = new Main();
    m.configure();
```

```
    for (;;) {
      try {
        Thread.sleep(10);
      } catch (InterruptedException e) {
        break;
      }
    }

    System.exit(0);
  }
}
```

El anterior programa subscribe a las notificaciones de creación, actualización y eliminación de objetos en IS. Cada una de estas tres situaciones se procesa mediante la invocación a los métodos infoCreated, infoUpdated, e infoDeleted del objeto Main del programa.

Primero hace falta registrar el objeto que implementa la interfaz InfoListener. Para ello hace falta crear un objeto Partition para obtener una referencia a la partición deseada, para luego con un objeto Repository registrar la instancia del listener a fin de recibir las notificaciones.

En este ejemplo se registraron notificaciones de todos los objetos de la partición "part_hlt_asantos" y del servidor "DQM", pero también es posible mediante el uso de una expresión regular filtrar los eventos y obtener notificaciones de solo algunos nombres de objetos. Esto se logra mediante la definición de una expresión regular al momento de crear la instancia del objeto Criteria.

3.2.4. Creando una nueva partición

Crear una nueva partición en lugar de utilizar las existentes permite hacer pruebas en un ambiente propio y controlado. Existen dos formas de lograrlo, la primera es inicializar manualmente el sistema de IS encargado de crear los canales de comunicación. Por ejemplo:

```
ipc_server -p part_hlt_asantos
is_server -p part_hlt_asantos -n MyServer
```

Los dos comandos anteriores se encargan, primero, de inicializar el mecanismo de comunicación IPC y crear una nueva partición part_hlt_asantos, para luego iniciar una nueva instancia de un servidor de IS llamado MyServer.

La segunda manera de crear una nueva partición es mediante el uso de un script destinado a crear la partición de IPC e iniciar la infraestructura de aplicaciones de TDAQ. Este método es el mismo que se utiliza a fin de iniciar los servicios en la partición ATLAS en el momento que el LHC y TDAQ se encuentran funcionando.

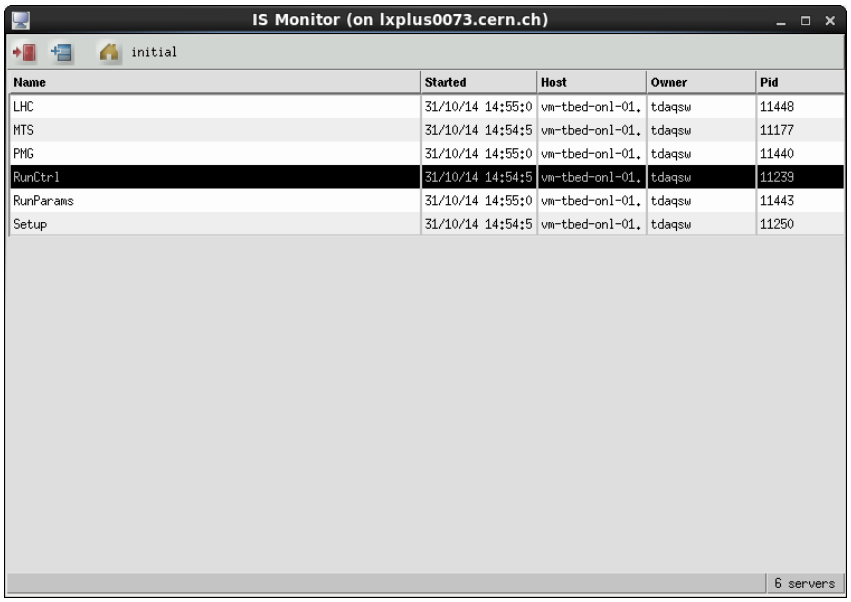

Figura 3.3: IS Monitor

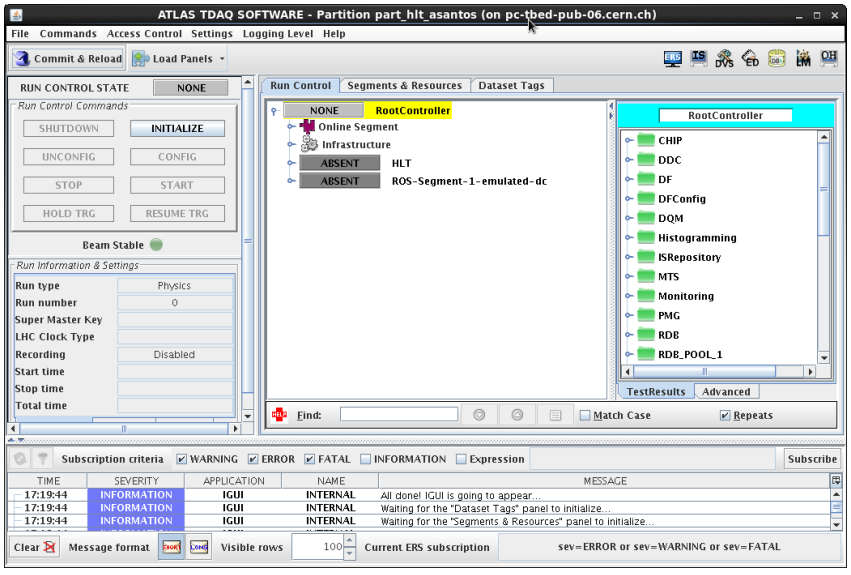

Figura 3.4: ATLAS TDAQ Software Partition

```
pm_part_hlt.py
setup_daq -p part_hlt_asantos -d part_hlt_asantos.data.xml
```

La herramienta `pm_part_hlt.py` se encarga de crear el archivo `part_hlt_asantos.data.xml` el cual contiene la configuración inicial de la partición. Este archivo es un archivo de configuración OKS Capítulo 3.6.

3.2.5. IS Monitor

Existe una herramienta conocida como `is_monitor` la cual permite consultar mediante una interfaz gráfica los objetos, sus atributos y valores asociados en IS. Eligiendo el nombre de la partición se puede ver la lista de servidores existentes, y a su vez dentro de cada uno se puede ver la lista de objetos. De cada objeto se puede ver la lista de sus atributos, los valores que toma cada atributo, y una breve descripción Capítulo 3.3.

3.3. Shifter Assistant

Shifter Assistant (SA) es una herramienta para la observación y análisis de eventos producidos por el detector ATLAS del CERN. Fue desarrollada en Java como parte de un trabajo de Doctorado en el CERN. Al proyecto también se lo conoce con los nombres de AAL, DAQAssistant, y Cassandra. [15] [16]

Mediante un conjunto de directivas que describen el comportamiento esperado de los eventos entrantes, es posible disparar alertas que les permiten a los operadores en la sala de control de ATLAS estar pendientes de eventos fuera de los parámetros normales producidos en las aplicaciones. Las directivas se escriben en un lenguaje de descripción eventos similar a SQL llamado EPL, y la herramienta específica de procesamiento de directivas EPL se conoce como Esper [12]. Esper es una librería Java.

El lenguaje EPL de Esper permite describir condiciones específicas de eventos, relacionarlos y filtrarlos. A diferencia de SQL que permite escribir consultas para procesar datos existenes en una serie de tablas y relacionarlos, EPL permite escribir consultas que reacciones a eventos entrantes. Por ejemplo, una simple directiva EPL puede ser:

```
SELECT *
FROM Evento
WHERE valor = 3
```

A diferencia de SQL donde se tomarían los valores ya existentes de una tabla con nombre `Evento`, en EPL se procesan eventos entrantes hacia Esper. Un evento es un objeto Java, una instancia de alguna de las clases registradas en Esper. Dado que Esper es una librería Java, esto se traduce en código a un objeto que implementa la interfaz `UpdateListener`, y cada vez que la condición

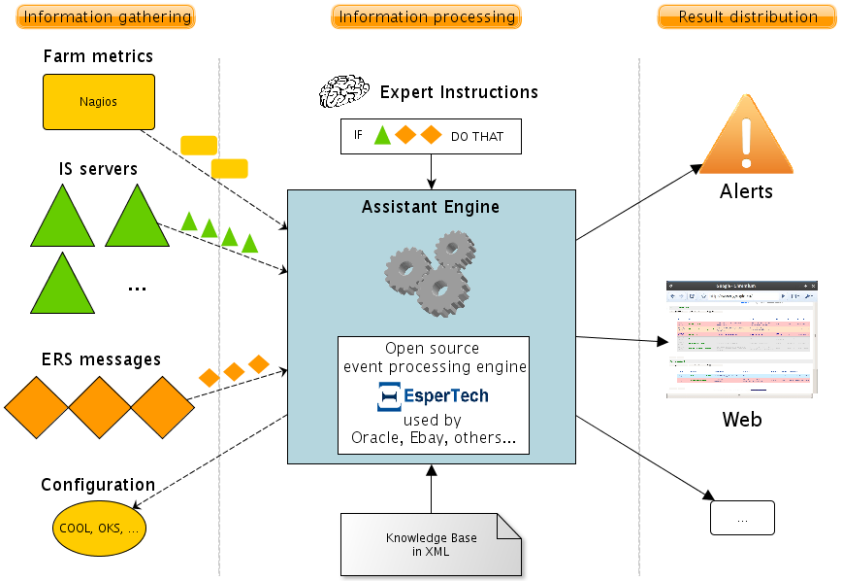

Figura 3.5: Shifter Assistant (SA). Fuente: *Intelligent monitoring and fault diagnosis for ATLAS TDAQ: a complex event processing solution* [15]

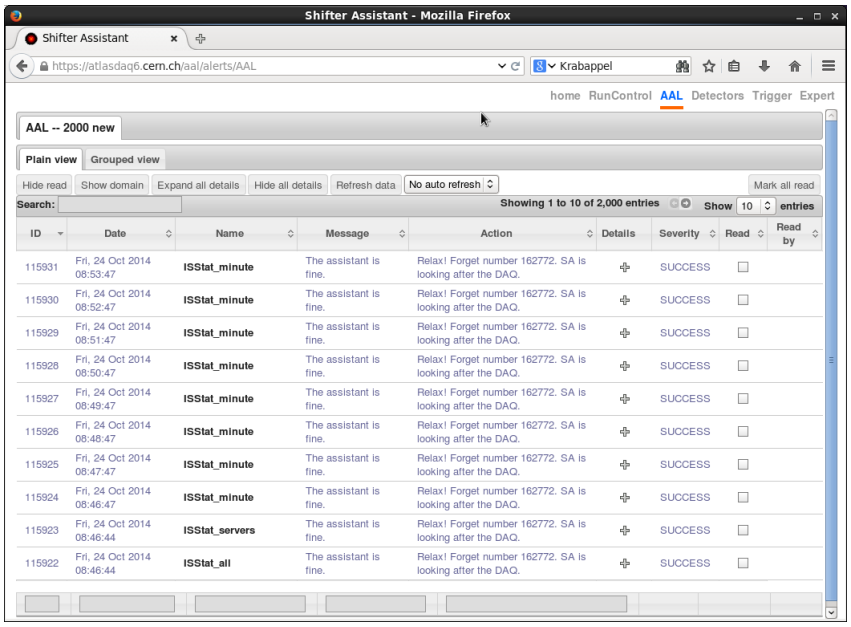

Figura 3.6: Shifter Assistant Web Appplication

de la sentencia EPL se evalúa como verdadera, el método `update` del objeto será invocado por Esper.

Por ejemplo, como se puede ver en la documentación de Esper, podemos crear una nueva clase Java que implemente la interfaz `UpdateListener`. A fin de recibir las notificaciones hace falta vincular la instancia de nuestra clase con la sentencia EPL propiamente: [13]:

```
public class MyListener implements UpdateListener {
    public void update(EventBean[] newEvents,
        EventBean[] oldEvents) {
         EventBean event = newEvents[0];
         System.out.println("avg=" + event.get("
            otroValor"));
    }
}

// ...
EPServiceProvider epService = EPServiceProviderManager
    .getDefaultProvider();
String expression = "SELECT * FROM Evento WHERE valor
    = 3";
EPStatement statement = epService.getEPAdministrator()
    .createEPL(expression);
MyListener listener = new MyListener();
statement.addListener(listener);
```

En Shifter Assistant, cuando la condición de la directiva se evalúa como verdadera se dispara una alerta. Las alertas son almacenadas en una serie de archivos de registro, y además son reenviadas a una base de datos a fin de mostrarlos en una aplicación web.

La aplicación web Capítulo 3.6 es la que los shifters de la sala de control se encuentran permanentemente observando a fin de que, al momento de producirse una alerta, puedan tomar una acción determinada.

3.4. P-BEAST

P-BEAST es un servicio para registrar en un medio de almacenamiento secundario los valores históricos de cada objeto de IS Capítulo 3.2. Entró en funcionamiento a comienzos del año 2014, y en el pasado no existía en general una forma de consultar un valor histórico de IS [17], [18].

Los datos de P-BEAST son almacenados como una secuencia de eventos; a un intervalo de eventos se le llama *time series*. Cada evento es un registro que contiene un valor, junto con el momento en el tiempo que el servidor de P-BEAST registró la recepción del evento.

Cada time series identifica los sucesos de cambios de un único atributo en un único objeto. Físicamente, cada time series se divide en archivos `.pb` que contienen aproximadamente siete días de eventos. Cada archivo que almacena cada time series puede ser identificado mediante la tupla de valores *(Partición, Clase, Atributo, Nombre, Rango de fechas)*, donde *Rango de fechas* suele ser siete días.

Sin embargo, la interfaz disponible para acceder a los datos de P-BEAST abstrae la separación en archivos `.pb`, por lo que desde un punto de vista de programación, de acceso a los datos y de la API, una *time series* puede identificarse por la tupla de valores *(Partición, Clase, Atributo, Nombre de Objeto)*.

```
/repository_root
  \-partition_name
    \-class_name
      \-attribute_name
        \-dotpb_file
          \-time_series for obj_1
            \-record
          \-time_series for obj_2
            \-record
```

partition_name Nombre de la partición, (i.e., "ATLAS")

class_name Nombre de la clase, (i.e., "RCStateInfo")

attribute_name Nombre del atributo, (i.e., "currentTransitionName")

dotpb_file Este es el archivo de datos, conteniendo aproximadamente una semana de datos. La convención para el nombre del archivo es `inicio -len.pb`, donde `inicio` es el *UNIX timestamp* del inicio de la ventana, y `len` es la longitud de la ventana de datos en segundos. Este archivo también puede tener una extensión `.pb.z` para archivos de datos comprimidos.

time series for obj_i Una secuencia de registros de un objeto dado.

record Una tupla con los valores (t_s, t_u, v), donde t_s es la fecha de creación o actualización del valor, t_u es la fecha de la última actualización del valor, and v es el valor concreto del evento.

Las herramientas existentes de P-BEAST no ofrecían la posibilidad de combinar diferentes *time series* en una única línea de tiempo, solo permitían leer de a un atributo a la vez. Es posible leer valores de un mismo atributo relacionados a varios objetos, aunque para hacer combinaciones complejas es necesario invocar a las aplicaciones de lectura de datos de P-BEAST repetivas veces. Internamente y desde un punto de vista de la API de P-BEAST Capítulo 3.4.1, una *time series* de `float` es leído en memoria como un `std::vector<float>`,

una *time series* de `std::string` es leído en memoria como un `std::vector <std::string>`, etc. Para combinar ambos vectores hace falta combinar ambos tipos de datos, una operación que no se encuentra definida de forma nativa en C++.

La primer forma de leer datos de P-BEAST es mediante el uso de las herramientas de línea de comandos de P-BEAST, la cual requiere un nombre de partición, de clase, y de un atributo a fin de poder leer datos. Mientras que estas aplicaciones permiten indicar un nombre de objeto o un conjunto de nombres de objeto mediante una expresión regular, la misma no combina diferentes *time series* de objetos en una única secuencia. Asimismo, estas aplicaciones no permiten leer datos desde más de una partición, clase o atributo. Esto significa que si se desea reconstruir un objeto con cada uno de sus atributos en un instante de tiempo dado, hace falta invocar a las herramientas de P-BEAST repetidas veces a fin de obtener para cada atributo su valor correspondiente.

Una limitación de P-BEAST es la política de almacenamiento de datos: no almacena eventos consecutivos con el mismo valor, sino que almacena el primero y escribe la fecha de actualización del último evento. Por ejemplo, si un atributo de un objeto obtiene la secuencia de actualizaciones $[1, 1, 1, 1, 2, 1]$, los datos registrados serán solamente $[1, 2, 1]$ perdiendo la información de los eventos consecutivos duplicados. La razón de esta elección es que la tasa de producción de datos en IS es muy alta, y en el período de un año y sin tener el LHC en funcionamiento ya se produjeros cerca de 1.8TB de datos de pruebas.

Los datos se almacenan físicamente en EOS Capítulo 3.5 fuera de *Point 1*, y en servidores dedicados para P-BEAST dentro de *Point 1*. La principal diferencia entre acceder a los datos es que, en el primer caso, las aplicaciones acceden a los archivos directamente mientras que en el segundo caso el acceso a los datos se realiza mediante CORBA hacia un servicio especializado de P-BEAST.

3.4.1. Acceso a los datos de P-BEAST

Para leer datos desde EOS es posible utilizar la herramienta de línea de comandos `pbeast_read_repository`, y un ejemplo de su uso es:

```
$ pbeast_read_repository -r $HOME/eos/atlas/
    atlascerngroupdisk/tdaq-opmon/pbeast -p ATLAS -c
    RunInfo -a activeTime -O '' -s "2014-06-25 15:35:00
    " -t "2014-06-25 15:37:00"
open shared memory file pbeast_writer_lock_memory
(INFO) - [2014-Jun-25 15:35:00.000000 - 2014-Jun-25
    15:37:00.000000] attribute type: u32, is array:
    false
RunParams.RunInfo:
    [2014-Jun-25 15:35:54.301367 - 2014-Jun-25
        15:35:54.301820] 0
    [2014-Jun-25 15:36:04.302812] 10
```

```
    [2014-Jun-25 15:36:14.303689] 20
    [2014-Jun-25 15:36:24.304799] 30
    [2014-Jun-25 15:36:34.305914] 40
    [2014-Jun-25 15:36:44.306733] 50
    [2014-Jun-25 15:36:54.307405] 60
```

Asimismo, para leer datos desde dentro de la red de *Point 1* es posible utilizar la aplicación `pbeast_read_server` que, desde la línea de comandos, permite leer los datos del servidor de P-BEAST. Un ejemplo de su uso es:

```
$ pbeast_read_server --server-name pbeast-server -p
    ATLAS -c RunInfo -a activeTime -O '' -s "2014-06-25
    ␣15:35:00" -t "2014-06-25␣15:37:00"
(INFO) - [2014-Jun-25 15:35:00.000000 - 2014-Jun-25
    15:37:00.000000] attribute type: u32, is array:
    false
RunParams.RunInfo:
    [2014-Jun-25 15:35:54.301367 - 2014-Jun-25
        15:35:54.301820] 0
    [2014-Jun-25 15:36:04.302812] 10
    [2014-Jun-25 15:36:14.303689] 20
    [2014-Jun-25 15:36:24.304799] 30
    [2014-Jun-25 15:36:34.305914] 40
    [2014-Jun-25 15:36:44.306733] 50
    [2014-Jun-25 15:36:54.307405] 60
```

Cada una de las aplicaciones utiliza internamente diferentes APIs para acceder a los datos almacenados. La primer API en C++ es la encargada de abstraer la lectura de archivos `.pb` desde EOS, mientras que la segunda API es una interfaz CORBA para comunicarse con el servidor de P-BEAST.

3.5. EOS

EOS es un servicio de almacenamiento de datos, enfocado en reemplazar el antiguo sistema conocido como CASTOR. Al día de hoy CERN almacena 12 millones de archivos nuevos por mes en el sistema de almacenamiento CASTOR, y su volumen se espera que crezca tres veces hasta el año 2015 cuando alcanzará 0.125 EB. Alrededor de 2000 servidores de almacenamiento son utilizados para almacenar datos experimentales y de usuarios. [28] [29] [30] [31]

EOS es un servicio organizado en capas y el sistema de almacenamiento se divide en:

1. Almacenamiento de baja latencia en discos rígidos ofreciendo acceso tanto aleatorio como secuencial, lectura y escritura.

2. Almacenamiento de media latencia en discos rígidos para lectura secuencial y escritura en única vez.

3. Almacenamiento de alta latencia en cintas magnéticas para lectura secuencial de datos y escritura en única vez, hacia contenedores de datos (datasets).

El servicio P-BEAST almacena datos a largo plazo en EOS. Al momento de escribir el presente trabajo se disponen de 1.8TB de datos de P-BEAST almacenados en EOS:

```
$ du -sh eos-pbeast/
1.8T    eos-pbeast/
```

3.6. OKS

OKS es una base de datos en memoria utilizada para almacenar la configuración de los diferentes componentes del sistema de aplicaciones de ATLAS. La configuración se realiza mediante la confección de archivos XML. [25] [26]

Capítulo 4

Documentación de Shifter Assistant

4.1. Introducción

Al momento de comenzar a escribir el presente trabajo[1] no existía documentación de desarrollo y configuración de Shifter Assistant. Las siguientes secciones del presente capítulo responden a la necesidad de tener un documento escrito que facilite la tarea de escribir nueva directivas EPL para Shifter Assistant, como parte de este trabajo de tesina de grado. Este capítulo fue primero escrito en inglés para luego ser traducido al español por el autor.

4.2. Directorio de configuración

Asumiendo que `directives-pkg` es el directorio base de configuración, la organización básica de esta configuración para ejecutar Shifter Assistant es:

directives-pkg/data/ En este directorio se almacena la salida de las alertas en el writer tipo `file`.

directives-pkg/log/ En este directorio se almacena la salida de logs, configurable por medio del archivo `log4j.xml`.

directives-pkg/etc/cassandra.properties Archivo de configuración principal de Shifter Assistant.

directives-pkg/etc/log4j.xml Archivo de configuración de logs. Este archivo debe estar presente, de otra forma los logs no se generarán.

directives-pkg/etc/esper-replay.cfg.xml Configuración de Esper EPL.

[1] Tesina de grado de Licenciatura en Informática

directives-pkg/directives/ En este directorio se almacenan las directivas en formato XML.

4.3. Configuración

A continuación se encuentran ejemplos de los archivos `cassandra.properties` y `esper-replay.cfg.xml`.

```
# Directives to load
directives.filelist = common.xml, daqhlt.xml, runctrl.xml, sct.xml, trt.xml
  pixel.xml, rpc.xml, mdt.xml, ctp.xml, tile.xml, l1calo.xml

# Custom Esper configuration.
esper.configuration=etc/esper-replay.cfg.xml

# Redirect writers to local file.
directives.writer.mask = ers=file,jms=file,email=file

# Replay settings
replay.mode=true

# Take the run from ERS. The four values should be filled, with the same
# information as the log_manager program shows.
replay.run1.runnumber=227016

# Multiple runs from different partitions can be selected at the same time.
# SA Replay sorts the runs and replays them in sequence, one after another.
replay.run2.runnumber=227020

# Osiris configuration.
# Path defaults to "sareplay_osiris". Explicitly setting this property to
# empty disables the IS part of the injector.
#replay.osiris.path=
replay.osiris.path=/path/to/sareplay_osiris
#replay.osiris.args=
replay.osiris.partclass=ATLAS;RunInfo,ATLAS;RCStateInfo,initial;RunInfo,ini
```

Ejemplo de `esper-replay.cfg.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<esper-configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.espertech.com/schema/esper"
  xsi:schemaLocation=
```

```
    "http://www.espertech.com/schema/esper/esper-configuration-2.0.xsd">

  <!-- Warning: don't change this! -->
  <event-type name="Message" class="ch.cern.tdaq.cassandra.event.ERSEvent"/>
  <event-type name="ISEvent" class="ch.cern.tdaq.cassandra.event.ISEvent"/>
  <event-type name="Configuration"
    class="ch.cern.tdaq.cassandra.event.ConfigurationEvent"/>
  <auto-import import-name="ch.cern.tdaq.cassandra.replay.ISReader"/>
  <auto-import import-name="ch.cern.tdaq.cassandra.reader.ConfigReader"/>
  <auto-import import-name="ch.cern.tdaq.cassandra.config.Utils"/>
  <auto-import import-name="java.lang.Math"/>
  <auto-import import-name="ch.cern.tdaq.cassandra.utils.*"/>
</esper-configuration>
```

4.3.1. Descripción de configuración

directives.filelist La lista de los archivos XML de donde cargar las directivas.

esper.configuration Nombre de archivo de configuración de Esper, generalmente `esper-replay.cfg.xml`.

directives.writer.mask Máscara para redirigir la salida a otro *writer*.

replay.mode=true Habilita el injector de Replay Capítulo 6.1 para datos archivados. Cualquier valor diferente a "`true`" deshabilita el injector de *Replay*.

replay.runX.runnumber El número de *Run* Capítulo 3.1.1 para correr el *Replay* Capítulo 6.1.

Aquí, `X` en `runX` puede ser cualquier carácter o secuencia, y se puede repetir el atributo cambiando el valor de `X` por otro para tener varios números de Run en el Replay.

Ejemplo: `replay.run19.runnumber = 123456`.

replay.runX.refrn Utiliza run número de Run Capítulo 3.1.1 como referencia de tiempo de este Replay específico, de forma que el tiempo de inicio y fin del número de Run de `.runnumber` sea efectivamente el tiempo del Run de `.refrn`. Los datos asociados al número de Run en `.refrn` no son inyectados, solo se toma el tiempo de inicio y fin.

replay.osiris.path Ruta completa al ejecutable OSIRIS Capítulo 5.

replay.osiris.rawargs Argumentos al programa OSIRIS Capítulo 5. Pueden encontrarse detalles sobre los argumentos de OSIRIS en Capítulo 5.3.2.

Cuando esta opción se encuentra presente, todas las demás opciones de OSIRIS son ignoradas: tiempo de inicio y fin, partición y clase, y partición

y clase y atributo. Esta opción es útil para elegir de forma específica qué datos de P-BEAST se desean leer, cuando eligiendo partición y clase y atributo no es suficiente.

Asimismo, es posible utilizar el patrón especial "`[S]`" y "`[T]`" que será reemplazado por el tiempo de inicio y fin del Run elegido. Por ejemplo:

```
replay.osiris.rawargs=-s "[S]" -t "[T]" --replay-sql "SELECT ..."
```

replay.osiris.partclass Enumeración de las particiones y clases a leer de P-BEAST. Formato: "`P1;C1,P2;C2,P3;C3,...`"

replay.osiris.partclassattr Enumeración de las particiones, clases y atributos a leer de P-BEAST. Formato: "`P1;C1;A1,P2;C2;A2,P3;C3;A3,...`"

4.4. Salida y alertas

Existen tres salidas que Shifter Assistant produce: logs, métricas, y salidas propias de las directivas.

4.4.1. Logs

Los logs son los archivos de texto generados por SA sobre su funcionamiento interno. Cuando por ejemplo una directiva tiene un error de sintaxis, el mensaje de error se almacena en los logs. Por defecto la configuración de logs se define en el archivo `etc/log4j.xml` y la salida se almacena en el archivo `log/aal.err`.

4.4.2. Salida de alertas de directivas

El formato de salida depende del `writer` original utilizado. Dado que en el modo replay normalmente se redirigen los `writers` mediante el uso de la opción `mask` como `file`, el formato de salida depende del tipo original de `writer`. Los tipos de `writer` disponibles son `file`, `ers`, `jms`, `email`.

4.5. Directivas

Las reglas y directivas deben ser escritas en un archivo XML, y almacenado en el directorio `etc/directives`.

Un ejemplo de regla XML es:

```
<cassandra domain="Configuration directives">
  <configuration>
    <init-stmt> <!-- check for running partitions -->
      insert into Partitions select "ATLAS" as name,
        "A.B" as info
```

```
        from Configuration(name = 'POST')
    </init-stmt>

    <init-stmt>
      insert into ...
    </init-stmt>
  </configuration>

  <directive name="PartitionsState">
    <epl>
      SELECT * FROM ISEvent(partitionName='ATLAS')
      WHERE attributes('X').integer > attributes('Y').integer
      OUTPUT FIRST EVERY 2 minutes
    </epl>
    <listener type="alert">
      <domain>AAL.COM</domain>
      <severity> INFORMATION </severity>
      <message>Partition $partitionName$ state changes: $state$, error state: $inerror$<
      <action>
      </action>
      <details>true</details>
      <writer type="file">
        <partition>SA</partition>
        <severity>INFORMATION</severity>
        <append>true</append>
      </writer>
      <writer type="jms">
        <format>XML</format>
      </writer>
    </listener>
  </directive>
</cassandra>
```

4.5.1. XML tags

`<init-stmt>` Las directivas dentro de un bloque `<init-stmt>` son ejecutadas primero, al inicializar Shifter Assistant, para todas las directivas configuradas. Este tag es utilizado normalmente para inicializar alguna clase de stream filtrado.

`<directive name="name»` Una directiva de Shifter Assistant. Debe incluir el atributo `name` y éste debe ser único entre todas las directivas de todos los archivos XML.

`<epl>` La directiva EPL propiamente en formato Esper [13].

<listener type=."alert» Listeners para la directiva. Debe incluir el atributo type=."alert". Por el momento el único valor de type es alert.

<domain> Valor de "Domain" de la alerta para ser mostrado en la aplicación web.

<severity> Valor de "Severity" de la alerta para ser mostrado en la aplicación web.

<message> Valor de "Message" de la alerta para ser mostrado en la aplicación web.

<action> Valor de "Action" de la alerta para ser mostrado en la aplicación web.

<details> Valor de "Details" de la alerta para ser mostrado en la aplicación web.

<writer type="...» Un "writer" de la alerta. Indica el destino de la alerta.

Tipos de `writer`

file Enviar y escribir la alerta al archivo `data/alert_name.out`, donde *alert_name* es el atributo `name` en el tag `<directive>`.

<filename> El nombre de archivo donde guardar la alerta. Es por defecto el nombre de la alerta.

<append> Si es "true" el archivo no se elimina al reiniciar el servicio. Por defecto vale "false".

<format> El formato de salida. Los valores posibles son: `xml`, `html`, `text`.

Ejemplo:

```
<writer type="file">
  <format>TEXT</format>
  <filename>test1.out</filename>
</writer>
```

jms Escribir y enviar la alerta hacia JMS.

ers Escribir y enviar la alerta hacia ERS.

email Escribir y enviar la alerta por e-mail.

Configuración: Las opciones para el tipo `email` son:

<severity> La severidad, un string corto agregado como prefijo en el asunto del correo.

`<recipient>` Por cada destinatario debe existir un tag `<recipient>`.

Ejemplo:

```
<writer type="email">
  <severity>ERROR</severity>
  <recipient>1234@mail2sms.cern.ch</recipient>
  <recipient>Alejandro.Santos@cern.ch</recipient>
</writer>
```

4.6. Eventos y Tipos de datos disponibles en SA

A continuación se detallan algunos de los tipos de datos y eventos disponibles desde las directivas. Todos estos se encuentran como parte de la distribución estándar de Shifter Assistant, existentes dentro del motor del mismo. En el Apéndice 1 Capítulo 8 se encuentra la lista completa de flujos existente al día de la fecha dentro de las directivas del paquete `daq` de SA.

4.6.1. `TypedObject`

El tipo `TypedObject` es un tipo de datos básicos utilizado como wrapper de tipos de datos heterogéneos. Todos los valores de IS son representados como un `TypedObject`, y es responsabilidad de quien escribe las directivas conocer el tipo de datos del atributo que se quiere leer.

Un `TypedObject` debe convertirse a un tipo de datos Java concreto mediante el uso de alguno de los métodos de conversión presentes en la clase. Invocando una conversión diferente a la del tipo de datos contenido dentro de la instancia del `TypedObject` disparará un error de Java.

value Devuelve el valor de este `TypedObject` como un `Object`.

string Convierte este `TypedObject` a un `String`.

integer Convierte este `TypedObject` a un `Integer`, 32 bits signed integer.

long Convierte este `TypedObject` a un `Long`, 64 bits signed integer.

double Convierte este `TypedObject` a un `Double`, double precisión floating point.

boolean Convierte este `TypedObject` a un `Boolean`.

short Convierte este `TypedObject` a un `Short`, 16 bits signed integer.

float Convierte este `TypedObject` a un `Float`, single precisión floating point.

U32 Convierte este `TypedObject` a un `Long`, 32 bits unsigned integer.

U16 Convierte este `TypedObject` a un `Short`, 16 bits unsigned integer.

U8 Convierte este `TypedObject` a un `Short`, 8 bits unsigned integer.

stringArray Convierte este `TypedObject` a un `String` Array.

floatArray Convierte este `TypedObject` a un `Float` Array.

doubleArray Convierte este `TypedObject` a un `Double` Array.

integerArray Convierte este `TypedObject` a un `Integer` Array.

Ejemplos:

```
select *
from ISEvent( ... )
where attributes('n_disabled').integer = 0
```

4.6.2. `ISEvent`

Un objeto `ISEvent` representa un único evento de creación, actualización o eliminación de un objeto IS con todos sus atributos.

Las propiedades de un objeto `ISEvent` son:

server Devuelve el servidor IS del evento, tipo `String`.

name Devuelve el nombre del objeto, tipo `String`.

type Devuelve el tipo del objeto, tipo `String`.

time Devuelve el tiempo de creación del objeto, tipo `String`.

timeMicro Devuelve el tiempo de creación del objeto, tipo `long` integer en micro segundos (10^6).

timeCreation Devuelve el tiempo de creación del objeto.

rawAtt(name) Devuelve el atributo en crudo (*raw*) del objeto IS subyacente como un `Object` Java.

attributes(name) Dado un nombre *name* del atributo, devuelve el atributo como un `TypedObject`.

partitionName Devuelve el nombre de la partición del objeto, as `String`.

4.6.3. `Message`

Un `Message` representa un único evento ERS Capítulo 3.1.

chainedMessage Devuelve el `Message` encadenado.

messageID Devuelve el *Message ID*.

longIssuedDate Devuelve la fecha de emisión del evento como un `Long`.

formattedDate Devuelve la fecha formateada del evento.

machineName Devuelve el nombre de la máquina que generó el evento.

severity Devuelve la severidad del evento como un `String`. Valores posibles: `DEBUG`, `LOG`, `INFORMATION`, `WARNING`, `ERROR`, `FATAL`.

applicationName Devuelve el nombre de la aplicación que generó el evento.

partitionName Devuelve el nombre de la partición donde se generó el evento.

userName Devuelve el nombre de usuario que generó el evento.

messageTxt Devuelve el cuerpo del mensaje.

messageTxtRegexp Método de ayuda para procesar y cortar el campo `messageTxt` con una expresión regular. Devuelve el primer grupo que coincide con la expresión regular.

parameters(name) Devuelve el parámetro con nombre `name` como un `String`.

chained Devuelve un boolean que indica si este mensaje posee otro mensaje encadenado. El mensaje encadenado puede obtenerse con el atributo `chainedMessage`.

qualifiersContains(name) Devuelve `true` si el nombre `name` existe en la lista de *qualifiers* de este mensaje.

4.6.4. `ISReader`

La clase `ISReader` permite consultar y obtener objetos IS Capítulo 3.2 desde una directiva. A diferencia de registrar eventos en directivas que se disparen cuando llega un nuevo `ISEvent`, la clase `ISReader` permite hacer *polling* de IS desde cualquier evento. Los métodos disponibles son:

getInfoByIndex(String partition, String infoName, int parameterIndex) Devuelve el atributo por el número de índice `parameterIndex` del objeto `infoName` en la partición `partition`. El tipo de datos del objeto devuelvo es `TypedObject`.

getInfoByName(String partition, String infoName, String paramName) Devuelve el atributo por el nombre `paramName` del objeto `infoName` en la partición `partition`. El tipo de datos del objeto devuelvo es `TypedObject`.

getEventByName(String partition, String infoName) Devuelve el objeto completo `infoName` en la partición `partition`. El tipo de datos del objeto devuelvo es `ISEvent`.

Ejemplos:

A la llegada del evento ERS de tipo `rc::EndOfRun` se consulta el atributo `DataVolumeSaved` del objeto `A.B` de `IS`:

```
select DataVolumeSaved.double as DataVolumeSaved
from Message(partitionName='ATLAS', messageID='rc::EndOfRun'),
method:ISReader.getInfoByName("ATLAS","A.B",
    "DataVolumeSaved") as DataVolumeSaved
```

A la llegada de un evento del flujo `PartitionState` se consulta el objeto `A.B` de `IS`.

```
select event.attributes("value").string as Tag
from PartitionState(partitionName in ('ATLAS'), state = 'CONFIGURED',
    inerror = false) as ATLAS_state unidirectional,
method:ISReader.getEventByName("PixelInfr",
    "A.B") as event,
where event.attributes("value").string = 'PIT_MOD_STANDBY'
```

4.6.5. `ConfigReader`

La clase `ConfigReader` permite leer valores de configuración desde OKS Capítulo 3.6.

isComponentEnabled(String partition_name, String class_name, String object_id) Devuelve un objeto que representa el estado del componente: `notfound`, `disabled`, `enabled`. El valor devuelvo posee el atributo `status` de tipo `String` con uno de los tres valores anteriormente mencionados. Ejemplo:

```
select ...
from ISEvent(...),
method:ConfigReader.isComponentEnabled(part, "HW_Object", "some") as e
where enabled.status = "enabled"
```

isSegmentEnabled(String partition_name, String segment_id) Devuelve un objeto que representa el estado del segmento. El valor es de tipo `boolean` y para obtenerlo hace falta consultar el atributo `boolean` del objeto devuelto. Ejemplo:

```
select ...
from ISEvent(...),
method:ConfigReader.isSegmentEnabled(partitionName, "some_Seg") as seg
where seg.boolean
```

4.6.6. BunchGroupDecode

getWrongBunchGroup(Float[] hystogram) Esta función toma como entrada un array del volcado de un histograma representando la comparación entre las opciones esperadas de *Bunch Group* y las definidas por *LHC*. Si la entrada es diferente a cero, la función evalúa el Bunch Group correspondiente a la entrada incorrecta (la relación entrada-bunch group está definida por Joerg en un correo). Si todo se encuentra bien (todas las entradas son igual a cero), se devuelve -1. Para leer un valor es necesario pedir por el atributo value del valor devuelvo por esta función.[2] Ejemplo:

```
select ...
from ...,
method:BunchGroupDecode.getWrongBunchGroup(someFloatArray)
  as wrong_bc
where wrong_bc.value != -1
```

4.6.7. SCTUtils

Clase de ayuda para obtener información del sub-detector SCT [10] [3].

isModuleBusy(int bits, int slot) Devuelve si el módulo está ocupado (busy).

isModuleEnabled(int bits, int slot, int crate) Devuelve un boolean indicando el estado activado/desactivado de éste módulo (como una máscara de ocupada).

getDisabledModules(int bits, int crate) Devuelve un array de módulos desactivados.

getDisabledModulesBean(int bits, int crate) Devuelve un array de módulos desactivados.

getBusyModules(int bits) Devuelve un array de índices de posiciones de módulos ocupados, asume que `bits != 0`.

getBusyModulesBean(int bits) Devuelve un array de índices de positrones de módulos ocupados, asume que `bits != 0`.

[2] Documentación copiada literalmente de los comentarios en el código fuente.
[3] Documentación copiada literalmente de los comentarios en el código fuente.

4.7. Escribiendo directivas EPL para Shifter Assistant

A continuación se ofrecen algunas experiencias encontradas al momento de escribir directivas para Shifter Assistant.

4.7.1. Valores `Null`

El motor EPL de Esper interpreta los valores `null` de una forma especial. A diferencia de Java donde es posible preguntar directamente por el valor de una variable a fin de verificar si vale `null`, en EPL de Esper la expresión "`var=null`" vale `null` y por lo tanto `false`. Para verificar si una variable vale `null` en EPL es necesario utilizar el operador especial "`is`" en reemplazo del operador de igualdad "=". Por ejemplo, "`var is null`" o "`var is not null`".

4.7.2. Expresiones Regulares

El motor de Esper utiliza el paquete `java.util.regex` de Java para procesar expresiones regulares.

Si se desea procesar un `String` con expresiones regulares en una directiva Esper EPL hace falta tener en cuenta que el patrón punto-asterisco "`.*`" no validará un caracter de nueva línea "\n". En caso de querer hacer esto hace falta habilitar la opción `DOTALL` en el motor de expresiones regulares mediante el string "`(?s)`" como prefijo al comiendo del patrón.

Por ejemplo para corresponder cualquier string que contenga un caracter de salto de línea la expresión regular a usar será "`(?s).*`", y para corresponder cualquier string que contenga la cadena `CERN` habrá que escribir "`(?s).*CERN.*`".

4.7.3. Ser lo más preciso posible

Al momento de filtrar eventos en las condiciones de las directivas se deberá ser lo más preciso posible. Dado que en el mecanismo de replay se puede elegir el subconjunto de datos a inyectar al motor de Shifter Assistant, es posible que al momento de ejecutar el replay las directivas se disparen con los eventos inyectados pero al momento de ejecutar las directivas en producción con todos los eventos reales de IS, se obtengan eventos no deseados, dado que Shifter Assistant estará escuchando y ejecutando las directivas entre todos los datos de IS.

4.7.4. Orden de los eventos

Los eventos de tanto IS como ERS provienen de diferentes fuentes. Es por ello que se deberá tener en cuenta que, por diferentes efectos de temporización y funcionamiento de la red, los eventos llegarán en un orden impredecible.

Desde el punto de vista de Shifter Assistant los eventos pueden llegar en un orden, y desde el punto de vista del servicio de almacenamiento de P-BEAST pueden llegar en otro orden diferente.

4.8. Directivas de ejemplo

4.8.1. Directivas simples

Producir una alerta por cada evento de IS:

```
SELECT * FROM ISEvent(partitionName='ATLAS')
```

Producir una alerta cada 10 segundos retrasando todos los eventos. Cada 10 segundos se disparan todas las alertas producidas por todos los eventos:

```
SELECT * FROM ISEvent(partitionName='ATLAS').win:time_batch(10 sec)
```

Producir una alerta cada 10 segundos con el primer evento recibido. Esta directiva toma solamente el primer evento y descarta los restantes, sin otras condiciones puede tomar cualquier evento aleatorio:

```
SELECT * FROM ISEvent(partitionName='ATLAS').win:time_batch(10 sec).std:firstevent()
```

Sintaxis alternativa para tomar el primer evento cada 10 segundos. De vuelta, esta directiva puede tomar cualquier evento aleatorio dentro de la ventana de 10 segundos:

```
SELECT * FROM ISEvent(partitionName='ATLAS') OUTPUT FIRST EVERY 10 seconds
```

Combinar incorrectamente una llamada a `time_batch` con una sentencia `output` imprimirá todos los eventos:

```
SELECT * FROM ISEvent(partitionName='ATLAS').win:time_batch(10 sec)
OUTPUT FIRST EVERY 10 seconds
```

4.8.2. IS and ERS directives

Desde un flujo IS, disparar una alerta cada 2 minutos si el valor del atributo `X` es mayor que el valor del atributo `Y`.

```
SELECT * FROM ISEvent(partitionName='ATLAS')
WHERE attributes('X').integer > attributes('Y').integer
OUTPUT FIRST EVERY 2 minutes
```

Si se recibe un mensaje ERS de clase `A` y el objeto `RootController` se encuentra en estado B (`RunCtrl.RootController.state`), disparar una alerta. No hacerlo más seguido que una vez por minuto:

```
SELECT * FROM Message(messageID='A')
WHERE method:ISReader.getInfoByName("ATLAS", "RunCtrl.RootController", "state")
OUTPUT FIRST EVERY 1 minutes
```

Si `N` mensajes ERS de tipo `C` llegaron en los últimos `M` segundos, disparar una alerta:

```
SELECT COUNT(*) FROM TestEvent(name='QQQ').win:time_batch(M sec)
HAVING COUNT(*) >= N
```

Capítulo 5

OSIRIS

5.1. Acerca de OSIRIS

OSIRIS[1] es una herramienta para indexar, ordenar, combinar y leer la información de `IS` almacenada en `P-BEAST`.

OSIRIS fue desarrollado como parte del presente trabajo a fin de facilitar la lectura de los datos almacenados en P-BEAST. La motivación de su creación fue la de extender las herramientas de P-BEAST, donde originalmente solo se permite la lectura de un único atributo a la vez, mientras que el objetivo es el de poder reconstruir objetos completos de IS. Para ello es interesante leer todos los atributos mediante un único comando u operación.

La herramienta puede ser utilizada para leer todos los valores de los atributos de uno o varios objetos, independiente la clase o partición en la que se encuentra, e independiente de su tipo de datos. La herramienta combina y ordena los valores de cada atributo, produciendo como salida una única representación en común. `OSIRIS` permite, además y opcionalmente, la creación de un índice local de los meta-datos de P-BEAST a fin de obtener una rápida lectura y consulta de los mismos. Este índice no es necesario y en caso de no estar presente localmente la herramienta procederá a consultar los meta-datos hacia el origen de datos de P-BEAST.

`OSIRIS` permite leer un subconjunto arbitrario de datos almacenados en `P-BEAST` intentando replicar tanto como es posible la secuencia original de eventos IS durante el *Run* respectivo. A diferencia de las herramientas propias de P-BEAST, OSIRIS permite leer varios conjuntos de datos al mismo tiempo ordenándolos al momento de producir la salida, independiente del tipo de datos de los mismos, y sin importar si estos tipos de datos son diferentes.

OSIRIS se presenta por un lado como una herramienta de linea de comandos, donde es posible invocarla con el nombre `sareplay_osiris`. A su vez, también existe la API de OSIRIS para C++, `osiris::api`, donde la aplicación de línea de comandos fue desarrollada mediante esta misma API.

[1] Object Sorting Indexing and Retrieval for the Information Service

5.2. Índice OSIRIS

El índice es una base de datos `sqlite` almacenando los meta-datos de P-BEAST para una rápida consulta local. El índice actualmente almacena:

- Nombres de particiones
- Nombres de clases
- Nombres de atributos
- Nombres de objetos

Y sus relaciones. Para consultar información específica es posible escribir una consulta SQL y ejecutarla en la base de datos sqlite correspondiente. Por ejemplo para obtener la lista de todos los nombres de objetos que cumplen con el patrón ‘‘`%RunInfo%`’’ una posible consulta SQL puede ser:

```
$ sqlite3 osirisindex.sqlite
sqlite> SELECT * FROM object_name WHERE obj_name LIKE '%RunInfo%';
255974|RunParams.RunInfo
```

Versiones anteriores del índice almacenaban la relación entre los atributos y los archivos `.pb` físicamente disponibles en EOS. Con la inclusión de la interfaz CORBA se decidió suprimir esta relación a fin de simplificar la interfaz en común entre ambos medios de acceso a los datos de OSIRIS.

5.3. Herramienta `sareplay_osiris` de línea de comandos

La herramienta `sareplay_osiris` se encuentra implementada mediante la API C++ OSIRIS Capítulo 5.7.

5.3.1. Formato de salida de datos

El formato de salida en las operaciones de lectura de datos de OSIRIS es de 7 columnas separadas por un único caracter de tabulación “
t”, cada registro en su una línea terminada con un caracter de fin de línea “
n”, y en ninguna de las columnas aparecerán estos caracteres. Ejemplo:

```
$ sareplay_osiris --replay-part-class "ATLAS;RunInfo"
1392621918155933 1392621918156662 ATLAS RunInfo ["activeTime"]
  ["RunParams.RunInfo", "-1"] single:6:["0"]
1392621928157628 1392621928157628 ATLAS RunInfo ["activeTime"]
  ["RunParams.RunInfo", "-1"] single:6:["10"]
1392621938158404 1392621938158404 ATLAS RunInfo ["activeTime"]
  ["RunParams.RunInfo", "-1"] single:6:["20"]
```

```
$ sareplay_osiris --replay-part-class "ATLAS;CoreInfo"
1406911267138885 1406911652365504 ATLAS CoreInfo ["requestsHandled"]
  ["DF.ROS.ROS-TDQ-MUCTPI-00.Core", "-1"]
  array:6:8:["290548", "289209", "290352", "289454", "290500",
    "289541", "290527", "290356"]
1406911267138885 1406911652365504 ATLAS CoreInfo ["requestsRetried"]
  ["DF.ROS.ROS-TDQ-MUCTPI-00.Core", "-1"]
  array:6:8:["694836013", "695356550", "694885656", "694862795",
    "694636591", "695251963", "694994680", "694766806"]

$ sareplay_osiris --replay-part-class "ATLAS;Result"
1406812732652394 1406911646552436 ATLAS Result ["tags", "Tag", "value"]
  ["DQM.L1CaloRatesPar_Module_tau_08400009", "-1", "tags", "2"]
  single:256:["0"]
1406812732652394 1406911646552436 ATLAS Result ["tags", "Tag", "value"]
  ["DQM.L1CaloRatesPar_Module_tau_08400009", "-1", "tags", "3"]
  single:256:["0"]
1406812732652394 1406911646552436 ATLAS Result ["tags", "Tag", "name"]
  ["DQM.L1CaloRatesPar_Module_tau_08400009", "-1", "tags", "0"]
  single:14:["Mean"]
1406812732652394 1406911646552436 ATLAS Result ["tags", "Tag", "name"]
  ["DQM.L1CaloRatesPar_Module_tau_08400009", "-1", "tags", "1"]
  single:14:["Number_of_bins_equal_zero"]
```

- La primer columna indica el tiempo de creación del valor en formato de *epoch UNIX*, tiempo transcurrido desde `1970-01-01 01:00:00`, en microsegundos. Ejemplo: 1392621918155933.

- La segunda columna indica el tiempo de última actualización del valor en formato de *epoch UNIX*, en microsegundos. Ejemplo: 1392621918156662.

- La tercer columna indica el nombre de la partición. Ejemplo: `ATLAS`.

- La cuarta columna indica el nombre de la clase. Ejemplo: `RunInfo`.

- La quinta columna indica el nombre del atributo actualizado. El formato es una lista de strings en formato JSON y con caracteres especiales escapados con una barra invertida antes. El primer elemento de la lista indica el nombre del atributo directamente presente en la clase de la segunda columna, y los siguientes elementos indican los atributos anidados. Ejemplo: `[.ªctiveTime"]`.

- La sexta columna indica el nombre del objeto actualizado. Al igual que la columna que indica el atributo, esta columna es también una lista de strings en formato JSON. El primer elemento indica indica el nombre del

objeto cuyo tipo de datos es directamente el de la clase de la cuarta columna, el segundo elemento es siempre "`-1`", y los restantes indican el nombre de los atributos anidados. Ejemplo: `["RunParams.RunInfo", "-1"]`.

- La séptima columna indica el valor concreto actualizado.

  El formato es: "`array:tipo:valor`", donde `array` es un string de dos posibles valores: "`single`" que indica que el elemento es un valor simple, o "`array`" que indica que el valor es un array. "`tipo`" es el código numérico de tipo que se corresponde con los tipos de P-BEAST Capítulo 5.7.3. Además de los tipos de datos de P-BEAST existen dos códigos adicionales propios de OSIRIS:

OSIRIS	Número
`osiris::OSIRIS_DOUBLE_AS_LONG`	265
`osiris::OSIRIS_FLOAT_AS_INT`	257

  La representación en string en base 10 de un número punto flotante `float` de 32 bits o un `double` de 64 bits no es perfecta. Por ejemplo el número $1{,}1$ en `double` se representa realmente en base 10 como $1{,}1000000000000000088817841970012523233890533447265625$. Es por esto que se decidió representar los valores de estos dos tipos como números enteros re-interpretando los bits en memoria, en lugar de `float` o `double`, como `int32_t` o `int64_t`. De esta forma al convertir el número a un `string` no se pierde su valor original en la conversión. Por ejemplo:

```
1417642191652642 1417642191652642 ATLAS DFSummary
  ["DCML1AveRate"]
  ["DF.DFSummary", "-1"]
  single:256:["4644159680891885178"]
1417654317463719 1417654317463719 ATLAS DFSummary
  ["HLTRate"]
  ["DF.DFSummary", "-1"]
  single:257:["1134670643"]
```

  La conversión de un número `float` o `double` a `int32_t` o `int64_t` se hace en C++ de la siguiente manera:

```
int64_t encodeOsirisDouble(double num) {
  union {
    double d;
    int64_t i;
  } n;
  n.d = num;
  return n.i;
}
```

```
int32_t encodeOsirisFloat(float num) {
  union {
    float f;
    int32_t i;
  } n;
  n.f = num;
  return n.i;
}
```

Para convertir los números nuevamente a un tipo de datos `double` o `float` en Java se hace lo siguiente:

```
float decodeOsirisDouble(String number) {
  return Double.longBitsToDouble(Long.parseLong(
      number));
}

float decodeOsirisFloat(String number) {
  return Float.intBitsToFloat(Integer.parseInt(
      number));
}
```

Los atributos en P-BEAST pueden contener valores simples, o arrays de valores simples. Un valor simple es representado con el indicador "`single`" y el valor será un array de un único elemento, representado como un string en formato JSON, con caracteres especiales escapados con "\" y el valor propiamente como una representación en string del mismo. Ejemplo: `single:6:["20"]` indica un valor simple de tipo `U32` cuya representación en string es 20, y `single:14:["one\ntwo\three\n"]` indica un valor de un `string` con tres caracteres especiales de avance de línea "\n" dentro.

Los arrays son representados con el indicador "`array`", y su representación es similar a "`single`", el campo valor tendrá como prefijo la cantidad de elementos del array, y el valor será un array de strings donde cada elemento es la representación en string de cada elemento del array. Ejemplo: `array:6:3:["123", "456", "789"]`.

5.3.2. Argumentos de ejecución

`-r [ --repository-dir ] arg` **Default**: `$HOME/eos/atlas/atlascerngroupdisk/tdaq-opmon/pbeast`

Ruta donde se encuentran los archivos de `.pb` de P-BEAST en EOS. No es necesario que el directorio sea EOS, solo que los archivos respeten la distribución esperada por P-BEAST.

`--pbeast-backend arg` **Default**: `eos`

Valores posibles: `eos`, `corba`. Indica el backend a utilizar para acceder a los datos de P-BEAST.

`--pbeast-ipc-partition arg` **Default**: `initial`

Indica el nombre de la partición donde se encuentra el servicio de P-BEAST corriendo. Por lo general suele ser `initial`.

`--pbeast-server-name arg` **Default**: `pbeast-server`

Nombre del servicio IPC del servidor de P-BEAST. Por lo general suele ser `pbeast-server`.

`-i [ --index-dir ] arg` **Default**: `/tmp/$USER/osirisindex.sqlite`

Ruta donde se encuentra la base de datos `sqlite` utilizada como índice de meta datos de OSIRIS.

`-s [ --since ] arg` **Default**: `2014-02-17 00:00:01`

Tiempo de inicio de la lectura de datos. El formato debe respetar `ISO 8601`.

`-t [ --till ] arg` **Default**: `2014-02-17 23:59:59`

Tiempo de fin de la lectura de datos. El formato debe respetar `ISO 8601`.

`--describe-object arg` Muestra la descripción de los meta-datos del objeto desde los datos almacenados en el índice, mostrando nombre de partición, clase, atributos y los tipos de datos de cada uno.

`--describe-part-class arg` Muestra la descripción de los metadatos de a partir del nombre de una partición y clase. Índice opcional. El formato es: "`particion;clase`". Ejemplo:

```
$ sareplay_osiris --describe-part-class "ATLAS;RunInfo"
ATLAS RunInfo ["activeTime"] Type=u32 is_array=0
```

`--replay-one-object arg` Ejecuta la lectura de todos los datos asociados al objeto indicado dentro de la ventana de tiempo. Índice requerido.

`--replay-sql arg` Ejecuta la lectura de todos los datos asociados al objeto indicado dentro de la ventana de tiempo. Índice requerido.

`--replay-part-class arg` Ejecuta la lectura de todos los datos asociados al tipo de datos indicado. Por ejemplo, indicando "`ATLAS;RunInfo`" se procederá a la lectura de todos los atributos de todos los objetos de tipo `RunInfo` en la partición `ATLAS`.

`--replay-part-class-attr arg` Similar a `--replay-part-class` aunque es posible indicar de manera más precisa la lista de atributos a leer.

`--replay-pcao-file arg` Equivalente a `--replay-part-class-attr` con la diferencia que el argumento indica un archivo donde, línea a línea, se tienen los atributos a leer.

`--list-object-names-like arg` Permite listar los nombres de objetos en el índice que respeten el patrón del argumento, utilizando el caracter especial ‘‘`%`’’ como un comodín. Índice requerido. Por ejemplo, `--list-object-names-like '%ATCN%'` muestra todos los objetos cuyo nombre contiene el substring `ATCN`.

`--list-objects-at arg` Obtiene una lista de todos los nombres de objetos en la partición y clase indicada. Formato: `--list-objects-at "Part;Class"`.

`--show-object arg` Reconstruye, a partir de los valores del intervalo de tiempo indicado, el objeto indicado por su nombre. Índice requerido.

`--list-partitions` Obtiene una lista de las particiones disponibles. Índice opcional.

`--update-index arg` Actualiza el índice. Las opciones son: “`basic`”: actualiza solo el árbol de metadatos (nombres de particiones, clases, atributos); y “`objects`”: primero actualiza el índice como la opción “`basic`” y luego actualiza la lista de nombres de objetos.

`--no-index` Omite el uso del índice aún cuando se encuentre.

`--vacuum` Ejecuta el comando `VACUUM` en la base de datos SQLite. De-fragmenta la base de datos.

`--analyze` Ejecuta el comando `ANALYZA` en la base de datos SQLite. Mejora la eficiencia de la base de datos actualizando los meta-datos del planificador de consultas.

`--example-sql` Muestra una consulta SQL de ejemplo para ser utilizada con el comando `--replay-sql`.

`-h [ --help ]` Muestra el actual mensaje de ayuda.

`-v [ --verbose ]` Muestra en la salida información adicional sobre las operaciones que OSIRIS se encuentra realizando.

5.4. Atributos anidados de P-BEAST

En IS, los tipos de datos de los atributos de los objetos pueden ser tanto un tipo de datos nativo (como entero de 32 bits con signo o string), o también otra

clase de IS, formando una relación 1:1 entre estos dos. Por ejemplo un objeto de tipo `A` puede tener un atributo llamado `x` de tipo `B`, donde `B` es un tipo de datos de `IS`. A este tipo de relación donde un atributo de un objeto no es de un tipo de datos nativo de `IS` se le conoce como *Tipos de Datos Anidados*.

Desde el lado de `IS` su uso es simple, la definición del tipo de datos debe incluir la relación hacia el tipo de datos anidado. Sin embargo, en `P-BEAST` su uso es diferente. Los únicos valores que tiene sentido almacenar son los valores concretos, números, strings, etc.

La forma que P-BEAST tiene de almacenar los atributos nativos de un atributo anidado es multiplicando el atributo de la clase padre con cada atributo de la clase hijo, y almacenando los valores en la clase padre. Esto es importante ya que al momento de querer acceder a su valor hace falta conocer cuáles son los atributos con tipos anidados, cuáles son los atributos del tipo anidado, y multiplicar el atributo padre con cada uno de los hijos.

Por ejemplo, en la partición `ATLAS` existe una clase `Result` con tres atributos: `objects`, `status` y `tags`, y donde uno de estos atributos, `tags`, es de un tipo anidado. El tipo de datos de `tags` es la clase `Tag`, donde `Tag` posee dos atributos concretos: `name` de tipo `string`, y `value` de tipo `double`. `tags` es un array de `Tag`.

Para reconstruir un objeto `Result` completo hace falta pedir la lectura de cuatro atributos:

1. `objects`
2. `status`
3. `tags/Tag/name`
4. `tags/Tag/value`

Para leer con P-BEAST un atributo anidado hace falta construir el nombre de atributo especial con la forma: `nombre de atributo padre, /, nombre de tipo anidado, /, nombre de atributo de tipo anidado`. Los atributos `tags/Tag/name` y `tags/Tag/value` son atributos compuestos pero que para P-BEAST son atributos propios de `Result`, y no de `Tag`.

5.5. Orígenes de datos

La primer abstracción de OSIRIS por sobre P-BEAST es la del origen de datos. P-BEAST permite leer información almacenada tanto desde archivos `.pb` persistentes en EOS, como también a partir de un servidor P-BEAST mediante el protocolo de comunicación CORBA.

Sin embargo, dependiendo del origen de datos, las herramientas necesarias son diferentes, y a nivel de programación la API a utilizar son también diferentes. Para leer datos desde EOS desde la línea de comandos hace falta utilizar `pbeast_`

`read_repository`, y para obtener información desde CORBA hace falta utilizar el comando `pbeast_read_server`. Ambas herramientas poseen argumentos de línea de comandos diferentes, y si bien el formato de salida es similar no se garantiza que lo sea a futuro.

Es por esto que en OSIRIS se decidió crear una abstracción de origen de datos denominada `osiris::api`, una clase abstracta en C++ con dos instancias concretas: `osiris::api_eos` y `osiris::api_corba`. Como su nombre lo indica, la primera permite leer datos desde EOS y la segunda desde CORBA. Asimismo la herramienta intenta detectar el orígen de datos apropiado para utilizar.

5.6. Combinando diferentes time series

La API de P-BEAST permite la lectura de un único atributo a la vez, para uno o varios objetos. Los objetos a leer se pueden indicar mediante una expresión regular y permitiendo la lectura de valores de varios objetos, pero para el caso de los nombre de partición, clase y atributo es necesario valores únicos y concretos. Dado que el objetivo es poder construir un objeto de IS completo en un instante en el tiempo pasado, hace falta obtener los valores de todos los atributos de cada objeto deseado.

Para cada tipo de datos, OSIRIS consulta mediante la API de P-BEAST la lista de atributos, y para cada uno ejecuta la lectura de los valores. Dado que la lectura se hace sobre un intervalo de tiempo y no para un instante dado, el valor devuelto por P-BEAST no es uno único sino una secuencia de varios valores. Esta secuencia de valores es la que P-BEAST llama `Time Series`, que representa cada uno de los valores tomados a lo lardo del tiempo por cada uno de los atributos de cada uno de los objetos.

Para ordenar la totalidad de las *Time Series* en una única línea de tiempo, OSIRIS toma cada *Time Series* y las coloca en una cola de prioridad, ordenando las *Time Series* entre sí por la el instante de tiempo del primer elemento en la secuencia. De esta forma se tiene una estructura ordenada de todas las *Time Series*.

A fin de obtener el primer elemento de la secuencia final, se toma la primer *Time Series* de la cola de prioridad y la remueve. Toma el primer elemento de esta *Time Series*, lo marca como leído, y avanza un puntero hacia el siguiente elemento de la *Time Series*. Si la *Time Series* posee más de un elemento se procede a volver a insertar esta en la cola de prioridad, utilizando como valor de tiempo el mismo del nuevo siguiente elemento. Este procedimiento se repite para cada *Time Series* de la cola de prioridad hasta que ya no queden elementos en ella.

En pseudocódigo se puede ver de la siguiente forma:

```
def procesar(inicio, fin):
  /* 'Q' es una Cola de Prioridad de Time Series */
```

```
Q = new ColaDePrioridad

para cada atributo A a leer de cada objeto O:
  P = partición de A
  C = clase de A

  /* 'valores' es una Time Series de (P, C, A, O) */
  valores = api.leer(P, C, A, inicio, fin, O)

  si valores.tamanio > 0:
    Q.insertar(valores)

mientras Q no sea vacia:
  /* 'ts' es una Time Series */
  ts = Q.menor_elemento
  Q.eliminar_menor_elemento

  imprimir_elemento(ts.siguiente_elemento)

  ts.avanzar
  si ts.tamanio > 0:
    Q.insertar(ts)
```

Este procedimiento es el utilizado para obtener la secuencia ordenada de cambios sobre cada atributo de cada objeto, permitiendo la reconstrucción de los objetos de IS almacenados en P-BEAST. La reconstrucción propia de objetos se realiza dentro del Inyector de Shifter Assistant.

5.7. OSIRIS C++ API

5.7.1. Rangos de tiempo

La mayoría de las funciones de `osiris::api` aceptan rangos de tiempo como dos parámetros de tipo `uint64_t`. La unidad de medida de tiempo es es microsegundos, donde 1 segundo $= 10^6$ microsegundos.

5.7.2. Time Series

P-BEAST utiliza el concepto de *serie temporal*[2] para almacenar una serie de valores que cambian en el tiempo. Un objeto de tipo `pbeast::SeriesData` es un único valor con la información temporal relacionada, tanto la fecha de creación del valor como la fecha de última validez del mismo.

[2] *time series* en inglés

5.7.3. Tipos de datos de P-BEAST

Los tipos de datos reconocidos por P-BEAST junto con su equivalencia de tipo de datos de C++ son[3]:

P-BEAST	Número	C++
pbeast::BOOL	0	bool
pbeast::S8	1	int8_t
pbeast::U8	2	uint8_t
pbeast::S16	3	int16_t
pbeast::U16	4	uint16_t
pbeast::S32	5	int32_t
pbeast::U32	6	uint32_t
pbeast::S64	7	int64_t
pbeast::U64	8	uint64_t
pbeast::FLOAT	9	float
pbeast::DOUBLE	10	double
pbeast::ENUM	11	int32_t
pbeast::DATE	12	uint32_t
pbeast::TIME	13	uint64_t
pbeast::STRING	14	std::string
pbeast::OBJECT	15	
pbeast::VOID	16	pbeast::Void

El caso de `pbeast::VOID` es un caso especial para indicar un tipo de datos sin valor asociado, y es utilizado para la construcción de time series que no poseen valores de datos.

En el caso de `pbeast::OBJECT` la lectura de datos se hace obteniendo la definición de tipo de datos del atributo anidado Capítulo 5.4.

5.7.4. Métodos de la interfaz `osiris::api`

list_partitions

Dado un repositorio devuelve una lista de particiones Capítulo 3.2.1. El repositorio se toma al momento de crear la instancia concreta de `osiris::api`.

list_classes

Dado el nombre de una partición devuelve una lista de nombres de clases Capítulo 3.2.1.

list_attributes

Dado el nombre de una partición y el de una clase devuelve una lista de nombres de atributos Capítulo 3.2.1.

[3]Valores obtenidos de los encabezados de la librería de P-BEAST

list_objects

Dado el nombre de una partición, clase y atributo y un rango de tiempo devuelve los nombres de objetos con al menos un atributo actualizado en el rango indicado. Existen dos variantes de esta función:

1. La primera toma un rango de tiempo como dos números `uint64_t` en microsegundos.

2. La segunda toma el nombre de un archivo `pbtime`.

list_attribute_types_series

Dado el nombre de una partición, clase y atributo devuelve la *serie* de tipos de datos de datos del atributo. Dado que en IS un mismo atributo puede cambiar de tipo de datos en el tiempo, y por ejemplo un `int32` cambiar por un `string`, esta función permite conocer estos cambios, con el momento del cambio.

La serie de valores es de pares `std::pair<bool, std::string>` donde el primer elemento del pair es un boolean que indica si el tipo de datos es un array, mientras que el segundo es la representación en string del nombre del tipo de datos. Estos valores corresponder a los tipos de datos reconocidos por P-BEAST.

list_attribute_descriptions

Dado el nombre de una partición, clase y atributo devuelve un vector de SeriesData de descripciones, donde cada elemento de la SeriesData indica el intervalo de tiempo en donde esa descripción es válida. Una descripción es un breve string que menciona en inglés el propósito del atributo.

get_attribute_type

Dado el nombre de una partición, clase y atributo y un rango de tiempo, devuelve el tipo de datos del atributo en este rango de tiempo.

Existen dos variantes de este método, el primero devuelve el tipo de datos en forma de un `int32_t` que coincide con los tipos de datos de `pbeast` Capítulo 5.7.3, y la segunda variante devuelve el nombre del tipo de datos con su representación en `string`. La ventaja de utilizar el string es que si el tipo de datos es anidado, el valor devuelto será el nombre del tipo de datos, mientras que la conversión a `int32_t` será simplemente como un `pbeast::OBJECT`.

Es importante tener en cuenta que el tipo de datos de un atributo puede variar en el tiempo, y si el rango de tiempo solicitado coincide con el cambio del tipo de datos, una excepción de tipo `std::runtime_error` será disparada.

expand_nested_attributes

Dado el nombre de una partición, clase y atributo y un rango de tiempo, devuelve la lista con los nombres de los atributos anidados.

Por ejemplo, si una clase `X` tiene un atributo anidado a cuyo tipo es `Y`, donde `Y` tiene dos atributos `j` y `k`, la llamada a:

```
expand_nested_attributes(P, ‘‘X’’, ‘‘a’’, s, t)
```

devolverá

```
[[‘‘Y’’, ‘‘j’’], [‘‘Y’’, ‘‘k’’]].
```

describe_class

Dado el nombre de una partición, clase y un rango de tiempo, devuelve la lista completa de sus atributos con el tipo de datos de cada uno. Si el parámetro `expand_nested` es `true` devolverá también la descripción de los atributos anidados, recursivamente.

describe_attribute

Dado el nombre de una partición, clase, atributo y un rango de tiempo, devuelve una lista con la descripción del atributo: nombre y tipo de datos. Si el parámetro `expand_nested` es `true` devolverá también la descripción de los atributos anidados, recursivamente.

get_attribute_real_type

Dado el nombre de una partición, clase, y una descripción de atributo, devuelve el verdadero tipo de datos y nombre de atributo del atributo original. Verdadero significa que, si el atributo indicado es un atributo anidado, el valor devuelto es el del último de la secuencia, mientras que si el atributo no es anidado el valor devuelto es el mismo original.

Por ejemplo, si una clase `X` tiene un atributo anidado a cuyo tipo es `Y`, donde `Y` tiene dos atributos `j` y `k`, la llamada a:

```
get_attribute_real_type(P, ‘‘X’’, [‘‘a’’, ‘‘Y’’, ‘‘j’’])
```

devolverá “Y” como nombre de clase verdadera, y “j” como nombre de atributo verdadero.

read_attribute_variant

Lee los valores asociados en P-BEAST. Existen tres versiones de este método. La primer versión no requiere indicar explicitamente el tipo de datos del atributo.

Dado el nombre de una partición, clase, atributo, una lista de nombres de objetos, el tipo de datos en forma de `pbeast::DataType`, un boolean indicando si el valor es un array, y un rango de tiempo, completa dentro del mapa `variant_data` por cada nombre de objeto una instancia de series para leer los valores solicitados.

El tipo de datos indicado por parámetro debe ser correcto, no existe una validación interna para que los valores sean apropiados.

read_attribute_re_variant

Lee los valores asociados en P-BEAST. A diferencia del método anterior, permite indicar los nombres de los objetos como una expresión regular. La eficiencia de este método es la mayor cuando se requiere leer los valores de un subconjunto de objetos.

Dado el nombre de una partición, clase, atributo, una expresión regular que representa los nombres de los objetos a leer, el tipo de datos en forma de `pbeast::DataType`, un boolean indicando si el valor es un array, y un rango de tiempo, completa dentro del mapa`variant_data` por cada nombre de objeto que sea aceptado por la expresión regular, una instancia de series para leer los valores solicitados.

Encabezado de `osiris::api`

```
namespace osiris {
class api {
public:
    virtual ~api();

    virtual std::list<std::string> list_partitions() =
        0;

    virtual std::list<std::string> list_classes(
        const std::string& part_name) = 0;

    virtual std::list<std::string> list_attributes(
        const std::string& part_name,
        const std::string& class_name) = 0;

    virtual void list_objects(
```

```
        const std::string& part_name,
        const std::string& class_name,
        const std::string& attr_name,
        uint64_t since,
        uint64_t till,
        std::set<std::string>& objects) = 0;

    virtual void list_objects(
        const std::string& part_name,
        const std::string& class_name,
        const std::string& attr_name,
        const std::string& pbtime_name,
        std::set<std::string>& objects) = 0;

    virtual std::vector<pbeast::SeriesData<
            std::pair<bool, std::string>>>
            list_attribute_types_series(
        const std::string& part_name,
        const std::string& class_name,
        const std::string& attr_name) = 0;

    virtual std::vector<pbeast::SeriesData<
            std::string>>
            list_attribute_descriptions(
        const std::string& part_name,
        const std::string& class_name,
        const std::string& attr_name) = 0;

    virtual void get_attribute_type(
        const std::string& part_name,
        const std::string& class_name,
        const std::string& attr_name,
        uint64_t since,
        uint64_t till,
        int32_t& data_type,
        bool& is_array) = 0;

    virtual void get_attribute_type(
        const std::string& part_name,
        const std::string& class_name,
        const std::string& attr_name,
        uint64_t since,
        uint64_t till,
        std::string& type_str,
```

```
    bool& is_array) = 0;

/** Given a nested-type description of part, class
    , attr, returns the list of nested attributes
    with its type.
 *
 * Example: given "ATLAS, Result, tags" returns ["
    Tag/name", "Tag/value"].
 */
virtual std::vector<std::vector<std::string> >
    expand_nested_attributes(
    const std::string& part_name,
    const std::string& class_name,
    const std::string& attr_name,
    uint64_t since,
    uint64_t till);

/** Given a part and class returns the description
    of its attributes. If expand_nested==true, it
    recursively queries nested types.
 */
virtual std::vector<class_description>
    describe_class(
    const std::string& part_name,
    const std::string& class_name,
    uint64_t since,
    uint64_t till,
    bool expand_nested);

/** Given a part and class returns the description
    of the attribute. If expand_nested==true, it
    recursively queries nested types.
 */
virtual std::vector<class_description>
    describe_attribute(
    const std::string& part_name,
    const std::string& class_name,
    const std::string& attr_name,
    uint64_t since,
    uint64_t till,
    bool expand_nested);

virtual void get_attribute_real_type(
    const std::string& part_name,
```

```
        const std::string& class_name,
        const std::vector<std::string>&
            attr_name_parts,
        std::string& real_class_name,
        std::string& real_attr_name);

    virtual series_ptr read_attribute_variant(
        const std::string& part_name,
        const std::string& class_name,
        const std::string& attr_name,
        const std::string& obj_name,
        uint64_t since,
        uint64_t till) = 0;

    virtual series_ptr read_attribute_variant(
        const std::string& part_name,
        const std::string& class_name,
        const std::string& attr_name,
        const std::string& obj_name,
        pbeast::DataType type,
        bool is_array,
        uint64_t since,
        uint64_t till) = 0;

    virtual void read_attribute_variant(
        const std::string& part_name,
        const std::string& class_name,
        const std::string& attr_name,
        const std::vector<std::string>& obj_names,
        pbeast::DataType type,
        bool is_array,
        uint64_t since,
        uint64_t till,
        std::map<std::string, series_ptr>&
            variant_data) = 0;

    virtual void read_attribute_re_variant(
        const std::string& part_name,
        const std::string& class_name,
        const std::string& attr_name,
        const std::string& obj_name_regexp,
        pbeast::DataType type,
        bool is_array,
        uint64_t since,
```

```
        uint64_t till,
        std::map<std::string, series_ptr>&
            variant_data) = 0;
};
}
```

Capítulo 6

Modificaciones al Shifter Assistant

6.1. Modo Replay

El Modo Replay en Shifter Assistant desactiva la infraestructura de comunicación `CORBA` de `IS` y `ERS`, y toma datos almacenados en la base de datos SQL para `ERS`, P-BEAST para `IS`, y el sistema de back-up de `OKS`. Para iniciar Shifter Assistant en modo replay hace falta utilizar el comando `sa_replay` ubicado en la distribución de software de `ATLAS TDAQ` e indicando por parámetro el directorio donde se encuentra la configuración Capítulo 4 de ejecución de Shifter Assistant, teniendo el modo *replay* activado.

6.1.1. Tomando datos desde ERS

Los datos de ERS se encuentran almacenados en una base de datos SQL. Es posible explorar los mensajes pasados mediante la aplicación `log_manager` Capítulo 3.1, la cual permite visualizar los mensajes pasados.

Dado un número específico de Run y partición, el modo Replay toma todos los datos, los ordena, reconstruye las estructuras de datos internas de Shifter Assistant e inyecta nuevamente los datos hacia el motor Esper de Shifter Assistant.

6.1.2. Tomando datos desde IS

Los datos de IS se encuentran almacenados en P-BEAST. Manualmente, es posible leer estos datos mediante las aplicaciones de línea de comandos, o mediante el uso de `sareplay_osiris`.

El modo Replay de SA toma las ventanas de tiempo de los números de Run elegidos de ERS para elegir los datos a leer desde P-BEAST, reconstruir las estructuras de datos internas de IS desde el punto de vista de Shifter Assistant, e inyectar los datos hacia el motor Esper de Shifter Assistant.

La reconstrucción de los datos de IS no es perfecta. Primero, IS es una base de datos distribuida que no ofrece garantía alguna en el orden de recepción de los eventos en los diferentes clientes. Una máquina escuchando datos de IS puede recibir los eventos en un orden diferente a otra. Por ejemplo, Shifter Assistant puede recibir los eventos en un orden diferente al del servicio de almacenamiento de P-BEAST.

Segundo, P-BEAST no almacena valores duplicados. Esto significa que en un evento repetitivo con los mismos valores de atributos solamente se almacenará el primer evento, y solo se tendrá en cuenta el tiempo del primer y último evento. Las directivas que dependan de los tiempos de llegada de eventos con valores repetitivos no funcionarán de la misma manera en el modo replay que en el modo normal de Shifter Assistant.

El modo Replay intenta solucionar este problema de eventos repetitivos simulando la repetición de eventos cuyo tiempo de validez sea mayor a un minuto. Si el tiempo entre la creación y la actualización de un valor es mayor a un minuto se simula la llegada de un evento que se repite cada un minuto, primero en el momento original de creación, después un minuto más tarde, y por último en el tiempo de su última actualización.

Internamente, para combinar los diferentes eventos de particiones, clases, atributos, objetos y flujos IS y ERS, el inyector de Replay utiliza una cola de prioridad de listas de eventos. Cada lista es una secuencia de eventos ordenados, y el próximo evento a ser inyectado es la cabeza de la lista en el frente de la cola de prioridad.

La aplicación `sareplay_osiris` convierte todos los tipos de datos a una representación de tipo String, y el inyector de Replay toma el String y lo convierte nuevamente a un tipo de datos Java.

La clase `ISReader` en Shifter Assistant permite leer tanto un objeto completo como atributos específicos en objetos concretos. Para esto es necesario mantener en memoria el último estado de todos los objetos y, a medida que los cambios evolucionan, actualizar esta estructura. Asimismo hubo que tener en cuenta que cuando finaliza un run y comienza otro hace falta limpiar esta estructura.

6.1.3. Tomando configuración desde OKS

La clase `ConfigReader` permite leer valores de configuración de OKS Capítulo 3.6 desde directivas EPL. Afortunadamente el sistema OKS almacena un histórico de la configuración por cada Run Number, por lo que al momento de iniciar el inyector de Replay y antes de comenzar a leer datos desde P-BEAST, se accede a esta base de datos, se descarga la versión de la configuración OKS, y se inicializa el estado de OKS dentro de Shifter Assistant con esta configuración.

Capítulo 7

Conclusiones y trabajo a futuro

7.1. Conclusiones

En este trabajo se ha presentado un completo mecanismo de pruebas para el Shifter Assistant de ATLAS que permite a los usuarios de diferentes dominios insertar sus conocimientos en la aplicación y verificar la correctitud de sus directivas. La reproducibilidad de las pruebas permite colaborar entre colegas en la etapa de desarrollo de software y también permite introducir una procedimiento formal de aprobación de código listo para ser puesto en producción. Permitir la distribución de sentencias EPL a una diversa comunidad de usuarios y expertos, y permitirá sobreponer el típico problema de herramientas de manejo de conocimientos: mantener el conocimiento actualizado.

La disponibilidad de buenas herramientas para ayudar en el proceso de desarrollo es esencial en la tarea de desarrollo. Tener un contexto para ejecutar directivas que permita iterar sobre variaciones de la misma directiva y/u orígenes de datos es un recurso importante en la experiencia de desarrollo. Tener un ambiente de pruebas en un estado de reproducir sin ambiguedades situaciones interesantes permite hacer un ajuste fino de las condiciones para la generación de alertas.

Más allá que los detalles de implementación son específicos a esta aplicación en particular, el enfoque de ofrecer pruebas extensivas mediante la entrada de datos del sistema real puede ser genéricamente aplicado a una variedad de sistemas. Mantener un historial de eventos en almacenamiento externo para una posterior reconstrucción, falsificando las entradas y salidas de datos, aplicando la re-ordenación de eventos en el tiempo, y ofreciendo utilidades de software para facilitar el procedimiento de reiniciar el motor de procesamiento de eventos a partir de eventos anteriores o falsos es un esquema que puede ser facilmente adaptado a otros sistemas con problemáticas similares.

El éxito de una herramienta de pruebas depende fuertemente en su uso: idealmente, el usuario no debe tener la necesidad de conocer en detalle el funcionamiento interno del mecanismo de pruebas.

7.2. Trabajo a futuro

El trabajo descripto en los capítulos anteriores promovió el desarrollo de nuevos proyectos: la creación de una interfaz web para facilitar su uso por parte de los usuarios, y la posibilidad de utilizar la API de OSIRIS desde Python y Java.

A partir de comentarios de los usuarios, fue decidido simplificar el proceso requerido a seguir. Los usuarios debían seguir una serie de pasos para ejecutar el workflow de pruebas; por esta razón una aplicación web para efectuar las pruebas de las directivas de SA fue creada como forma de facilitar el acceso a la herramienta.

La aplicación web consiste en una interfaz de usuario simple donde el usuario debe ingresar una identificación de la ejecución, el intervalo de tiempo sobre el cual efectuar la operación de lectura de datos, el subconjunto de tipos de datos de P-BEAST, y las directivas de SA definidas por el usuario a ser probadas. El usuario puede también definir un tiempo límite para la ejecución, teniendo por defecto un límite de una hora. Si la ejecución demora más tiempo, la ejecución del proceso es terminada.

Escrito enteramente en Python con el framework Django, la aplicación dispone de los módulos para interactuar con el usuario y con una base de datos donde la información de ejecución es almacenada, y desde donde un proceso en segundo plano se encarga efectivamente de la ejecución del proceso.

Cuando los usuarios ingresan los campos para comenzar una nueva ejecución, la información es almacenada en una base de datos. Luego, uno de los procesos de segundo plano ejecutándose en el servidor toma la siguiente tarea sin procesar desde la base de datos, crea un nuevo entorno de ejecución para ejecutar el proceso de *SAReplay*, y comienza el proceso de ejecución mientras se registra su salida y estado.

El proceso en segundo plano puede terminar la ejecución bajo varias condiciones: la común siendo cuando la ejecución propiamente termina. El proceso en segundo plano también puede decidir terminar la ejecución porque se alcanzó el tiempo límite, o porque demasiado espacio en disco fue utilizado.

Cada ejecución es identificada unívocamente por un identificador; para cada ejecución toda la información relevante y archivos asociados son almacenados en la base de datos, y los usuarios pueden consultar y buscar entre las ejecuciones pasadas, tanto las asociadas a su propia cuenta de usuario como la de otros usuarios.

Con respecto a la API de OSIRIS, fue motivada inicialmente por parte de este trabajo a fin de mejorar la eficiencia de lectura de datos. Inicialmente el inyector Java de SA ejecutaba el subproceso de OSIRIS invocando el programa `sareplay_osiris` y procesando la salida estándar del programa se reconstruían los datos necesarios. Este enfoque presentaba el problema de tener que hacer *parsing* de la salida de OSIRIS Capítulo 5.3.1, por lo que mediante una API en Java se ofrecía una simplificación de su uso.

La API en Java fue hecha como un *binding* de la API de C++, donde mediante el uso de la herramienta de generación de *bindings* SWIG, permitió poder ejecutar las funciones de la API C++ desde Java. SWIG [39] se encarga de generar automáticamente el código necesario para hacer el puente entre el código C++ y el requerido por Java. Asimismo la herramienta SWIG también permite generar código para poder de forma equivalente ejecutar las llamadas desde Python, donde la única diferencia es cambiar el lenguaje de salida generado en la configuración del programa.

La API Python de OSIRIS permitió en el grupo de trabajo producir rápidamente una serie de pequeñas aplicaciones específicas para la lectura de datos desde P-BEAST, que de otra forma hubiese sido necesario procesar la salida de los programas de líneas de comandos, o escribiendo los programas necesarios en C++.

Capítulo 8

Apéndice 1: Listado de flujos existentes en Shifter Assistant

8.1. Listado de flujos existentes en Shifter Assistant

Mediante el uso de una sentencia `INSERT INTO` en directivas de Esper, es posible crear tipos de datos de eventos personalizados. El paquete daq de directivas de Shifter Assistant ya incluye varias sentencias encargadas de crear flujos de eventos filtrados o agregados, los cuales son:

```
ATLASRunStarted {
  class java.lang.String
    state;
}
ATLASRunStopped {
  class java.lang.String
    state;
}
ATLASState {
  class java.lang.String
    status;
}
AppRunOut {
  class java.lang.String
    time;
  class java.lang.String
    application_name;
  class java.lang.String
    membership;
  class java.lang.String
    state;
}
BMode {
  class java.lang.String
    mode;
}
CTPBusy1percent {
  class java.lang.Double
    GlobalBusy;
}
CTPBusy5percent {
  class java.lang.Double
    GlobalBusy;
}
Configuration {
  class java.lang.String
    name;
}
ControllersHanging {
  class java.lang.String
    HostName;
```

```
  class java.lang.String
     Controller;
}
DetectorsBusy {
  class java.lang.String
     Detector;
  class java.lang.Double
     BusyFraction;
}
DetectorsBusyIS {
  class java.lang.String
     Detector;
  class java.lang.String
     ISserver;
  class java.lang.String
     busy_mask;
  class java.lang.String
     busy_value;
  class java.lang.Integer
     index;
}
EFDAverages60secs {
  class java.lang.Double
     EFDelQueueAverage;
  class java.lang.Double
     HeapOccAv;
  class java.lang.Double
     EventsInsideAverage;
  class java.lang.Integer
     EventsInside;
  class java.lang.Double
     BackpressureAverage;
  class java.lang.Long
     NofEFDs;
}
EFDCounters {
  class java.lang.String
     efname;
  class java.lang.String
     countername;
  class java.lang.Integer
     value;
}
ISEvent {
  interface java.util.Map
     attributes;
  class java.lang.String
     name;
  class ch.cern.tdaq.
     cassandra.event.
     ISEvent$OP operation;
  class java.lang.String
     partitionName;
  class java.lang.String
     server;
  class java.lang.String
     time;
  long timeCreation;
  long timeMicro;
  class java.lang.String
     type;
  class java.lang.Object
     rawAtt;
}
LHCBeamMode {
  class java.lang.String
     BeamMode;
}
Message {
  class java.lang.String
     applicationName;
  class java.lang.Boolean
     chained;
  class ch.cern.tdaq.
     cassandra.event.
     ERSEvent
     chainedMessage;
  class java.lang.String
     formattedDate;
  class java.lang.String
     issuedDate;
  class java.lang.Long
     longIssuedDate;
  class java.lang.String
     machineName;
  class java.lang.String
     messageID;
```

```
  class java.lang.String
     messageTxt;
  interface java.util.Map
     parameters;
  class java.lang.String
     partitionName;
  class [Ljava.lang.String
     ; qualifiers;
  class java.lang.String
     severity;
  class java.lang.String
     userName;
  class java.lang.String
     messageTxtRegexp;
  boolean
     qualifiersContains;
}
PartitionState {
  class java.lang.String
     state;
  class java.lang.String
     partitionName;
  class java.lang.Boolean
     inerror;
  class java.lang.String
     errorCode;
  class java.lang.String
     errorDesc;
}
Partitions {
  class java.lang.String
     name;
  class java.lang.String
     info;
}
ProblematicROS {
  class java.lang.String
     ROS;
}
ProblematicROS10s {
  class java.lang.String
     ROS;
  class java.lang.String
     Message;
}
ProblematicTRTROSBursts {
  long size;
  class java.lang.String
     ROS;
  class java.lang.String
     Message;
}
RampOverThreshold {
  class java.lang.Float
     rampvalue;
  class java.lang.String
     thresholdname;
}
RunNumbers {
  class java.lang.Integer
     RN;
}
SCTBusyMask {
  class java.lang.Integer
     crate;
  class java.lang.Integer
     bitmask;
}
SCTBusyState {
  class java.lang.Integer
     crate;
  class java.lang.Integer
     bitmask;
}
SCTCrateBadLinkState5sec {
  class java.lang.String
     Crate;
  class java.lang.Integer
     BadLinks5sec;
}
SCTCrates {
  class java.lang.String
     crate;
  class java.lang.String
     is_name;
}
SCTFirmware {
```

```
  class java.lang.String
     firmware;
  class java.lang.String
     version;
  class java.lang.String
     ROD_id;
}
SCTLinkErrorsCounter {
  class java.lang.String
     Crate;
  class java.lang.Integer
     NofBadLinks;
}
SFOFSUsage {
  class java.lang.Long
     occupancy;
  class java.lang.String
     fsname;
  class java.lang.String
     sfoname;
}
Statistics {
  class java.lang.String
     name;
  class java.lang.Long
     counter;
}
ch.cern.tdaq.cassandra.
    config.
    FSUsageStatistics {
  interface java.util.Map
     usage;
}
ch.cern.tdaq.cassandra.
    reader.TypedObject {
  class java.lang.Boolean
     boolean;
  class java.lang.Double
     double;
  class [Ljava.lang.Double
     ; doubleArray;
  class java.lang.Float
     float;
  class [Ljava.lang.Float;
      floatArray;
  class java.lang.Integer
     int;
  class java.lang.Integer
     integer;
  class [Ljava.lang.
     Integer; integerArray
     ;
  class java.lang.Long
     long;
  class java.lang.Short
     short;
  class java.lang.String
     string;
  class [Ljava.lang.String
     ; stringArray;
  class java.lang.Integer
     u16;
  class java.lang.Long u32
     ;
  class java.lang.Integer
     u8;
  class java.lang.Object
     value;
}
ch.cern.tdaq.cassandra.
    utils.ArrayBean {
  class [Ljava.lang.
     Integer; array;
  class java.lang.Integer
     size;
}
ch.cern.tdaq.cassandra.
    utils.BooleanBean {
  boolean boolean;
}
ch.cern.tdaq.cassandra.
    utils.ComponentStatus {
  class java.lang.String
     status;
}
ch.cern.tdaq.cassandra.
    utils.IntegerBeam {
```

```
  class java.lang.Integer
     value;
}
com.espertech.esper.client
    .metric.EngineMetric {
  class java.lang.String
     engineURI;
  long inputCount;
  long inputCountDelta;
  long scheduleDepth;
  long timestamp;
}
com.espertech.esper.client
    .metric.StatementMetric
     {
  long cpuTime;
  class java.lang.String
     engineURI;
  long numInput;
  long numOutputIStream;
  long numOutputRStream;
  class java.lang.String
     statementName;
  long timestamp;
  long wallTime;
}
```

Capítulo 9

Glosario

ATLAS *A Toroidal LHC ApparatuS*, Aparato Toroidal del LHC, uno de los detectores del LHC Capítulo 2.2.

CERN Organización Europea para la Investigación Nuclear Capítulo 2.1.

ERS *Error Reporting Service*, Servicio de Reporte de Errores Capítulo 3.1.

EOS *Exabyte Scale Storage*, sistema que permite almacenar grandes volúmenes de información en CERN Capítulo 3.5.

EPL *Event Processing Language*, el lenguaje utilizado por Esper para la definición de directivas Capítulo 3.3.

Experto Miembro de ATLAS que conoce en detalle el funcionamiento de algún componente y/o sub-detector del experimento.

HLT *High Level Trigger*, Trigger de Alto Nivel. El HLT es el sistema encargado de un primer filtrado de los eventos de física producidos en ATLAS Capítulo 2.3.

IS *Information Service*, Servicio de Información, utilizado como mecanismo para la comunicación en alto nivel de las diferentes aplicaciones de ATLAS Capítulo 3.2.

LHC *Large Hadron Collider*, Gran Colisionador de Hadrones, el mayor colisionador de partículas existente al día de hoy, administrado y mantenido por CERN Capítulo 2.1.

OKS *Object Kernel Support*, base de datos utilizada en ATLAS para la configuración de los sub componentes de software Capítulo 3.6.

OSIRIS *Object Sorting Indexing and Retrieval for the Information Service*, desarrollo del autor para facilitar la lectura de datos históricos almacenados en P-BEAST Capítulo 5.

P-BEAST *Persistent Back-End for the ATLAS TDAQ On-line Information Service*, base de datos encargada de almacenar valores históricos de IS Capítulo 3.4.

Run 1 Período de tiempo comprendido entre 2009 y 2013 en el que el LHC estuvo en funcionamiento y los detectores adquiriendo datos de física.

SA *Shifter Assistant*, motor de procesamiento de directivas con eventos operacionales de ATLAS Capítulo 3.3.

Shifter Miembro de ATLAS que dedica parte de su trabajo en tareas de guardia telefónica o de estancia en la sala de control del experimento.

TDAQ *Trigger and Data Acquisition*, sistema encargado de la lectura y primer filtrado de los datos de física del detector ATLAS Capítulo 2.3.

Bibliografía

[1] LHC: The guide - http://cds.cern.ch/record/1092437?ln=en

[2] What is ATLAS? - http://atlas.ch/what_is_atlas.html

[3] ATLAS Data Taking Operations Model for Run-2 - https://edms.cern.ch/document/1348082/1

[4] ATLAS high-level trigger, data-acquisition and controls : Technical Design Report - http://cds.cern.ch/record/616089/

[5] Evolution of the ReadOut System of the ATLAS experiment - http://cds.cern.ch/record/1710776

[6] Anders, Gabriel and Avolio, Giuseppe and Kazarov, Andrei and Lehmann Miotto, Giovanna and Santos, Alejandro and Soloviev, Igor (2015) "A Validation System for the Complex Event Processing Directives of the ATLAS Shifter Assistant Tool", 21st Conference in High Energy and Nuclear Physics 2015, https://cds.cern.ch/record/2016465

[7] Anders, Gabriel and Avolio, Giuseppe and Kazarov, Andrei and Lehmann Miotto, Giovanna and Santos, Alejandro and Soloviev, Igor (2015) "A Validation System for the Complex Event Processing Directives of the ATLAS Shifter Assistant Tool", 21st Conference in High Energy and Nuclear Physics 2015, (SLIDES) https://cds.cern.ch/record/2005098

[8] "Upgrade of the ATLAS Control and Configuration Software for Run 2", 21st Conference in High Energy and Nuclear Physics 2015, CERN. https://cds.cern.ch/record/2007850

[9] "ATLAS TDAQ System Administration: an overview and evolution" http://atlas-tdaq-sysadmin.web.cern.ch/atlas-tdaq-sysadmin/public/talks/ISGC2013-compressed.pdf

[10] "ATLAS SCT" http://www.atlas.ch/sct.html

[11] Networks for ATLAS Trigger and Data Acquisition. - http://cds.cern.ch/record/933762/

[12] Esper project - http://esper.codehaus.org/

[13] Esper 4.11.0 documentation - http://esper.codehaus.org/esper-4.11.0/doc/reference/en-US/html/index.html

[14] Esper Quick Start - http://esper.codehaus.org/tutorials/tutorial/quickstart.html

[15] Intelligent monitoring and fault diagnosis for ATLAS TDAQ: a complex event processing solution - https://cds.cern.ch/record/1442916

[16] The AAL project: automated monitoring and intelligent analysis for the ATLAS data taking infrastructure - https://cds.cern.ch/record/1403070

[17] A Persistent Back-End for the ATLAS Online Information Service (P-BEAST) - https://cds.cern.ch/record/1402973

[18] New Persistent Back-End for the ATLAS Online Information Service - https://cds.cern.ch/record/1703443

[19] Information Service (IS) - http://atlas-tdaq-monitoring.web.cern.ch/atlas-tdaq-monitoring/IS/Welcome.htm

[20] Applications of CORBA in the ATLAS prototype DAQ - http://cds.cern.ch/record/446324

[21] CORBA - http://www.corba.org/

[22] Error Reporting Service (ERS) High level design - https://edms.cern.ch/document/459790/1.0

[23] Error Reporting Service (ERS) Architectural analysis and low level design - https://edms.cern.ch/document/544930/1

[24] The Error Reporting in the ATLAS TDAQ System - https://cds.cern.ch/record/1950302

[25] The ATLAS DAQ System Online Configurations Database Service Challenge - http://dx.doi.org/10.1088/1742-6596/119/2/022004

[26] OKS Documentation - http://atlas-onlsw.web.cern.ch/Atlas-onlsw/components/configdb/docs/oks-ug/2.0/pdf/OksDocumentation.pdf

[27] A Scalable and Reliable Message Transport Service for the ATLAS Trigger and Data Acquisition System - https://cds.cern.ch/record/1703434?ln=en

[28] The EOS project - http://eos.cern.ch/

[29] Exabyte Scale Storage at CERN - http://dx.doi.org/10.1088/1742-6596/331/5/052015

[30] XROOTD - http://xrootd.org/index.html

[31] CASTOR - http://castor.web.cern.ch/

[32] J. DeVale and P. Koopman, "Robust Software—No More Excuses," Proc. IEEE Int'l Conf. Dependable Systems and Networks, pp. 145-154, June 2002. http://dx.doi.org/10.1109/DSN.2002.1028895

[33] L. Williams, G. Kudrjavets, N. Nagappan, "On the Effectiveness of Unit Test Automation at Microsoft" http://research.microsoft.com/apps/pubs/default.aspx?id=102349

[34] Robert M. O'Keefe, Daniel E. O'Leary "Expert system verification and validation: a survey and tutorial"

[35] Per Runeson "A Survey of Unit Testing Practices" http://dx.doi.org/10.1109/MS.2006.91

[36] Pedro Meseguer and Alun D. Preece (1995). Verification and validation of knowledge-based systems with formal specifications. The Knowledge Engineering Review, 10, pp 331-343. doi:10.1017/S0269888900007542.

[37] Lee, Sunro, and Robert M. O'Keefe. "Developing a strategy for expert system verification and validation." Systems, Man and Cybernetics, IEEE Transactions on 24.4 (1994): 643-655.

[38] Edwards, Stephen H. "A framework for practical, automated black-box testing of component-based software" Software Testing, Verification and Reliability 11.2 (2001): 97-111.

[39] Simplified Wrapper and Interface Generator - SWIG - http://www.swig.org/