Ingeniería guiada por Modelado y Simulación de Eventos Discretos: Metodología y Caso de Estudio en la Red de Datos del Experimento ATLAS

Matías Bonaventura[1] y Rodrigo Castro[1]
[1] Departamento de Computación, Facultad de Ciencias Exactas y Naturales.
Universidad de Buenos Aires, Argentina
{mbonaventura, rcastro}@dc.uba.ar

Resumen. En este trabajo presentamos una metodología iterativa e incremental para desarrollo de proyectos de ingeniería guiados por modelado y simulación (M&S) formal. Basándonos en el marco formal DEVS (Discrete Event Systems Specification), y sumando prácticas estándar en proyectos de software, se obtienen modelos que dan respuesta a preguntas de diseño y optimización de redes de datos. Presentamos una aplicación práctica de la metodología a un caso de estudio de ingeniería de redes, en el contexto de un experimento científico a gran escala: el experimento ATLAS (Máquina de Dios) en el cual científicos de todo el mundo intentan responder preguntas sobre el origen del universo. Estas redes absorben en tiempo real datos de experimentos físicos, y por su criticidad se someten a diseños exhaustivos y mejoras continuas. Sin embargo, por la escala del proyecto, estas redes están disponibles para ensayos solo esporádicamente. Mostramos como asistimos su diseño mediante técnicas de modelado y simulación, hipotetizando sobre el sistema real cuando se encuentra inaccesible.

1 Introducción

Con el avance de la Ingeniería de Software las metodologías que proveen un marco para el ciclo de vida de proyectos probaron ser en la comunidad una herramienta clave y hoy en día presente en cualquier proyecto de desarrollo de software. En simultáneo, diversas técnicas de modelado y simulación (M&S) se han vuelto cada vez más utilizadas para el diseño de sistemas complejos, en particular en entornos donde es muy difícil o imposible predecir el comportamiento de un sistema ante cambios.

Sin embargo, existe comparativamente poca experiencia en metodologías para desarrollo de proyectos de hardware y software basados en M&S formal. DEVS es el formalismo más general para el modelado de sistemas a eventos discretos [1, 2, 3] adoptado en muchas áreas para diseño y análisis de sistemas complejos [4, 5]. Además de suministrar herramientas formales para la definición de modelos provee una metodología para el análisis del sistema, la definición del marco experimental y la verificación de la simulación. Sin embargo, si bien esta metodología hace foco en la correctitud formal del proceso, no tiene en cuenta características particulares de un proyecto de software como elicitación de requerimientos, comunicación de resultados, planificación de fases, calidad y flexibilidad del producto, etc.

En el presente trabajo proponemos una metodología para desarrollo de proyectos de ingeniería de sistemas guiados por M&S, basada en el marco formal DEVS, sumando prácticas habituales de proyectos de desarrollo de software.

Presentamos aquí una experiencia/caso de estudio en ingeniería de redes de datos para un experimento de física a gran escala (detector de partículas ATLAS [6] en el CERN [7]) siguiendo la metodología propuesta. Este proyecto presenta grandes desafíos desde el punto de vista del M&S debido al complejo sistema que se debe modelar, así

ATL-DAQ-PROC-2015-024
29 June 2015

como también requerimientos exigentes de tiempos de entrega, calidad y flexibilidad de la herramienta desarrollada, presentación de resultados a los interesados (en su mayoría científicos), comunicación con actores interdisciplinarios, etc.
Por medio de una metodología iterativa se validan frecuentemente requerimientos y resultados de simulaciones. Mediante un proceso incremental se gana conocimiento sobre el sistema real, añadiendo gradualmente características relevantes al modelo y generando simulaciones que replican al sistema real desde las primeras iteraciones.
La definición de diferentes tipos de ciclos permite enmarcar los esfuerzos durante diferentes etapas de un proyecto de M&S, que difieren de las etapas típicas de un proyecto de software. La utilización de fases en cada ciclo permite desarrollar y ejecutar modelos a la vez que se extienden y perfeccionan las herramientas de base utilizadas para M&S. En este trabajo presentamos avances obtenidos en modelos de simulación que responden preguntas acerca de la red de datos estudiada, junto a mejoras implementados en las herramientas de M&S, ambos progresos guiados por las necesidades y tiempos del proyecto ATLAS.
En las secciones 4.1 y 4.2 se presentan dos iteraciones de construcción de un modelo que reproduce fehacientemente el comportamiento del sistema real, se discuten las decisiones metodológicas y de modelado, y se presenta una comparación entre los resultados de simulación y los medidos en el sistema real. En la sección 4.3 se presenta una iteración de hipotetización: mediante M&S se realiza el diseño y puesta a prueba de una mejora en algoritmos de control de tráfico de red, hasta su implementación en el sistema real con resultados satisfactorios predichos por la simulación.

2 Conceptos Preliminares

2.1 Sistema real bajo estudio: El Experimento ATLAS en CERN.

El Gran Colisionador de Hadrones (LHC [8]) es el acelerador de partículas más grande del mundo, de 27 kilómetros de circunferencia, donde se encuentran 4 grandes detectores de partículas: ATLAS [6], CMS [9], ALICE [10] y LHCb [11]. En 2013 los detectores se apagaron para someterlos a mejoras y actualizaciones (Long Shutdown 1, LS1) hasta el nuevo Run2 en 2015. ATLAS es un detector de partículas de propósito general donde se producen colisiones de conjuntos de protones de muy alta energia para la búsqueda de nuevas evidencias físicas (Bosón de Higgs, dimensiones extra, etc.). Cada colisión entre protones se denomina “Evento”, generando desintegraciones de nuevas partículas cuyas trayectorias son digitalizadas por detectores para su análisis posterior. El throughput en la red supera los 60 TeraBytes/Segundo. Para asimilar toda esta información, ATLAS utiliza un sofisticado sistema de filtrado (TDAQ) que decide en tiempo real qué subconjunto de datos debe ser preservado. Un primer nivel de filtrado (First Level Trigger o L1) filtra eventos de una muestra inicial de 40 millones a 100.000 por segundo. Los eventos aceptados en L1 se almacenan temporalmente en el Sistema de Lectura (Read Out System, ROS), en estructuras lógicas llamadas Fragmentos, para luego ser accedidos por el segundo nivel de filtrado (High Level Trigger, HLT). En HLT, algoritmos físicos analizan los fragmentos, reteniendo solo 100 eventos interesantes por segundo.
TDAQ y la red de datos HLT-ROS será nuestro *sistema real bajo estudio*, para el que detallaremos técnicas y procesos para su modelado y simulación.
Aplicaciones y Red de Datos en el High Level Trigger. La **Fig 1** muestra la interac-

ción entre diferentes aplicaciones que componen HLT. Después que el L1 almacena un nuevo evento en el ROS, notifica al servidor High Level Trigger Supervisor (HLTSV) que dicho evento está listo para continuar siendo procesado. El HLTSV asigna eventos a servidores de procesamiento denominados Trigger Processing Units (TPU). Cada TPU ejecuta una aplicación llamada Data Collection Manager (DCM) que centraliza la comunicación entre la TPU y el resto del sistema (ROS, HLTSV y DataLoggers). El DCM crea instancias de la aplicación Processing Unit (PU) -una por cada core disponible, entre 8 y 24-. Cada PU finalmente ejecuta el algoritmo que define si el Evento es físicamente relevante y debe ser almacenado permanentemente o descartado. La relación lógica PU-Evento es 1 a 1 (un Evento no se paraleliza).

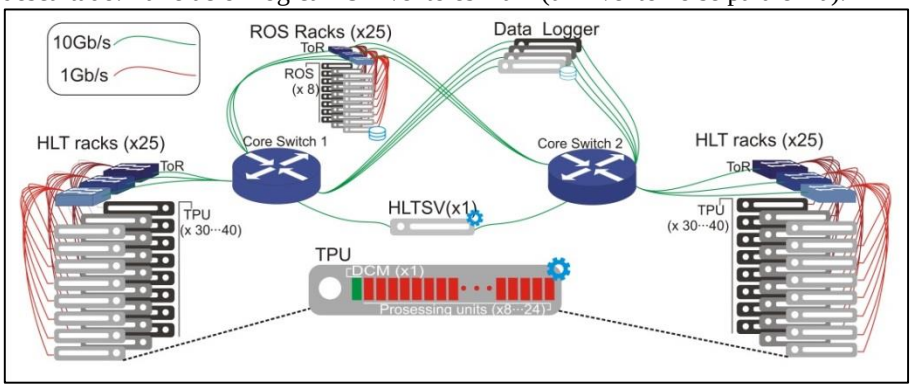

Fig. 1. Aplicaciones y Topología de la granja de servidores para el HLT de TDAQ.

Las aplicaciones ejecutan sobre una red ethernet a 1 y 10 Gbps, donde 2 routers centrales y ~100 switches interconectan ~3000 servers multicore mediante protocolos TCP/IP. La **Fig 1** muestra un diagrama de esta red. La granja contiene ~50 racks de servers TPU y ~25 racks con nodos del ROS. Los racks de TPU contienen ~40 servers (aplicaciones DCM y PU) y los racks del ROS contienen 8 servers (un total de ~200 nodos). En cada rack, los servers se conectan a su Top of Rack Switch (ToR) con enlaces de 1Gbps. Los ToR y los servers ROS se conectan los core switches a 10Gb/s.

2.1 DEVS para el modelado de redes de datos

DEVS [1, 4] es un formalismo de modelado y simulación genérico basado en la teoría general de sistemas. Permite describir exactamente cualquier sistema discreto y aproximar numéricamente sistemas continuos. La especificación formal de los modelos provee medios para su manipulación analítica y permite independencia en la elección del lenguaje a utilizar para su implementación [2]. Un sistema modelado con DEVS se describe como una composición jerárquica de modelos acoplados (CM) y modelos atómicos (M) definidos por tuplas matemáticas como muestra la **Fig 2**.

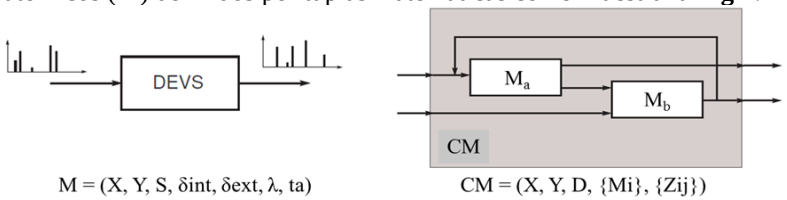

Fig. 2. Modelos atómicos y acoplados DEVS

Los modelos acoplados definen la estructura del sistema (interconexión de modelos acoplados y atómicos). Los modelos atómicos definen el comportamiento dinámico.

Para M, cada posible estado del modelo $s \in S$ tiene asociado un tiempo de vida definido por la función $ta: S \rightarrow R_0^+$. Cuando el modelo se encuentra en el estado s=s1 en el instante de tiempo t_1 durante $ta(s_1)$ unidades de tiempo, el sistema realiza una transición interna δint que modifica el estado del modelo a $s2=\delta_{int}(s1)$. $\delta_{int}: S \rightarrow S$ se denomina función de transición interna. En este caso, se produce al mismo tiempo un evento de salida $y_1=\lambda(s_1)$. La función $\lambda: S \rightarrow Y$ se denomina función de salida.

Cuando un modelo recibe un evento de entrada $x_1 \in X$, se produce una transición externa que modifica instantáneamente el estado del modelo al estado $s_4 = \delta_{ext}(s_3,e,x_1)$, donde s_3 es el estado del modelo en el momento en que se recibe el evento de entrada, y *e* es el tiempo transcurrido desde la última transición (con $e \leq ta(s_3)$). La función $\delta_{ext}: S x R_0^+ x X \rightarrow S$ se denomina función de transición externa. El algoritmo de simulación DEVS para modelos M y sus posibles acoplamientos CM, es universal, no ambiguo, sencillo de implementar, e independiente de lenguajes de programación.

DEVS Vectoriales. Existen muchas extensiones y especializaciones de DEVS para atacar diversas necesidades (e.g. Cell-DEVS [12] para autómatas celulares, PDEVS [13] para modelado de paralelismo, etc.). En particular nos interesa Vectorial DEVS (VDEVS) [14] que permite representar sistemas de gran escala de manera gráfica compacta. Un modelo vectorial es un vector de modelos DEVS clásicos idénticos que pueden diferir en sus parámetros. Formalmente la estructura de un modelo vectorial se define según: $VD = \{N, X_V , Y_V , P, \{M_i\}\}$, donde N es la dimensión del vector, Xv es el conjunto de eventos vectoriales de entrada, Yv es el conjunto de eventos vectoriales de salida, P es el conjunto de parámetros y cada M_i es un modelo atómico DEVS clásico. Para la interacción entre modelos vectoriales y no vectoriales se utilizan modelos multiplexores y demultiplexores.

Herramienta de modelado y simulación: PowerDEVS. El modelo de TDAQ se ha desarrollado con la herramienta PowerDEVS [15], que ofrece una interfaz gráfica para definir y estructurar modelos DEVS mediante diagramas de bloques y un editor para definir en C++ las 4 funciones dinámicas de la tupla M. PowerDEVS incluye bibliotecas de modelos reutilizables. Utilizamos una biblioteca de redes de datos (colas, servidores, generadores de tráfico y una implementación de TCP [16]) y la extendimos para nuestro caso de estudio. PowerDEVS posee interfaz nativa con Scilab [15], un paquete abierto para cómputo numérico alternativo a Matlab.

En cuanto a simuladores establecidos específicos para redes de datos podemos mencionar OMNeT++[17], ns2[18] y OPNET[19], que modelan con gran detalle nodos reales de red y protocolos incluyendo versiones de TCP. Estas no eran opciones viables para TDAQ. Por el gran tamaño del sistema, la simulación detallista torna la simulación inviable en tiempos razonables. Se debe poder elegir paso a paso el nivel de abstracción suficiente para cada pregunta a contestar. Por otro lado, un requisito a futuro implica realizar simulaciones híbridas (por eventos discretos y con flujos continuos), capacidad disponible en DEVS (métodos numéricos QSS [20]) e implementada de manera avanzada en PowerDEVS para redes de datos [21].

3 Requerimientos y Metodología de M&S para TDAQ

Para cualquier caso de estudio que pueda plantearse, existen en TDAQ contextos y requerimientos que demandan una metodología de trabajo robusta, flexible y que explote al máximo las características de las herramientas de M&S adoptadas.

3.1 Contexto y Requerimientos

- **Contexto**: La granja de filtrado HLT de TDAQ es un sistema en evolución constante. Está sometida a cambios permanentes, la infraestructura de hardware y los algoritmos de control son mejorados continuamente. **Impacto**: Es difícil o imposible predecir el impacto de estos cambios antes de ponerlos en producción. Los servidores y componentes de red no se instalan hasta el momento de su adquisición, por lo que es imposible conocer con antelación los impactos que tendrán en las aplicaciones y en el flujo de datos.
- **Contexto**: La granja HLT es utilizada por diversos grupos (centenas de científicos) por lo cual sólo está disponible para pruebas de TDAQ aproximadamente una de cada seis semanas. **Impacto**: Esto retrasa las tareas de puesta a prueba de nuevos algoritmos de control, que se mejoran continuamente pero no pueden ser verificados por completo hasta que el sistema esté disponible.

A continuación se definen los requerimientos elicitados al comienzo del proyecto a raíz del contexto del grupo TDAQ descrito anteriormente:

- **Requerimiento:** Poder evaluar cambios propuestos para la red y los algoritmos antes de su instalación. **Objetivo:** Reaccionar de manera temprana ante riesgos.
- **Requerimiento:** Definir anticipadamente el mejor conjunto de pruebas a realizar sobre el sistema real durante el lapso de disponibilidad. **Objetivo:** Aprovechar la ventana de prueba enfocando el tiempo en contestar las preguntas más relevantes.
- **Requerimiento:** Flexibilidad para la elección del nivel de detalle/precisión con que se obtienen las evaluaciones. **Objetivo:** Poder adaptarse dinámicamente a distintas complejidades de las modificaciones a evaluar, y a cambios de cronograma.
- **Propuesta:** Se propone un proceso de ingeniería dirigida por modelado y simulación del sistema de TDAQ basado en el formalismo DEVS.
- **Beneficios:** 1) Contar con un simulador del sistema sobre el cual ensayar tempranamente -sin restricciones temporales- propuestas de innovaciones, y obtener conclusiones que guíen al diseño, implementación y pruebas de cambios sobre la infraestructura real cuando no esté disponible. 2) Construir una base de conocimiento no ambigua, reusable y organizada del sistema (un modelo) que unifique conceptos dentro del equipo interdisciplinario de científicos de TDAQ. 3) Poder decidir flexiblemente cual es el nivel de abstracción adecuado para modelar distintos casos de estudios sobre TDAQ, adaptándose a la complejidad de la pregunta, precisión requerida para la respuesta y al cronograma establecido.

Los requerimientos para los comportamientos que se desean reproducir mediante simulación son elicitados en reuniones de análisis con científicos expertos en cada área. Dichos requerimientos cambian a lo largo del proyecto y distintos especialistas pueden tener requerimientos diferentes sobre las mismas partes del modelo.

3.2 Metodología Propuesta

Para poner en práctica una estrategia de ingeniería dirigida por modelado y simulación se propuso una metodología de procesos (ciclos) iterativos como en la **Fig 3**.

Marco Formal DEVS[1]**.** En el corazón de la metodología, las entidades Sistema, Modelo y Simulador están estrictamente separadas y formalmente relacionadas por el Marco Formal DEVS. El Sistema (real) es ensayado dentro de un Marco Experimental del Sistema (ME_S): Se realizan preguntas para las cuales se definen Parámetros del Sistema Θ_S a variar, los cuales una vez aplicados en ensayos prácticos devuelven una Base de Datos de Comportamientos del Sistema λ_S.

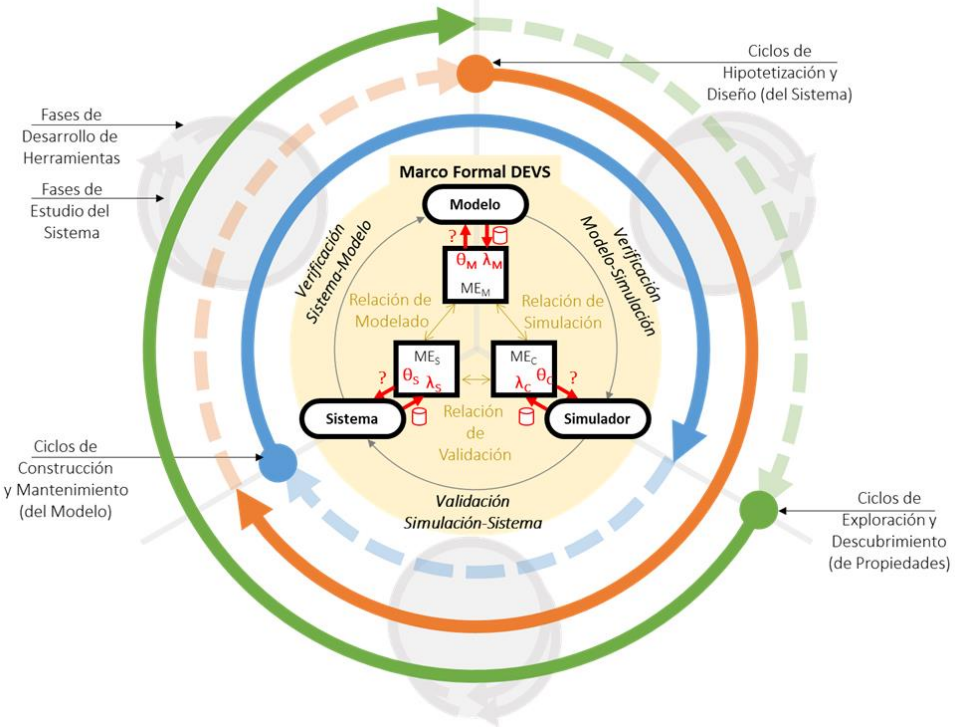

Fig. 3. Ingeniería dirigida por Modelado y Simulación. Metodología basada en el Marco Formal DEVS, ciclos iterativos y fases incrementales

Un Modelo DEVS (especificación abstracta de estructuras y comportamientos) se construye a partir del Sistema en su ME_S, según una *Relación de Modelado* guiada por homomorfismos e isomorfismos. El Marco Experimental del Modelo MEM permite hacer preguntas definiendo los parámetros Θ_M y obtener respuestas λ_M acerca de atributos abstractos del modelo, como por ejemplo densidad de acoplamiento, tipos de variables (discretas, continuas, etc.) independientemente de cualquier actividad de simulación. Un Simulador DEVS (implementación ejecutable de un modelo DEVS) se construye a partir de un Modelo DEVS en su ME_M, según una *Relación de Simulación* que implementa estrategias de cómputo suficientes para el Modelo en cuestión. El Marco Experimental de Cómputo ME_C define las preguntas y parámetros Θ_C para experimentar sobre el modelo computable (simulador) y obtener una Base de Comportamientos de simulación λ_C. La *Relación de Validación* permite regresar al Sistema

[1] Extensión original del Marco Formal DEVS tomando elementos de [1,4 y 5].

original para validar la correctitud de una simulación (λ_S vs. λ_C), o bien realizar exploraciones en el ME_S debido a observaciones novedosas en el ME_C.

Ciclos y Fases. Definimos 3 **Ciclos** principales: de *Construcción* (del Modelo) en azul, de *Hipotetización* (sobre el sistema) en rojo y de *Exploración* (sobre las Simulaciones) en verde. Si bien el objetivo de cada ciclo difiere, en todos los casos se itera sobre el Marco Formal DEVS en el sentido Sistema→Modelo→Simulación. A su vez, en cada evolución entre Marcos Experimentales, se distinguen 2 **Fases** que cooperan en paralelo: las fases de *Estudio del Problema* dirigen el avance acorde a las preguntas que se quieren contestar sobre el Sistema de partida, mientras que las fases de *Desarrollo de Herramienta* procuran mejorar el software de soporte (potenciando las capacidades de modelado, simulación y análisis en sintonía con las preguntas-guía del proyecto). El Ciclo de *Construcción* parte de la observación del Sistema y su objetivo es obtener Modelos de buena calidad, que una vez simulados tengan buen grado de validación con el Sistema. El Ciclo de *Hipotetización* parte de ejercitar sobre el Modelo cambios hipotéticos a realizar sobre el Sistema. Su objetivo es encontrar oportunidades de mejoras en el Sistema cuando este no está disponible o es muy costosa la experimentación directa. El Ciclo de *Exploración* parte de observar grandes volúmenes de información producidos por simulación y su objetivo es descubrir propiedades y correlaciones no imaginadas previamente en fases de experimentación.

Beneficios de la metodología propuesta. Utilizar un proceso iterativo permite tomar conocimiento paulatinamente, tanto del sistema y sus componentes como de la organización. En particular conocer la parte humana, los expertos y científicos a cargo de las diferentes áreas, resultó fundamental para comprender la problemática. Esto se ve reflejado también en el modelo desarrollado, ya que se añaden características al modelo iterativamente, comenzando con un alto nivel de abstracción y gradualmente incrementando el nivel de detalle, lo que permite comenzar a reproducir las partes más relevantes del sistema desde las primeras etapas, brindando respuestas en el corto-mediano plazo como fue planteado en los requerimientos y cubriendo también el requerimiento de flexibilidad para la elección del nivel de detalle. Al tomar conocimiento del sistema gradualmente, surgen en primera instancia las características más relevantes, añadiendo mayor complejidad sólo en casos necesarios. Esta técnica derivó en el desarrollo de un modelo que reproduce el comportamiento del sistema real con tiempos razonables de simulación, ya que dinámicas menos relevantes no forman parte de la simulación (e.g. capa física de red). Los ciclos de la metodología propuesta brindan un marco para las etapas del ciclo de vida de la herramienta de simulación.

Se presentará un caso particular de aplicación de esta metodología, donde se comenzó por ciclos de construcción, observando el sistema, plasmando los conocimientos en un modelo de simulación ejecutable y utilizando herramientas de visualización y análisis de datos para la comparación de resultados (secciones 4, 4.1 y 4.2). Para dar respuestas y previsiones se utilizaron ciclos de hipotetización (simulación de escenarios que no eran ejercitables en el sistema real), implementando luego esas mejoras en aplicaciones reales (sección 4.3) (el ciclo de exploración y sus resultados preliminares quedan fuera del alcance de este trabajo).

Durante la ejecución de cada ciclo, las fases permiten enfocar los esfuerzos tanto en la problemática específica de TDAQ como en el desarrollo de soluciones genéricas

para las herramientas teóricas y prácticas. En las Fases de Estudio del Problema es posible relevar el sistema real (sección 4), implementar un modelo que refleje su comportamiento en la simulación (secciones 4.1 y 4.2) y diseñar mejoras a los algoritmos de control de TDAQ (sección 4.3). El conocimiento obtenido durante estos períodos permite extender las técnicas utilizadas para mejorar las herramientas para su uso general en las tres áreas que se muestran en la metodología: en el área de modelado (por ejemplo, extendiendo VDEVS o desarrollando bloques reutilizables, sección 4.2), en el área de la simulación (por ejemplo, con la infraestructura para ejecutar simulaciones en diferentes nodos, sección 4.2) y desde el punto de vista de la validación de la simulación (extendiendo capacidades de visualización y cálculo de Scilab).

Metodologías actuales. Existen en la Ingeniería de Software procesos y metodologías que proponen marcos de trabajo para el control de ciclo de vida en proyectos de desarrollo. Entre los más utilizados se pueden mencionar RUP [22], XP [23], FDD [24], TDD [25], etc. Algunos proponen técnicas de programación como *peer programming* o *code reviews*, utilizados durante este trabajo. Otras (e.g. RUP) proponen ciclos iterativos e incrementales, con entregas frecuentes enfocadas en agregar valor. Nuestra metodología comparte ciertos aspectos con estos marcos. Sin embargo, ninguna de las metodologías mencionadas contempla aspectos formales de M&S provistos por DEVS: separación estricta entre formalismo de modelado, mecanismo abstracto de simulación, código de implementación de los modelos y código de implementación de los motores de simulación. Esto presenta la ventaja de poder manejar marcos experimentales específicos para el trabajo con el sistema real, el modelo, y el simulador. Además, en proyectos típicos de desarrollo de software no es usual modificar las herramientas de base utilizadas durante del proceso. Sin embargo, en proyectos científicos basados en M&S, las herramientas de modelado, simulación y visualización son dispositivos cruciales que demandan sus propios requerimientos a la par del modelo. Aún más, las mejoras incorporadas en las herramientas deben ser genéricas, reusables en proyectos similares. Nuestra metodología cubre naturalmente esta necesidad.

4 Caso de estudio: flujo y red de datos en TDAQ

En esta sección se resumen detalles técnicos de TDAQ utilizados para crear el modelo de simulación. Los detalles de implementación, arquitectura y análisis de software quedan fuera del alcance de este documento para hacer hincapié en las decisiones metodológicas y de modelado. El modelo se enfocará en el flujo de datos de la red HLT para la predicción de su performance, siendo la *latencia de filtrado* la principal métrica analizada (tiempo que transcurre desde que el HLTSV asigna un evento a una PU hasta que el evento es descartado o almacenado).

La **Fig 4** muestra un diagrama de secuencia de las aplicaciones que forman parte del filtrado de un evento. Las PU solicitan información al ROS en dos etapas denominadas Filtrado de Nivel 2 (L2) y Construcción de Evento (EB). Primero en L2 se solicita y analiza una porción del evento. Luego, en caso necesario, en EB se solicita el evento completo. Para cada evento, los nodos del ROS envían sus respuestas a una misma DCM casi al unísono, generando ráfagas en dirección ROS→DCM que por efectos de encolamiento en los core switch y ToRs incrementan la latencia de filtrado.

Para evitar pérdidas de paquetes por saturación de las colas, el DCM limita la cantidad de pedidos simultáneos hacia el ROS, permitiendo que en la red haya una canti-

dad limitada de fragmentos "en vuelo", tantos como "créditos" se hayan configurado. Como los fragmentos varían su tamaño, el control de tráfico no evita completamente el descarte de paquetes, por lo que es importante estudiar su efecto.

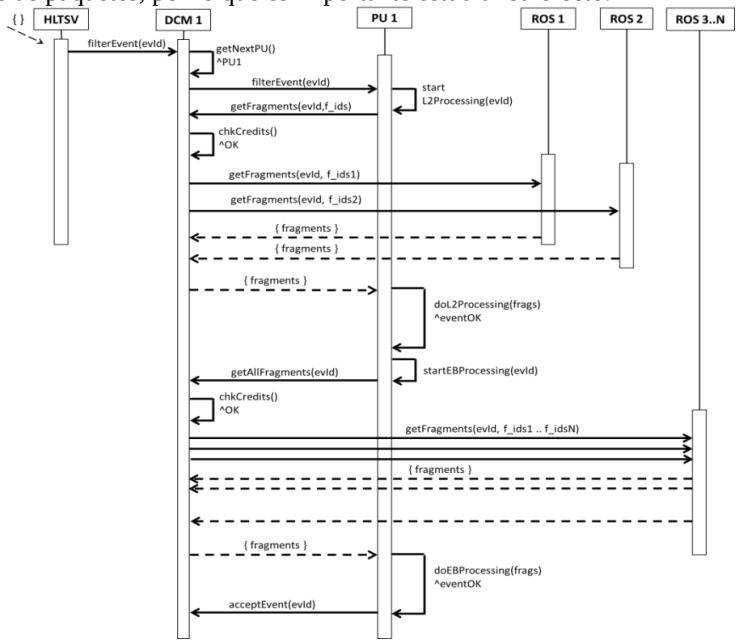

Fig. 4. Diagrama de secuencia de las aplicaciones TDAQ para el filtrado de un evento

La red de TDAQ utiliza TCP/IP para el envío de mensajes, posee un gran ancho de banda y una baja latencia en relación al mínimo tiempo en que TCP demora en retransmitir un paquete perdido: 200ms [27]. Se produce una problemática tipificada como efecto TCPIncast [26]. Esto tiene un alto impacto ya que si un único paquete TCP es descartado, una PU no puede comenzar a procesar el evento hasta la retransmisión, aumentando más de 100 veces la latencia de red (de 1.6ms a >200ms).

4.1 Primera iteración: Construcción del modelo

Basados en la metodología propuesta, comenzamos con un ciclo de construcción (ciclo azul) tomando como marco experimental ME_S un subconjunto del sistema completo: utilizando una única instancia de las aplicaciones DCM y PU.

Mediciones en el sistema real. La metodología inicia con la observación del sistema real (experimentación y toma de métricas)[2]. Luego, medimos latencia red en diferentes escenarios. Como ME_S se utilizó el ROS con todos sus nodos y una única máquina TPU (corriendo una única aplicación DCM) con un único proceso PU. Se experimentó definiendo Θ_S={# créditos iniciales en el DCM}. Los resultados (λ_S) se muestran en la **Fig 5 a)** Se observa que la configuración óptima (donde la latencia se mantiene en ~20ms) se obtiene utilizando entre 100 y 600 créditos. Con una cantidad menor de créditos la latencia aumenta ya que el DCM puede enviar menos solicitudes simultáneas. Utilizando más de 600 créditos, la latencia aumenta rápidamente y des-

[2] Asumimos que se tiene acceso (al menos esporádico) al sistema real para tomar mediciones. Cuando esto no se cumple, se modela en base a hipótesis de diseño y/o a leyes universales.

pués de 1800 créditos se estabiliza en 500ms. Los descartes de paquetes en el ToR aparecen coincidentemente con créditos mayores a 600, por lo que el rápido aumento de la latencia en esta zona se debe a la congestión en la red (descartes de paquetes y retransmisiones). No se observó pérdidas de paquetes en los core switch.

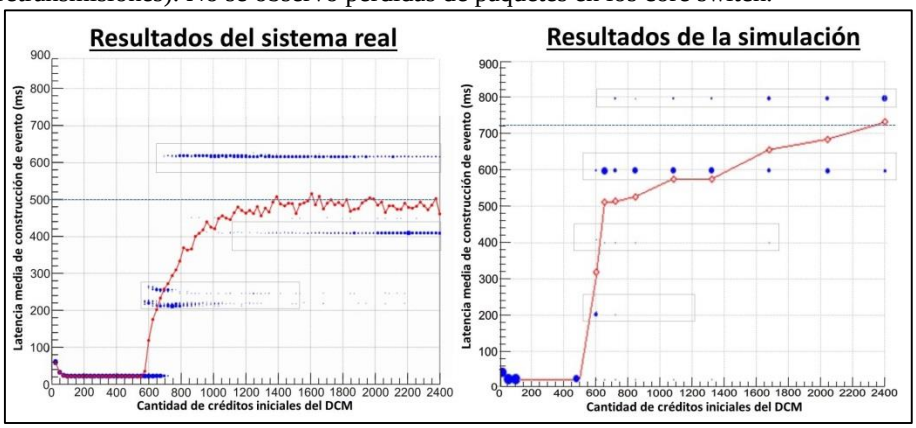

Fig. 5. Latencia vs créditos iniciales (1DCM, 1PU) **a)** Sistema Real **b)** Simulación. *Puntos rojos*: latencia promedio de los eventos filtrados. *Puntos azules*: latencias individuales (puntos más grandes denotan mayor cantidad de ocurrencias).

Implementación del modelo. La siguiente etapa del ciclo de construcción es la creación de un modelo con el comportamiento y observaciones de la sección anterior. La **Fig 6** muestra una vista del modelo de TDAQ implementado en PowerDEVS.

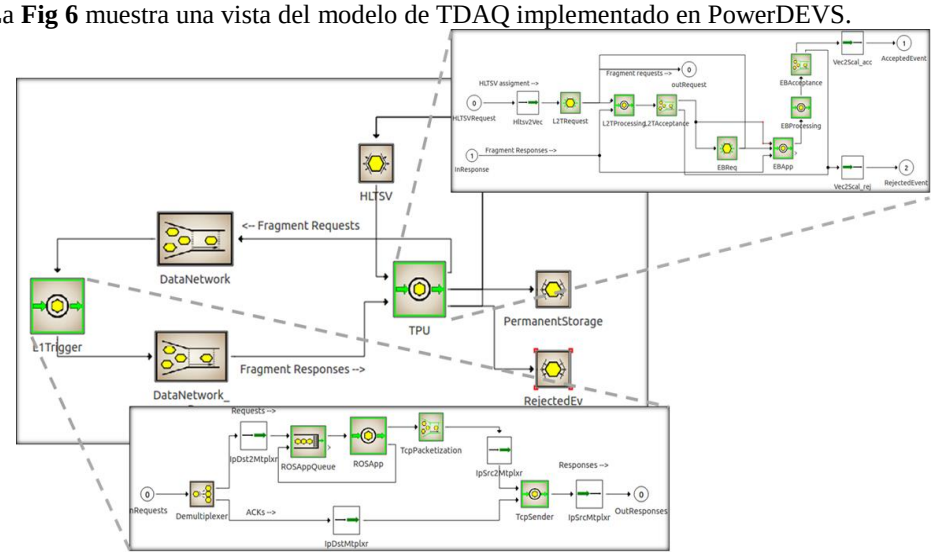

Fig. 6. Modelo de simulación para TDAQ implementado en PowerDEVS

Preservando la semántica del sistema real, se creó un modelo jerárquico con sus componentes principales respetando nombres, estructura y comportamiento. Para los modelos DCM, PU y HLTSV, se decidió extraer lógica de algoritmos de control en C++ directamente de las aplicaciones reales, maximizando el homomorfismo con el sistema estudiado, tarea facilitada por la preservación de la semántica. Los modelos

acoplados del ROS y DCM implementan la lógica de TCP basado en librerías pre-existentes de PowerDEVS [16]. TCPSender modela la máquina de estados de TCP Cubic [28] implementando solo el comportamiento de TCP relevante para el caso de estudio (e.g. Fast Retransmit no está modelado aún). Gracias a las pruebas realizadas sobre el modelo TCP y sus validaciones contra el sistema real se detectó un posible bug en la implementación de TCP en Linux SCL6 [29].

Como parte de las Fases de Desarrollo de Herramienta, los modelos atómicos de TCP (receptor y emisor) y de nodos de red se implementaron genéricos y reutilizables, incorporados a la biblioteca de PowerDEVS. Se crearon en Scilab y ROOT nuevos mecanismos de visualización para post-análisis de latencias, y una infraestructura para ejecutar simulaciones en nodos paralelos barriendo espacios de parámetros.

Mediciones y validación de la simulación. Siguiendo con la metodología, el siguiente paso fue la verificación del modelo y la validación de la simulación. Para ello se imitó la configuración de las pruebas con el sistema real descrita anteriormente. Se realizó en la simulación el mismo experimento que en el sistema real, aumentando la cantidad de créditos iniciales del DCM.

Los resultados pueden verse en la **Fig 5 b)**, donde la simulación aproxima de manera suficientemente satisfactoria el perfil de las latencias medias para el tipo de preguntas que se desean responder (aceptación de la simulación, $\lambda_S \sim \lambda_C$). Al igual que en el sistema real, utilizando entre 100 y 600 créditos la latencia es óptima y luego se observa un gran incremento causado por retransmisiones. Las latencias individuales (puntos en azul) se corresponden con las métricas reales, validando la dinámica de TCP (retransmisiones y efecto TCPIncast). También se validó la utilización de los enlaces y la ocupación promedio de las colas, observando en la simulación, detalles imposibles de medir en el sistema real (e.g. encolamiento de ráfagas en los ToR).

La simulación presenta diferencias respecto del sistema real. Por ejemplo, la latencia promedio en el sistema real se estabiliza en ~500ms mientras que en la simulación se estabiliza en ~700ms. Esto será estudiado en mayor profundidad en el futuro, ya que puede estar relacionado con el bug en TCP mencionado anteriormente [29].

4.2 Segunda iteración: Mejoras al modelo

En una segunda iteración del ciclo de construcción, se amplió el marco experimental MEs aumentando el número de instancias de DCMs y PUs. Asimismo, durante el ciclo, el sistema real sufrió modificaciones, disparando nuevos cambios en el modelo.

Actualizaciones en el sistema real - cambios en la topología de red. El grupo de TDAQ realizó diversos cambios en el HLT en preparación para el Run2, donde se duplicará la energía de las colisiones. En el ROS se quitaron los ToR switch y se reemplazaron los 200 nodos por 100 nuevas computadoras con 4 interfaces de 10Gb/s cada una conectadas directamente a ambos core switch. En cada ToR switch se agregó un enlace adicional de 10Gb/s conectado al otro core switch.

Mediciones en el sistema real. Siguiendo con la metodología, el primer paso en este ciclo es la toma de métricas del sistema real actualizado, con un número mayor de aplicaciones. En este caso, el tráfico de red está mayormente determinado por la frecuencia de asignaciones del HLTSV, por lo que se experimentó variando este parámetro (Θ_S={frecuencia HLTSV}). Se usaron los nuevos nodos del ROS y un rack completo de TPUs (40 DCMs, 960 PUs). Se configuró las PUs para solicitar eventos

completos el 50% de las veces. En el DCM se utilizaron 500 y 700 créditos iniciales.

La **Fig 7 a)** muestra la latencia promedio variando la frecuencia del HLTSV. Cuando el HLTSV asigna eventos a 50Hz, la latencia es mínima (~13ms) ya que la red está completamente libre en el momento de comenzar a filtrar cada evento. Al aumentar la frecuencia de asignación, la latencia comienza a subir ya que varias PUs operan realizando solicitudes al mismo tiempo, por lo tanto, deben compartir los recursos de la red y créditos del DCM. A partir de ~3.2kHz la latencia aumenta exponencialmente, ya que se alcanza la capacidad máxima de la red (93% de utilización).

Implementación del modelo. Como siguiente paso del ciclo de construcción, se realizó la implementación del modelo basado en las nuevas observaciones del sistema. En cuanto a las actualizaciones de la topología, los cambios en el modelo fueron menores: se quitaron los modelos de ToR del ROS y se actualizó la configuración para la nueva capacidad de los canales. Esto muestra la flexibilidad del modelo y la ventaja de tener una relación uno a uno entre los componentes del sistema real y del modelo. En esta etapa se desarrolló una versión completa del HLTSV reusando código C++ del HLTSV real, aportando mayor credibilidad. Para aumentar la cantidad de instancias de modelos, se utilizó VDEVS, creando 16 nuevos modelos vectoriales. Diez nuevos modelos de multiplexores representan el ruteo de los paquetes.

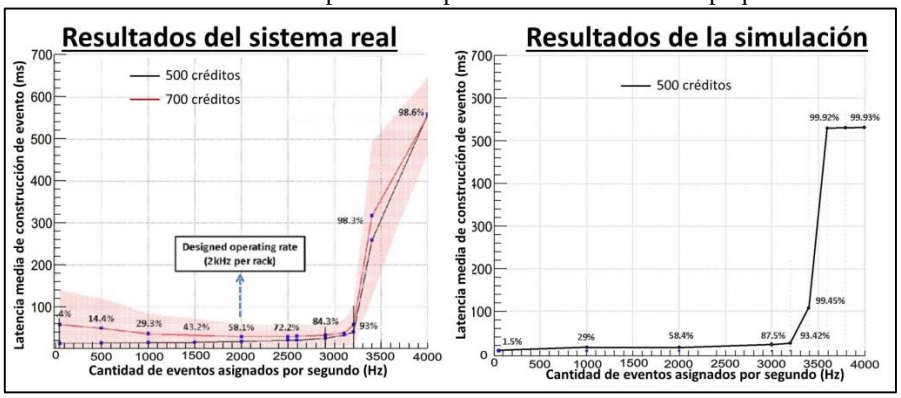

Fig. 7. Simulación barriendo el HLTSV rate (1 rack) a) Sistema Real b) Simulación

Como parte de las Fases de Desarrollo de Herramienta, se implementaron tres soluciones genéricas para atacar el problema de escalabilidad, necesarias en este proyecto para aumentar 50 veces la cantidad de instancias a simular. Se extendió la propuesta original de VDEVS permitiendo el uso de SmartPointers en mensajes vectoriales. Se modificó el motor de simulación de PowerDEVS con una infraestructura de SmartPointers que permite un manejo automático y transparente de memoria para cualquier modelo atómico, y se desarrolló un framework para lanzar simulaciones simultáneas en nodos distribuidos para barrido de parámetros, reduciendo considerablemente los tiempos de simulación. Estas nuevas herramientas quedan disponibles para ser utilizadas por otros modelos DEVS independientes de TDAQ.

Resultados de simulación y validación con el sistema real. Para finalizar con este ciclo de construcción, se validan las simulaciones contra el sistema real.

Se realizó en la simulación un barrido paramétrico de la frecuencia de asignaciones del HLTSV, con las mismas configuraciones que lo detallado para el sistema real al

principio de esta sección. En este experimento se simularon 60s por cada punto y se utilizaron tres nodos diferentes para la ejecución. Como se puede observar en Fig 7 b), los resultados de la simulación reproducen una curva de latencia similar a la curva medida en el sistema real. Los valores absolutos de latencias y carga de la red en la simulación difieren de los reales dentro de un margen aceptable (<5% de diferencia), mostrando un buen grado de validación entre simulación y sistema real.

4.3 Tercera iteración: Propuesta de mejora del sistema real

El alto grado de similitud entre los resultados de simulación y de experimentación sobre el sistema real (sección anterior) brindan un nivel de confianza suficiente para pasar a la siguiente etapa de la metodología: un ciclo de hipotetización. Gracias al conocimiento adquirido sobre el TDAQ al modelar su dinámica, fue posible plantear propuestas alternativas para mejorar su rendimiento, como se describe debajo.

Hipotetización en el modelo: propuesta de mejora del HLTSV. El HLTSV implementa un algoritmo FIFO para la selección del nodo que procesará el próximo evento. Al cabo de unos segundos de ejecución, el orden de asignación se vuelve aleatorio ya que cada PU termina el filtrado en tiempos diferentes. El incremento lineal de la latencia que se observa en la **Fig 7** se debe a que varias PUs compiten por los mismos recursos (créditos del DCM y enlaces de la red). Sin embargo, en las condiciones de este experimento y a la frecuencia por diseño (2kHz por rack), cada DCM recibe en promedio una asignación cada 50-60ms y la latencia mínima (en óptimas condiciones) para el filtrado de un evento es de ~13ms. Luego, no es necesario que dos PUs procesen simultáneamente en un mismo DCM. Sin embargo, como las asignaciones son aleatorias, se dan casos en que 4 o 5 PUs de un mismo DCM procesan en paralelo.

Se modelaron tres políticas de asignación del HLTSV: 1) FIFO, utilizado por el sistema real 2) RANDOM, el HLTSV selecciona una PU desocupada al azar 3) LEAST_BUSY_DCM, el HLTSV selecciona una PU libre dentro del DCM que posea menor cantidad de PUs filtrando eventos.

Resultados de la simulación. Una vez implementados los cambios en el modelo, se realizaron simulaciones para comparar el algoritmo RANDOM con la nueva propuesta LEAST_BUSY_DCM (se omite FIFO ya que al cabo de un tiempo es equivalente a RANDOM). Para la comparación se realizó el mismo experimento que en la sección anterior, incrementando la frecuencia del HLTSV, pero simulando 9 racks de TPUs (con un total de 267 DCMs y con 6408 PUs). La **Fig 9 b)** muestra los resultados de la simulación comparando ambos algoritmos. El algoritmo RANDOM muestra el mismo comportamiento que el descrito en la sección anterior donde se utilizaba un algoritmo FIFO. En cuanto al nuevo algoritmo, la simulación muestra que la latencia promedio se mantiene constante y con valor mínimo (~16ms) para frecuencias menores a 24kHz. Para frecuencias mayores la latencia crece exponencialmente debido a la saturación de la red. La simulación muestra que, para esta configuración, el nuevo algoritmo podría reducir la latencia entre dos y cuatro veces.

Implementación en el sistema real y Validación. El siguiente paso fue la implementación y puesta a prueba del nuevo algoritmo de asignaciones en el HLTSV real, siguiendo la lógica desarrollada en la simulación. Es necesario que el HLTSV funcione estrictamente a 100kHz, por lo que se reestructuró convenientemente el algoritmo y sus estructuras de datos. Se realizó el mismo experimento en el sistema real: barrido

de la frecuencia del HLTSV utilizando 9 racks de TPUs. La **Fig 9 a)** muestra los resultados de la comparación del algoritmo RANDOM y LEAST_BUSY_DCM. Con el nuevo algoritmo, la latencia promedio en el sistema real se mantiene mínima para frecuencias menores a 24kHz y también muestra mejoras de dos a cuatro veces comparadas con el algoritmo actual, como se había predicho en la simulación.

Los resultados experimentales son muy similares a las predicciones de la simulación, probando que el modelo de simulación es capaz de reproducir comportamientos conocidos y también de predecir impactos ante cambios al sistema.

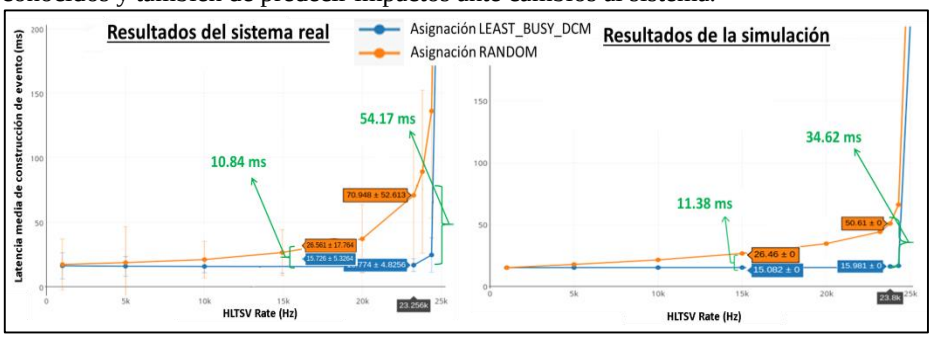

Fig. 8. Comparación de algoritmos de asignación a) Sistema Real b) Simulación

5 Conclusiones y Trabajo Futuro

Presentamos una metodología iterativa e incremental para el desarrollo de proyectos basados en M&S formal, junto a su aplicación práctica en un caso de estudio de ingeniería de redes en el contexto de un experimento científico a gran escala: el sistema TDAQ del experimento ATLAS. Nuestra metodología permitió cubrir los requerimientos planteados y obtener las mejoras requeridas mediante Ciclos de Construcción, obteniendo entregables de calidad incremental en tiempos cortos. Obtuvimos un modelo de simulación que reproduce el comportamiento del sistema real en diferentes condiciones. La herramienta de simulación implementada probó ser flexible ante cambios en el sistema real. Mediante Ciclos de Hipotetización, fue posible utilizar M&S para diseñar y evaluar mejoras en los algoritmos de control de TDAQ antes de su puesta en producción, donde el modelo de simulación pudo dar predicciones acertadas. Utilizando Fases de Desarrollo de Herramienta se desarrollaron mejoras al simulador de base (PowerDEVS): se implementó una infraestructura para desacoplar el manejo de memoria de la lógica en los modelos atómicos y un nuevo framework para simulación en nodos paralelos. La metodología produjo un caso de éxito en tiempos cortos, de manera ordenada, y en un contexto exigente donde los usuarios de los modelos (principalmente científicos) requieren informes minuciosos para justificar cada próximo paso. Como trabajo futuro, se continuará añadiendo nuevos escenarios de TDAQ y evaluando diferentes técnicas de control de tráfico. Desde el punto de vista de las herramientas, se integrará PowerDEVS con software de análisis de grandes datos, comenzando así con los Ciclos de Exploración de Propiedades del Sistema.

6 Referencias

1. Zeigler, B.P.: Theory of Modeling and Simulation. John Wiley & Sons, New York, 1976
2. Chow, Alex Chung Hen, and Bernard P. Zeigler: "Parallel DEVS: a parallel, hierarchical, modular, modeling formalism." Proceedings of the 26th conference on Winter simulation. Society for Com-

puter Simulation International, 1994.
3. Zeigler, B.P., Praehofer H., and Kim T.G.: Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems. Academic press, 2000.
4. Wainer, G. Discrete-event modeling and simulation: a practitioner's approach. CRC Press, 2009.
5. Wainer, G. and Mosterman J., eds. Discrete-event modeling and simulation: theory and applications. CRC Press, 2010.
6. ATLAS Collaboration: "The ATLAS experiment at the CERN large hadron collider." J. Instrum 3 (2008): S08003.
7. Pestre, Dominique: "L'organisation européenne pour la recherche nucléaire (CERN): Un succès politique et scientifique." Vingtieme siecle. Revue d'histoire (1984): 65-76.
8. Evans, Lyndon, and Philip Bryant: "LHC machine." Journal of Instrumentation 3.08 (2008): S08001.
9. CMS collaboration: "The cms experiment at the cern lhc. JINST 3." S08004 (2008): 1748-0221.
10. ALICE Collaboration, and K. Aamodt. "The ALICE experiment at the CERN LHC. JINST, 3 (S08002), 2008.
11. LHCb Collaboration: "The LHCb detector at the LHC, 2008." Jinst 3: S08005.
12. Wainer, G. and Castro R. A Survey on the Application of the Cell-DEVS Formalism. J. Cellular Automata 5(6): 509-524 (2010)
13. Alex Chung Hen Chow and Bernard Zeigler. Parallel DEVS: a parallel, hierarchical, modular, modeling formalism. Winter simulation, 1994.
14. Bergero, F., and Kofman E. A vectorial DEVS extension for large scale system modeling and parallel simulation. Simulation 90.5 (2014): 522-546.
15. Bergero, F., and Kofman, E. (2011). PowerDEVS: a tool for hybrid system modeling and real-time simulation. Simulation, 87(1-2), 113-132.
16. Castro, R. Herramientas Integradoras para Modelado, Simulación y Control de Redes de Datos, Capítulo 7, p135. Tesis Doctoral, FCEIyA, Universidad Nacional de Rosario, Argentina, 2010
17. OMNeT++ Community (2004). OMNeT++ Discrete Event Simulation System. www.omnetpp.org.
18. Issariyakul, T. and Hossain, E. (2008). Introduction to Network Simulator NS2. Springer.
19. Chang, Xinjie. "Network simulations with OPNET." Proceedings of the 31st conference on Winter simulation: Simulation---a bridge to the future-Volume 1. ACM, 1999.
20. Cellier F. and Kofman E. Continuous System Simulation. Springer, New York, 2006.
21. Castro, R. and Kofman E. An integrative approach for hybrid modeling, simulation and control of data networks based on the DEVS formalism. In Obaidat et al (Eds.) Modeling and Simulation of Computer Networks and Systems, 1st Ed. Elsevier, 2015.
22. Kruchten, P. The rational unified process: an introduction. Addison-Wesley Professional, 2004.
23. Beck, K. Extreme programming explained: embrace change. Addison-Wesley Professional, 2000.
24. Palmer, S.R., and Felsing M. A practical guide to feature-driven development. Pearson Education, 2001.
25. Astels, Dave. Test driven development: A practical guide. Prentice Hall, 2003.
26. Kulkarni, Santosh, and Prathima Agrawal. Analysis of TCP Performance in Data Center Networks. Springer, 2014
27. Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981
28. Ha, Sangtae, Injong Rhee, and Lisong Xu. "CUBIC: a new TCP-friendly high-speed TCP variant." ACM SIGOPS Operating Systems Review 42.5 (2008): 64-74
29. Red Hat Bugzilla: “TCP bug creates additional RTO in very specific condition”. https://bugzilla.redhat.com/show_bug.cgi?id=1203742. Reported:2015-03-19