**Modeling and simulation-driven engineering based on discrete-event systems specifications  
  
Methodology and case study for the   
Data Acquisition Network of the ATLAS Experiment**

Matías Bonaventura, Daniel Foguelman and Rodrigo Castro  
Departamento de Computación, Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires, Argentina  
{mbonaventura, dfoguelman, rcastro}@dc.uba.ar

We present an iterative and incremental development methodology for simulation models in network engineering projects. Driven by the DEVS (Discrete Event Systems Specification) formal framework for modeling and simulation we assist network design, test, analysis and optimization processes.

A practical application of the methodology is presented for a case study in the ATLAS particle physics detector, the largest scientific experiment built by man where scientists around the globe search for answers about the origins of the universe. The ATLAS data network conveys real-time information produced by physics detectors as beams of particles collide. The produced sub-atomic evidences must be filtered and recorded for further offline scrutiny.

Due to the criticality of the transported data, networks and applications undergo careful engineering processes with stringent quality of service requirements. A tight project schedule imposes time pressure on design decisions, while rapid technology evolution widens the palette of network design options. Finally, due to the large scale of the project, networks and systems are available for tuning and testing only sporadically.

By adopting the DEVS M&S formal framework in combination with software engineering best practices we develop network simulation models side by side with enhanced modeling capabilities and boosted simulation performance for our tools in a robust yet flexible way.

We thus maximize the team’s capabilities to hypothesize and exercise candidate network design options while the real system is not available, narrowing the space of tests worth running on the real system during the scarce windows of opportunity.

# Introduction

Robust software engineering methodologies offering product life-cycle control has proved to be cornerstone in modern software development projects. Simultaneously, various techniques of modeling and simulation (M&S) have become increasingly adopted in engineering processes to design complex systems, particularly in scenarios where it is difficult or impossible to predict system behavior as changes are introduced. However, there exist comparatively much less experience in hardware and software development methodologies driven by formal M&S.

DEVS is the most general formalism for modeling discrete event systems [1, 2, 3] and has been adopted in several disciplines for complex software/hardware systems design and analysis [4, 5]. In addition to providing an unambiguous formalism to define behavior and structure for models, DEVS provides a clear framework for system analysis, experimental framework definition, model-to-simulator verification, and model-to-system validation. Although this approach focuses on the formal correctness of an M&S process, it does not contemplate characteristics specific of software development projects such as elicitation of requirements, communication of results, planning, flexibility of the product, etc.

We present a DEVS-based methodology for M&S-driven engineering projects that integrate software development best practices tailored to the case of a large-scale physics experiment (ATLAS particle detector [6] at CERN [7])

This project poses M&S challenges from several points of view, e.g. complexity of the system to be modeled, tight delivery times, quality and flexibility of the developed models and tools, interdisciplinary communication of results to collaborators (mostly scientists), big data-scale type of data analysis, etc.

By adopting an iterative methodology, requirements and simulation results are systematically validated. By means of an incremental process, knowledge about the real network is gradually acquired, adding relevant features progressively and generating simulation models that replicate real system features since the early stages.

The definition of three types of iterative cycles allows to frame and distribute the efforts across stages in the M&S project, which differ from the phases of a typical software project. The use of phases within each cycle allows to develop and execute models while refining the base M&S tools. Thanks to the DEVS formalism, the models, modeling interfaces and simulation algorithms are kept strictly separated yet formally coupled, thus permitting to decide flexibly on what kind of feature is to be enhanced and when according to given time and resources constraints.

We present progresses made ​​in building simulation models that answer questions about the data network under study along with improvements implemented in the M&S tools, both driven by the demands of the ATLAS project.

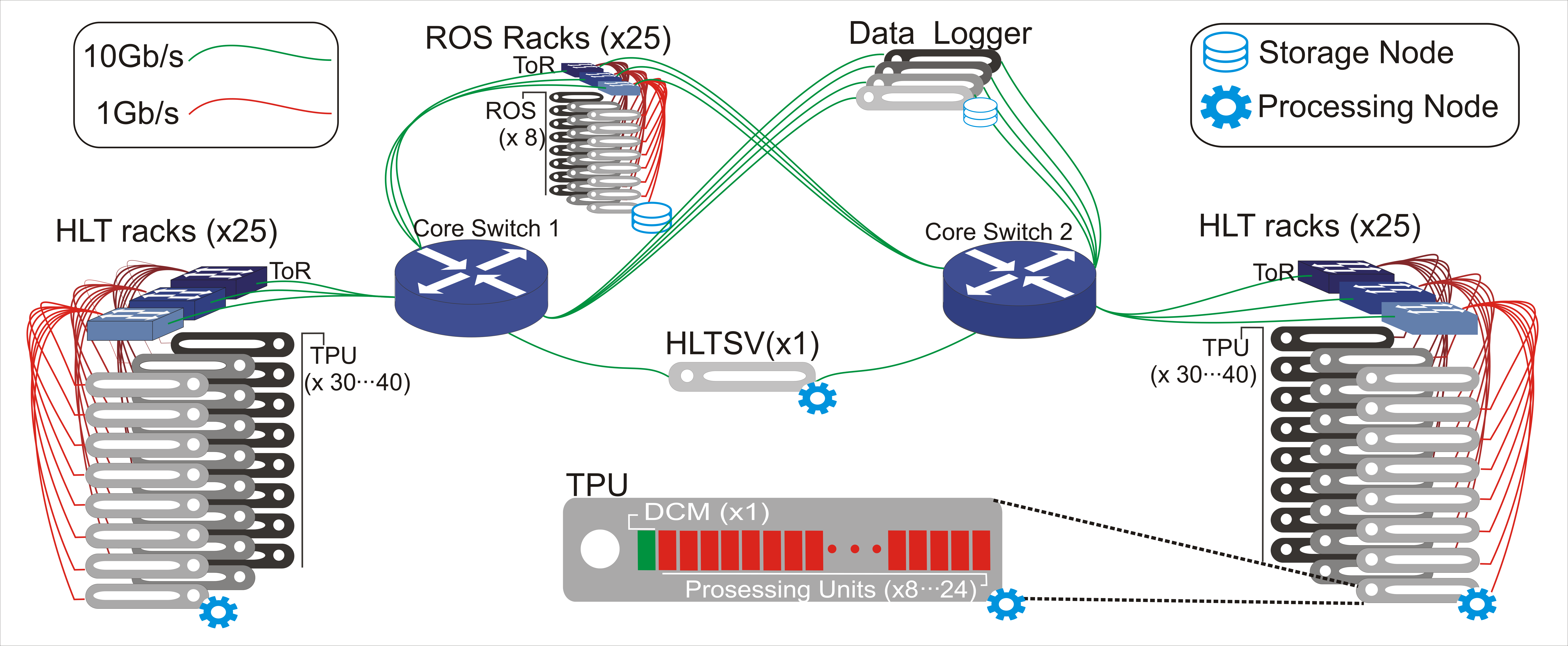
Two model build cycles will be described that reliably reproduce the behavior of the real system. Methodological and modeling decisions are discussed and a comparison between the simulation results and real system measurements are presented. We will show an explore cycle over the simulated data that led to the discovery of a suboptimal load balancing algorithm. We conclude with a hypothesis cycle: an improved load balancing scheme was designed and verified in the M&S domain, and later implemented and tested in the real network validating the correctness of the expected performance enhancements. The new algorithm is then set ready to be deployed in the production system.

# Preliminary concepts

## System under study: The Data Acquisition Network at CERN's ATLAS Experiment

The Large Hadron Collider (LHC [8]) is the world's largest particle accelerator with 27 kilometers of circumference and four large particle detectors: ATLAS [6], CMS [9] ALICE [10] and LHCb [11]. In 2013 the Run1 detectors went off for maintenance and upgrades (Long Shutdown 1, LS1) until the Run2 restart scheduled for 2015. ATLAS is a general purpose particle detector where proton-proton collisions generate very high energy enabling for the search of novel physical evidences (Higgs boson, extra dimensions, dark matter, etc.) Each protons collision is called an Event, where new smaller particles appear whose trajectories are digitized by detectors for further analysis. The network throughput exceeds 60 Terabytes/second. To assimilate all this information, ATLAS uses a sophisticated layered filtering system (Trigger and Data Acquisition, TDAQ) that decides in real time the relevancy of each data subset (what should be preserved and what can be safely discarded) according to physics algorithms. The First Level Trigger (L1) filters events from an initial raw rate of 40 million Events/second down to a filtered rate of 100 thousand Events/second. L1-accepted events are temporarily stored in a ReadOut System (ROS computer farm) in the form of structures called Fragments, and then accessed by a second level filter called the High Level Trigger (HLT). At the HLT physics algorithms reanalyze the fragments (this time around with a different granularity) retaining only 100 “interesting” Events/second. The TDAQ system and its HLT-ROS data network is our real *System Under Study,* for which we will describe modeling and simulation techniques and processes.

**Applications and Data Network in the High Level Trigger.** Figure 1 shows the interconnections between various applications that forms the HLT. After the L1 stores a new event in the ROS, the server notifies the High Level Trigger Supervisor (HLTSV) that the event is ready to be processed. The HLTSV assigns events to Trigger Processing Units servers (TPU). Each TPU runs an application called Data Collection Manager (DCM) that centralizes the communication between the TPU and the rest of the system (ROS, HLTSV, and DataLoggers). The DCM creates instances of the application Processing Unit (PU) - one per available core, between 8 and 24. Each PU finally runs the algorithm that determines whether the event is physically relevant and should be permanently stored or discarded in case of uninteresting events. The PU to event ratio is 1 to 1 (event processing is not parallelized).

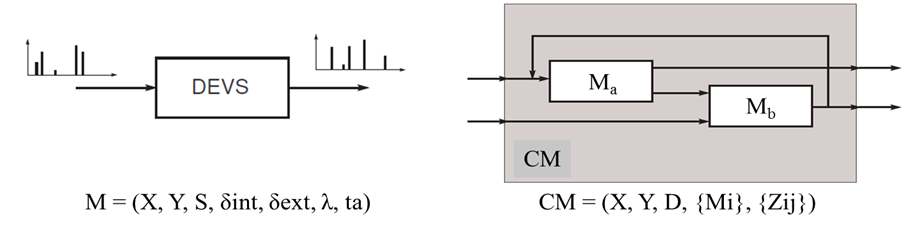


**Fig. 1.** Topology and applications in the HLT TDAQ farm

Applications run over an Ethernet Local Area Network with link capacities of 1 and 10 Gbps. Two core routers and ~100 switches interconnect ~3000 multicore servers using the TCP/IP protocols. **Fig. 1** shows a diagram of the network. The farm is composed by ~50 racks for TPU servers and ~25 racks for ROS nodes. Each TPU rack contains ~40 servers (DCMs and PU applications) and each ROS rack contains 8 servers. Within each rack, servers are connected to a shared Top of Rack Switch (ToR) using 1Gbps links. The HLTSV node and the ToRs are connected to the core switches over 10Gbps links.

## DEVS for data networks modeling

DEVS [1,4] is a mathematical formalism for modeling and simulation based on general systems theory, i.e. it is independent of any specific application. DEVS allows describing exactly any discrete system, and to approximate numerically continuous systems with any degree of desired accuracy. The formal specification of models provides tools for its analytical manipulation and offers independence in choosing the programming language for implementation [2]. DEVS models are described as a hierarchical composition of Atomic Models (M) and Coupled Models (CM) defined by mathematical tuples as shown in **Fig. 2**



**FIG. 2.** Basic DEVS atomic models (left) and coupled models (right)

 Coupled models define the structure of the system (interconnections between coupled and atomic models). Atomic models define the dynamic behaviors. For M, each possible model state s ∈ S has an associated lifetime defined by the function ta:S → R0+. When the model is in the state s = s1, at time t1=ta(s1) units of time the system performs an internal transition evolving towards a new state s2=δint(s1). δint:S→S is called internal transition function. In this case, at the same time an output event is produced with value y1=λ(s1). The function λ:S→Y is called the output function.

When a model receives an input event x1 ∈X, a transition is triggered that instantly changes the model state to state s4=δext (s3,e,x1), where s3 is the model state at the time that receives the input event, and e is the elapsed time since the last transition time (with e < ta (s3)). The function δext:SⅹR0+ⅹX → S is called external transition function.

The DEVS simulation algorithm for atomic models M and for coupling models CM is universal, unambiguous, easy to implement, and independent of programming languages.

**Vectorial DEVS.** There are many extensions and specializations of DEVS that tackle different needs (e.g. Cell-DEVS [12] for cellular automata, PDEVS [13] for modeling parallelism, etc.). In particular we are interested in Vectorial DEVS (VDEVS) [14] that allows representing large-scale systems with a compact graphical representation. A vectorial model is an array of quasi identical classic DEVS which may differ in their initial parameters. Formally the structure of a vector model is defined by: VD = {N, XV, YV, P, {M i}}, where N is the vector dimension, Xv is the set of input events vector, Yv is Vector set of output events, P is the set of parameters and each Mi is a classic DEVS atomic model. For the interaction between vectorial and non-vectorial, scalar to/from vector models are defined.

**Modeling and simulation tool: PowerDEVS.** The TDAQ model has been developed with the PowerDEVS [15] tool, which provides a graphical interface to define DEVS models using block diagrams and a C ++ editor to code the four dynamic functions for the M tuple. PowerDEVS includes libraries with reusable models. We adopt a data networks library (queues, servers, traffic generators, an implementation of TCP, etc. [16]) and extended it for our case study. PowerDEVS has a native interface to Scilab [15], an open source alternative to Matlab for numerical computation purposes.

As for established simulators specific for data networks we can mention OMNeT ++ [17], NS2/3 [18] and OPNET [19]. They model in great detail real network nodes and protocols, but these were not viable options for TDAQ. Due to the large size of the system, a too detailed simulation model becomes unfeasible to run in reasonable time. The TDAQ requirement is to be able to select a sufficient level of abstraction to answer each particular research question. Additionally, another requirement involves performing hybrid simulations (discrete events mixed with continuous flows). This capability is readily available in DEVS (QSS numerical integration methods [20]) already implemented in advanced implementations of PowerDEVS for data networks [21].

# Context, Requirements and Methodology for TDAQ M&S

For any case study that may arise in TDAQ, there are crosscutting contexts and requirements that call for a flexible yet robust development methodology. The following is a brief description of the scenario that motivated the proposal of our M&S-based methodology.

* **Context:** The TDAQ HLT filtering farm is a constantly evolving system. It is subject to recurring changes while hardware and control algorithms are regularly improved. **Impact:** is difficult or impossible to predict the impact of these changes before deploying them into production. Servers and network components are not installed until they are purchased, thus making it impossible to know in advance the impacts on the applications and the data flow.
* **Context:** The HLT farm is used by hundreds of scientists. TDAQ is thus available for full testing only about one out of every six weeks (during a full ATLAS stop). **Impact:** This delays the testing of new control algorithms that are continuously improved but cannot be fully validated until the system is available.

Listed below are the resulting elicited requirements:

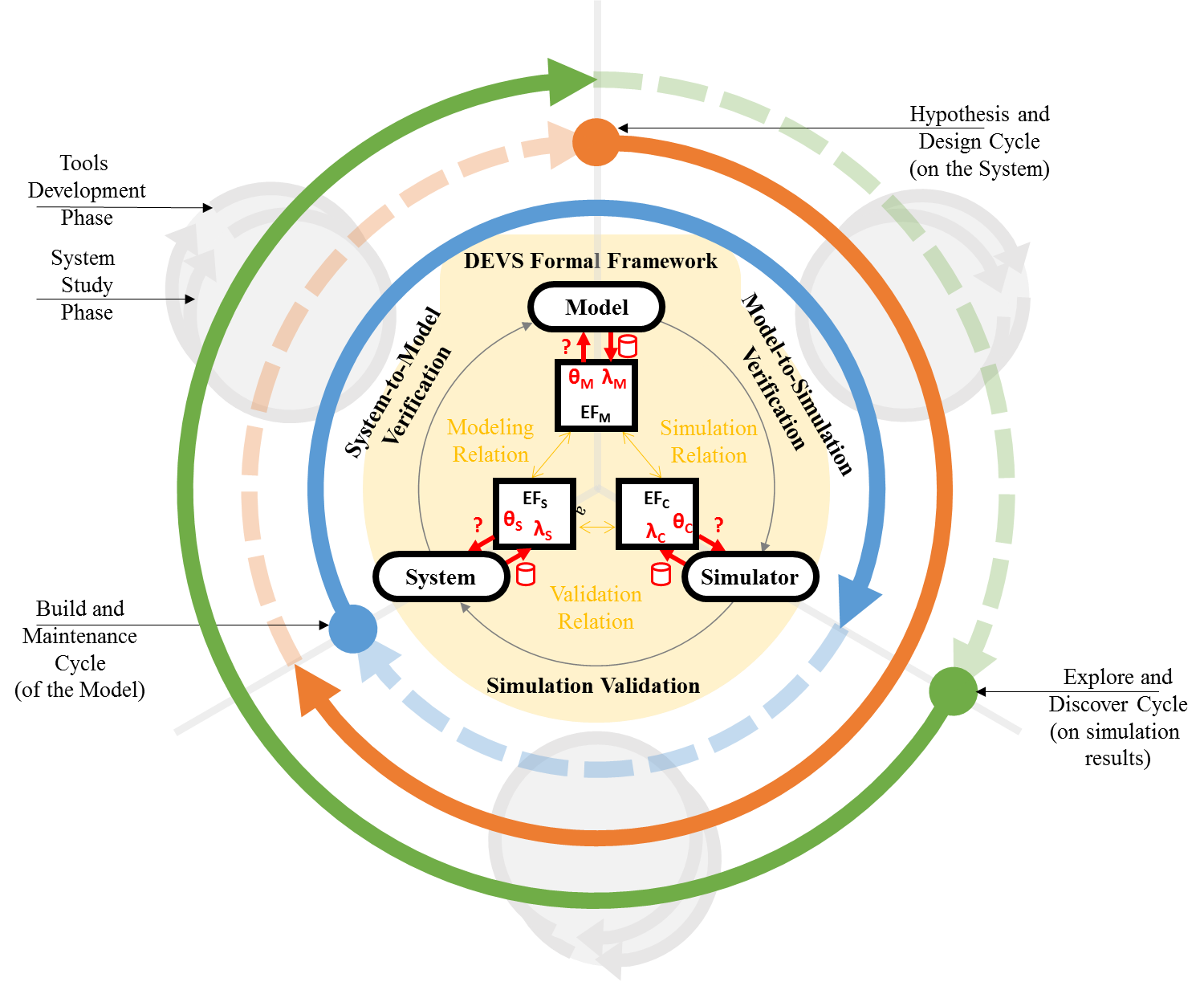
|  |  |
| --- | --- |
| **Requirement** | **Goal** |
| Evaluate candidate changes for the network and control algorithms before their commissioning. | Early risk assessment. |
| Define in advance the best set of tests to be performed on the real system, during scarce windows of availability. | Harnessing the test window to focus on the most relevant questions. |
| Flexibility for choosing the level of detail / accuracy with which the evaluations are obtained. | Dynamically adapt to different and complex modifications that need to be assessed, and to schedule changes**.** |

Requirements for each case study are elicited during system analysis meetings. Moreover, elicited requirements are likely to change dynamically throughout a project, with different experts having varying requirements on a same component of the system.

The proposal to cope with the aforementioned demands is to adopt an M&S-based engineering process for networked systems design, relying on the DEVS formalism.

## Proposed Methodology

To implement an engineering strategy driven by modeling and simulation, an iterative process-based methodology is proposed (see **Fig 3**.)



**Fig. 3.** Modeling and Simulation-Driven Engineering. Methodology diagram based on the DEVS formal framework. Iterative *cycles* and incremental *phases*.

**DEVS Formal Framework[[1]](#footnote-1)**. At the core of the methodology, the entities *System*, *Model* and *Simulator* are strictly separated yet formally related by the DEVS formal framework. The real (or “source”) system is experimented under a *System Experimental Frame* (EFS). Questions are encoded in the form *System Parameters* ΘS that define experimental conditions. After experiments are performed, results relevant to the original questions are stored into a *System Behavior Database* λS .

Every new DEVS *Model*, as a specification of structures and behaviors, is built for a pair {System, EFS} according to a *Modeling Relation* and guided by selected homomorphisms/isomorphisms. A new *Model Experimental Framework* (EFM) is also obtained that allows performing questions about model’s attributes (using *Model Parameters* ΘM for queries, and a *Model Database* λM for storing answers) Such questions may enquiry about coupling density, model topology, types of variables (discrete, continuous), etc., have no access to the real system, and are independent of any simulation exercise. A DEVS simulator is an algorithm capable of reading a DEVS model and producing an output trajectory by obeying the model’s dynamics. Its most common realization is a computer program, usually referred to simply as *Simulator*, which is constructed/adapted/maintained to read and compute DEVS models efficiently within their MEM. This establishes a *Simulation Relation*. The *Compute Experimental Framework* (EFC) defines new questions and new *Compute Parameters* ΘC for experimenting with (simulate) the computable model. It also hosts the simulation results in a *Compute Behavior Database* λC. The *Validation Relationship* allows to *relate* back to the original system to validate the correctness of a simulation (λS vs. λC) or to perform scans over EFS due to unexpected observations discovered in the EFC.

**Cycles and Phases.** We define 3 main **cycles:** *Build* (the model) in blue, *Hypothesis* (on the system) in red, and *Explore* (simulation results) in green. While the goal of each cycle differs, in all cases the flow across the DEVS formal framework follows the System→Model→Simulation path. In turn, for each evolution through the cycle two parallel, cooperative **phases** are defined. The *System Study* phase drives progress according to questions about the System under study. The *Tools Development* phase seeks to improve the supporting software algorithms and interfaces, leveraging the modeling, simulation and analysis capabilities.

The *Build Cycle* starts with the observation and measurement of the System. Its objective is to provide quality models that, once simulated, will exhibit an adequate degree of validation against the original System. The *Hypothesis Cycle* exercises on the Model several candidate changes to be applied ​​onto the system. Its goal is to find improvement opportunities for the system when it is unavailable or direct experimentation is too expensive. The *Explore Cycle* starts with analyzing the large amounts of information produced by simulations, and its goal is to discover properties and correlations unthought-of during the experimentation phases.

Cycles need not occur in any specified order (although a *Build Cycle* is usually required at the begging of the project) as it will be shown in the case study.

This approach led to the building of a model that reproduces relevant behaviors of the real system within reasonable simulation times, since less relevant dynamics are kept out of the model (e.g. network physical layer). The methodology also offers a guideline for the development phases of the underlying modeling and simulation software tools; new features are added to the tools at specific phases, responding to specific needs, framed within unambiguous cycle goals.

**Other existing techniques and methods.** Software Engineering processes and methodologies propose frameworks to control development projects’ lifecycles. Among the most commonly used techniques RUP [22], XP [23], FDD [24], and TDD [25] can be mentioned. Some propose programming techniques such as pair programming or code reviews, also used for this work. Others (e.g. RUP) propose iterative and incremental cycles, with frequent deliveries focused on adding value quickly. Our methodology shares some aspects with these approaches.   
However, none of the aforementioned methods include formal M&S aspects provided by DEVS: strict separation between modeling formalism, abstract simulation mechanism, and code implementation (of both models’ behavior and simulation engines). This bears the advantage of independency between experimental frameworks for the real system, the model and the simulator. This way, enhancements in any of these three areas are straightforwardly propagated to any of the others. Besides, in typical software-based projects it is not usual to modify the base tools themselves, used for the execution of the project. Nevertheless, in M&S-driven scientific projects the base tools for modeling, simulation and data analysis are crucial devices that call for their own requirements alongside the requirements of the model itself. Our methodology naturally fills this need.

Large sets of simulation results can support data-driven hypothesis and predictive analytics [32]. A well-structured simulation database together with reusable data analysis libraries can systematize different layers of data aggregation to enable stratified levels of analyses. This mimics the four layers of data (from tier-0 to tier-3) used for storing the LHC physics information, both recorded from real collisions and also simulated physics.

# Case Study: Improving the TDAQ flow and data network

We now describe a real-life case study where the methodology was applied.

It starts with two *Build Cycles*, observing the system, translating knowledge into an executable simulation model and upgrading the model to represent important design changes in the system (4.1 and 4.2)

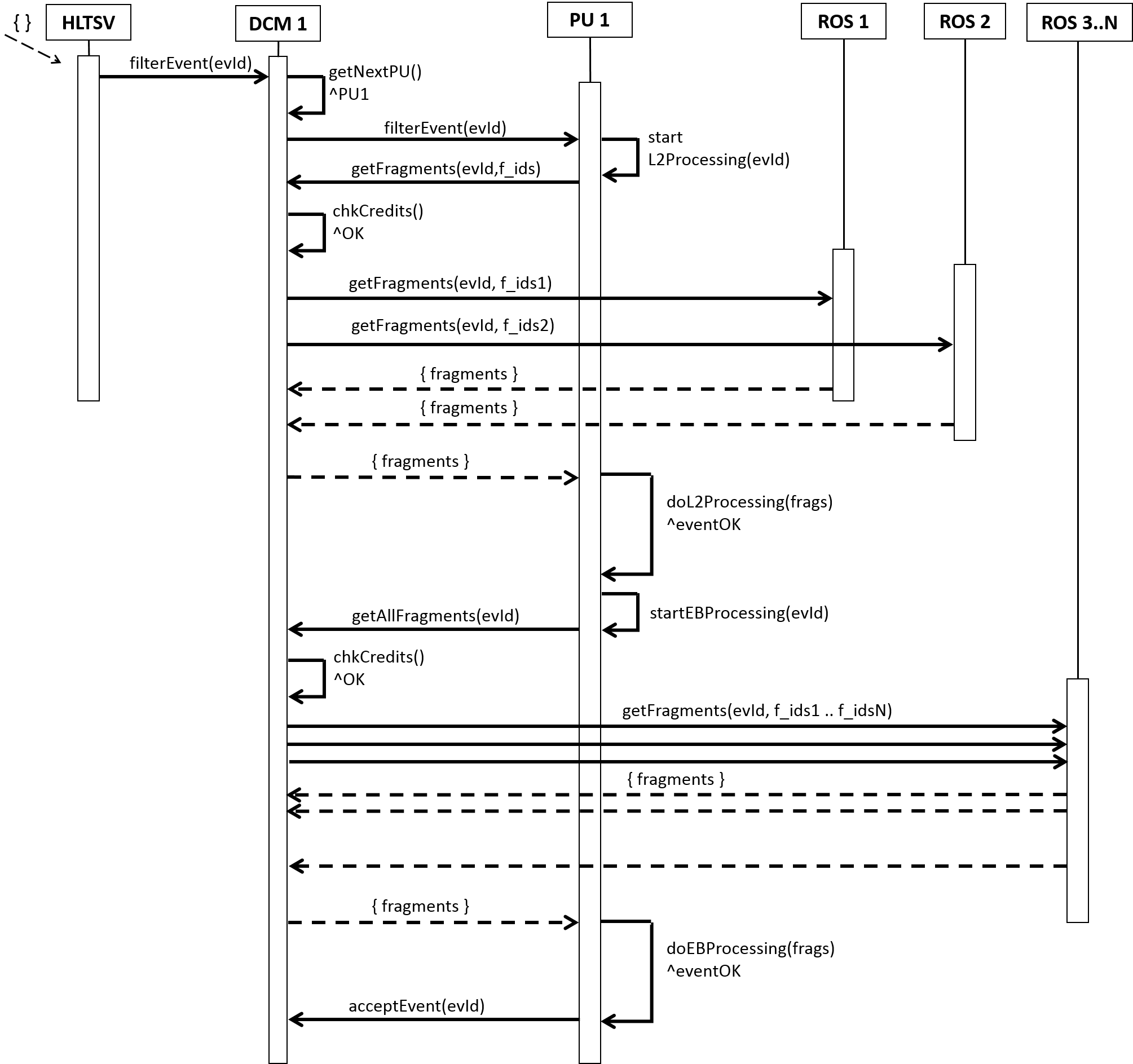
Then we ran an *Explore Cycle* where we discovered hidden undesirable behaviors in load balancing mechanisms. Such behaviors were confirmed to exist in the real system (section 4.3), and raised the need for improvements along with open questions about possible solutions.

To answer the new questions and provide for predictions, *Hypothesis Cycles* were used to test alternative scenarios that were not exercisable on the real system. We then implemented into the real application a set of improvements that proved satisfactory in the simulated environment. Finally, their true effectiveness was validated in the real world (section 4.4)

During the execution of each cycle, intermediate phases allowed dedicating efforts also to improve theoretical and practical tools that support the DEVS methodology. For instance in the areas of modeling (extending VDEVS, see 4.2), simulation (developing an infrastructure to run parallel simulations on distributed nodes, also in 4.2) and simulation validation.

**The TDAQ Data flow and performance metrics of interest.** The model focuses on the prediction of the HLT data flow performance. The *filtering latency* is selected as the main performance metric, representing the time since the HLTSV assigns an event to a given PU until the event is either discarded or stored.

Thesequence diagram shown in **Fig.4** depicts the applications that take part of event filtering (cf. Fig. 1). The PUs request information from the ROS in two stages called Level 2 filtering (L2) and Event Building (EB). In L2 a small portion of the event is first requested and analyzed. L2 step can be repeated several times until EB takes place and all pending information is requested as a whole. For each requested portion of the event, all involved ROS nodes send their replies to the same DCM almost simultaneously, creating traffic bursts in the direction ROS→DCM that increase the *filtering latency* because of the queuing effect generated at the Core and ToR Switches.



**Fig. 4.** TDAQ applications sequence diagram involved in the event filtering of a single event.

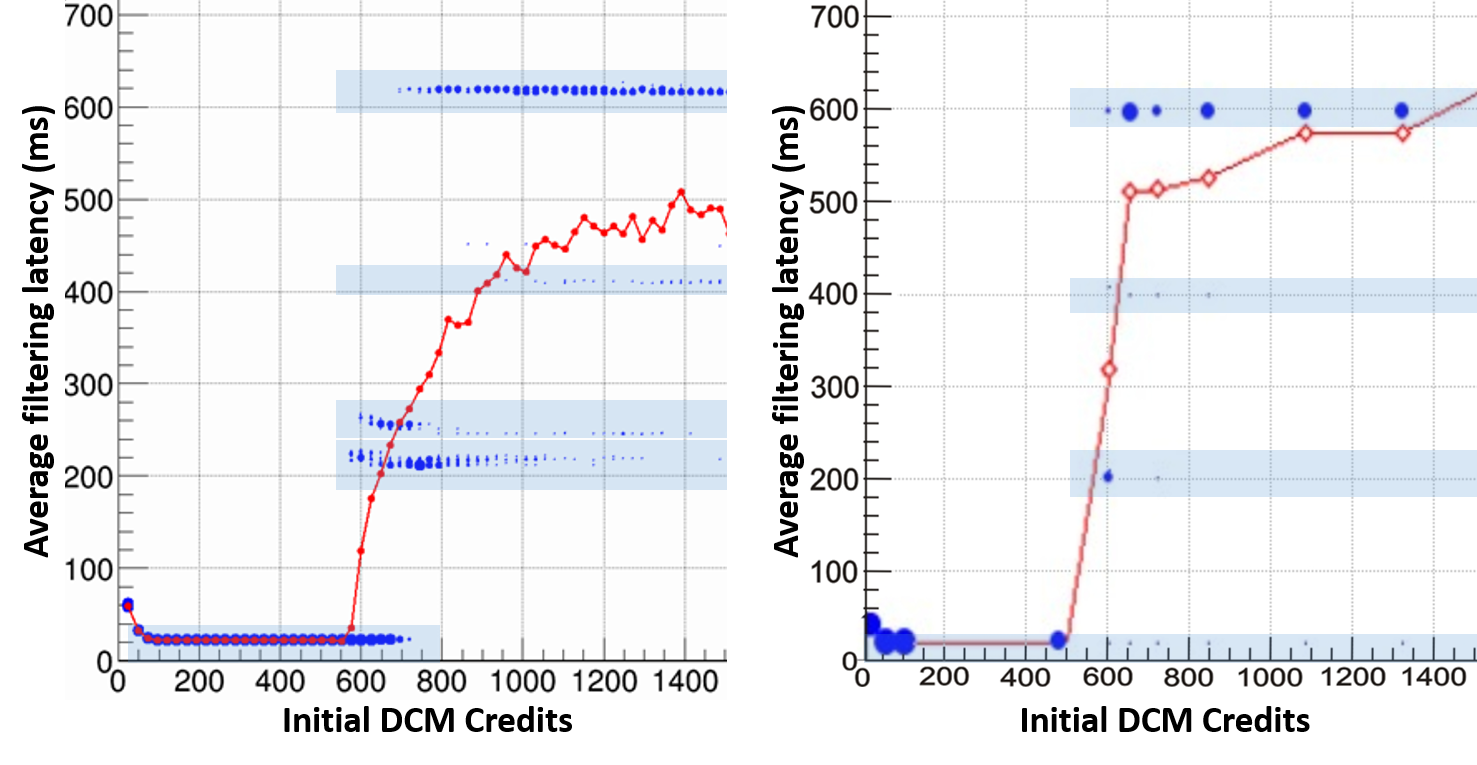
TDAQ has high bandwidth and low latency in relation with TCP minimum retransmission time: 200ms [27]. Together with the data flow described above these conditions create a TCP throughput collapse well-known as TCP Incast [26]. The impact on TDAQ is huge. Whenever a single TCP packet is discarded at the switches, a PU cannot start processing the event until said packet is retransmitted (after 200ms at best), raising the perceived *network latency* of a fragment request from around 1.6ms to more than >200 ms. To avoid the Incast effect, the DCM application restricts the number of simultaneous requests to the ROS using a credit-based traffic shaping control that limits the "in flight" requests on the network. As responses can vary importantly in their size, traffic shaping do not completely prevent packet losses. Thus it is important to study the effects of queue saturation (and TCP retransmissions) and engineer the network and its algorithms in order to maximize performance and minimize high latency risks. This is where our M&S-driven network engineering methodology comes into play.

## First iteration: Building of the model

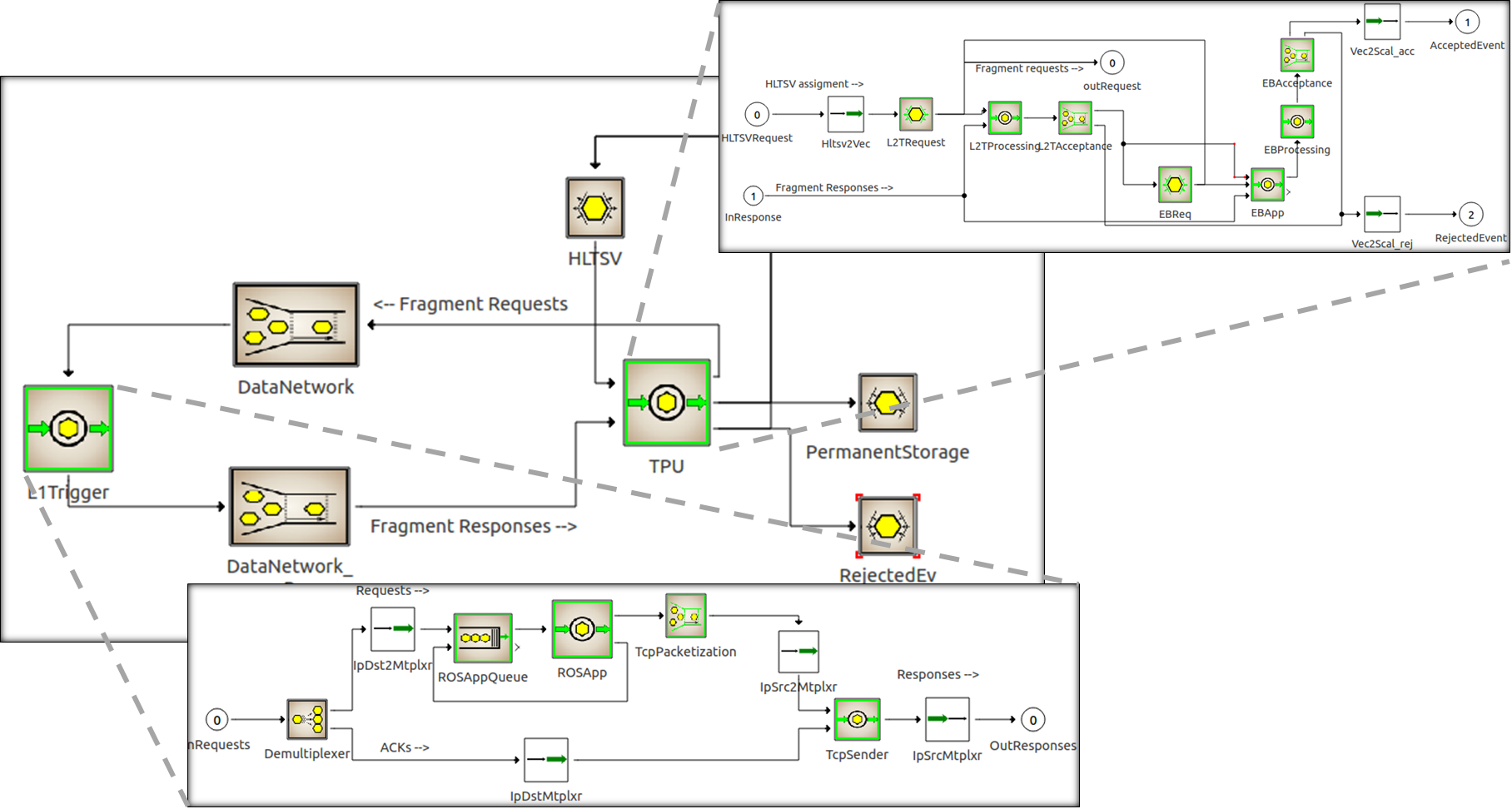
We start the model implementation with a *Build Cycle* (blue cycle in **Fig 1**). We defined the experimental framework EFS for this cycle as a subset of the complete system: the HLTSV, all ROS nodes and a single instance of the DCM and PU applications.

**Real System measurements.** The *Build Cycle* begins with the observation of the real system (experimentation and metrics acquisition)  so we measured *filtering latency* in different scenarios. Experiments were defined using ΘS={# initial DCM credits} and results stored in λS. In **Fig. 5** (Left) it can be seen that an optimum configuration exists where *average* latency stabilizes at ~20ms within a range of about 100 to 600 DCM credits. With fewer credits (12 to 100) latency increases (DCM can send less simultaneous requests, underutilizing network capacity). Using more than 600 credits, latency increases rapidly and stabilizes around 500ms. Coincidentally, we observed packet discards on the ToR switches when more than 600 credits were used, thus verifying that the latency increase is due to network congestion and TCP retransmission mechanisms (no packet loss was observed at Core switches).

**Fig. 5.** Filtering latency vs. initial DCM Credits (1DCM, 1PU). **Lef**t: Real System measurements. **Right:** Simulation results. *Red curve:* Averages latency. *Blue dots:* individual latencies larger dot clusters denote higher number of occurrences, which gather around discrete ranges (~15ms, ~200ms, ~400ms, ~600ms)



**Model implementation.** The *Build Cycle* continues with the creation of a DEVS model guided by the TDAQ architecture and data flow described before. **Fig. 6** shows a PowerDEVS view of the implemented TDAQ model.



**Fig. 6.** TDAQ simulation model implemented in PowerDEVS. Atomic and coupled models.

To preserve the semantics of the real system, a hierarchical model was built complying with the TDAQ naming and structure conventions. This greatly facilitated to extract some control logic straight out of the C++ algorithms in the real applications, thus maximizing the homomorphism with the system under study. ROS and DCM coupled models implement the TCP flow and congestion control logic based on pre-existing PowerDEVS [16] libraries. TCPSender models TCP Cubic [28], implementing only the TCP behavior relevant for the case study. Tests to validate the TCP model against the real system shifted our focus from the average latency (red curve in **Fig 6**) to discover the clustered latencies pattern (blue dots). While the explanation for the occurrence of clustered latencies is outside of the scope of this document, it has a central role in the TCP Incast effect. Moreover, the modeling efforts led to the detection of a bug in the Linux SCL6 TCP implementation [29] which is responsible for the (unexpected) cluster around 600ms.

Following the *Tools Development* approach, TCP atomic models (sender and received) and network nodes (channels and switches) were implemented generic and reusable, incorporating them to the PowerDEVS network library. New Scilab and ROOT [30] visualization mechanisms for post-analysis of latencies were also implemented along with new distributed simulation launching infrastructure for sweeping a given parameter space. These tools are meant to be reused in generalized simulation applications.

**Simulation results and validation against the real system.** Next step of the *Build Cycle* is model verification and simulation validation. We configured the simulation to follow the real system setup described above (controlled ΘS→ΘM→ΘC translation) sweeping the number of initial DCM credits. Results are shown in **Fig 5** (Right). Simulation reproduce the individual filtering latencies (blue points) following the same clustered patterns, validating the dynamics of TCP (retransmissions and TCP Incast effect). Also the simulated average latency approximates real measured latencies (λS ~ λC), where using 100 to 600 credits the latency attains a minimum, and with less than 100 credits latency increases slightly. For credits above 600 simulation also showed congestion and packet drops on the ToR switches, but the increase of the average latency is much steeper compared to the real system. Another difference is the stabilization point under congestion, as the real system latency stabilizes at ~500ms whereas the simulated latency grows up to ~700ms. Although these differences remain to be further studied, the simulation reproduces very closely the intervals of major interest. This evidences the constant tradeoff between degrees of model detail, simulation accuracy, and delivery times for a given engineering concern.

An important advantage of the simulated model is that it allows for fine grained analysis (packet by packet if required). For example, link utilization and queue occupancies can be visualized and studied in detail in the simulation, whereas it is impossible to measure them precisely in the real system due to sampling capacity limitations (e.g. critical queuing bursts occur in less than 8ms).

## Second iteration: Model improvements

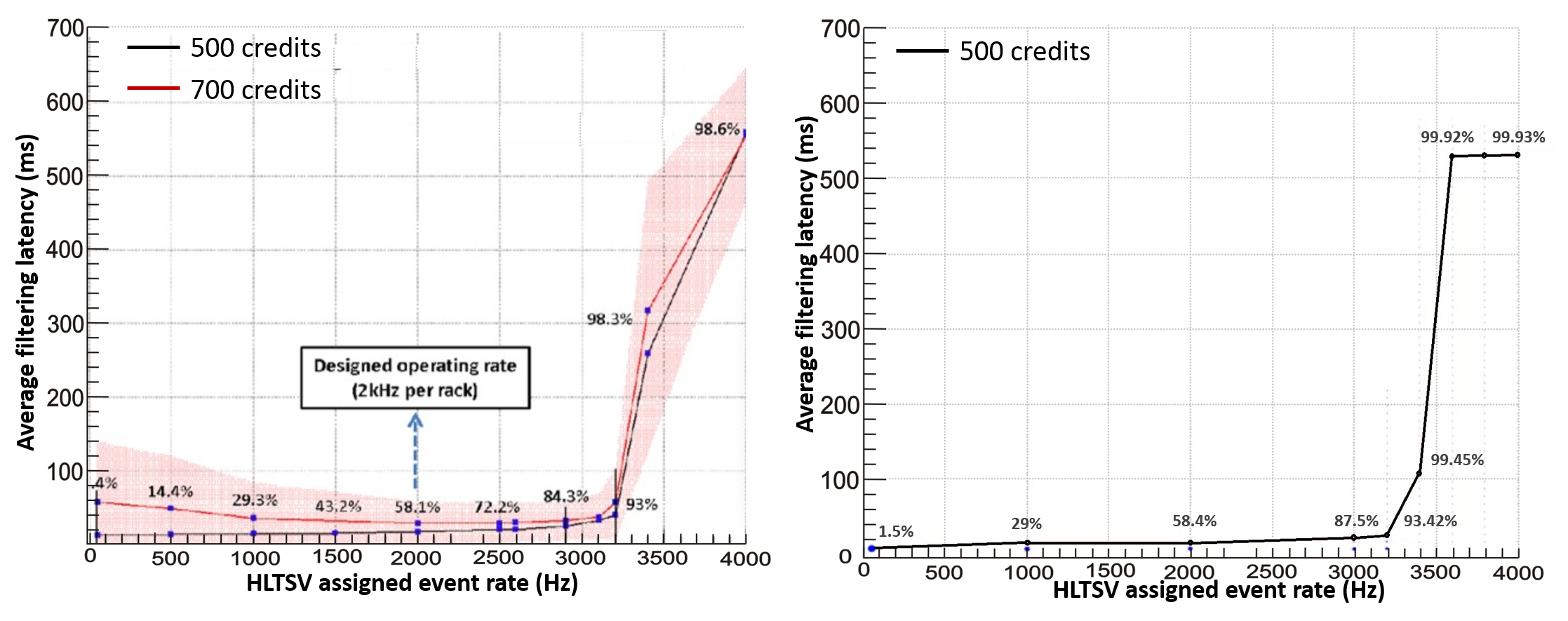
In a second iteration of the *Build Cycle*, the system’s experimental framework EFs was expanded by increasing the number of instances of DCM and PU applications. Also during this cycle, the real system was upgraded calling for changes in the model.

**Upgrades on the real system - changes in the network topology.** The TDAQ team commissioned several changes in the original HLT network (**Fig 1**) in preparation for the Run2 phase of ATLAS where maximum particle’s collision energy was doubled. The ROS ToR switches were removed and the 200 ROS nodes were replaced by 100 new computers with four 10Gbps interfaces, each directly connected to both Core switches. ToR switches were expanded with an additional 10Gbps links towards both Core Switches.

**Real System measurements.** Again, the first step in the *Build Cycle* is taking real system metrics from the upgraded system (all new ROS nodes and a full rack of TPUs -40 DCMs, 960 PUs-). In this scenario, network traffic is largely determined by the HLTSV assignment rate. Thus, new experiments must sweep this parameter (ΘS={HLTSV rate}). PUs were configured to accept events 50% of the time, and the DCM to use 500 and 700 credits.

**Fig.7** (Left) shows the average event latency for increasing the HLTSV assignment rate. When the HLTSV assigns events at 50Hz, latency is minimal (~13ms) because the network is completely free when applications start filtering each event. For increasing assignment rates, latency rises as several PUs simultaneously request for events, sharing network resources and DCM credits. After ~3.2kHz rates, latency increases exponentially as the maximum capacity of the network (93% utilization) is reached.

**Model implementation.** Model changes related to the topology upgrades were minimal: ROS ToR switch models were easily removed thanks to the modularity fostered by DEVS, and the channel’s configuration changed to match the new link capacities. This shows the model flexibility and the advantage of having a one to one mapping between the components of the real system and the simulation model. At this stage a complete HLTSV implementation was developed, reusing chunks of C++ code from the real HLTSV application, providing greater reliability. To increase the number of model instances VDEVS was used, developing 16 new vectorized DEVS models and 10 new multiplexer models to represent packet routing.



**Fig. 7.** Average event latency sweeping HLTSV assignment rate (40 DCMs, 960 PUs).   
**Left:** Real System measurements. **Right:** Simulation results.

Regarding the *Tools Development* phases, three generic solutions were implemented to address the scalability requirement of increasing 50 times the number of simulated instances. VDEVS's original proposal was extended allowing for SmartPointers in vector DEVS messages. SmartPointers were also included directly into the PowerDEVS simulation base engine to allow for automatic and transparent memory management in any atomic DEVS model. This approach reduced dramatically the memory footprint of the simulator, pushing its scalability to a next order of magnitude. Also a new general framework was developed for PowerDEVS to automatically launch simultaneous simulations on distributed nodes, greatly reducing simulation times for parameter sweeping experiments.

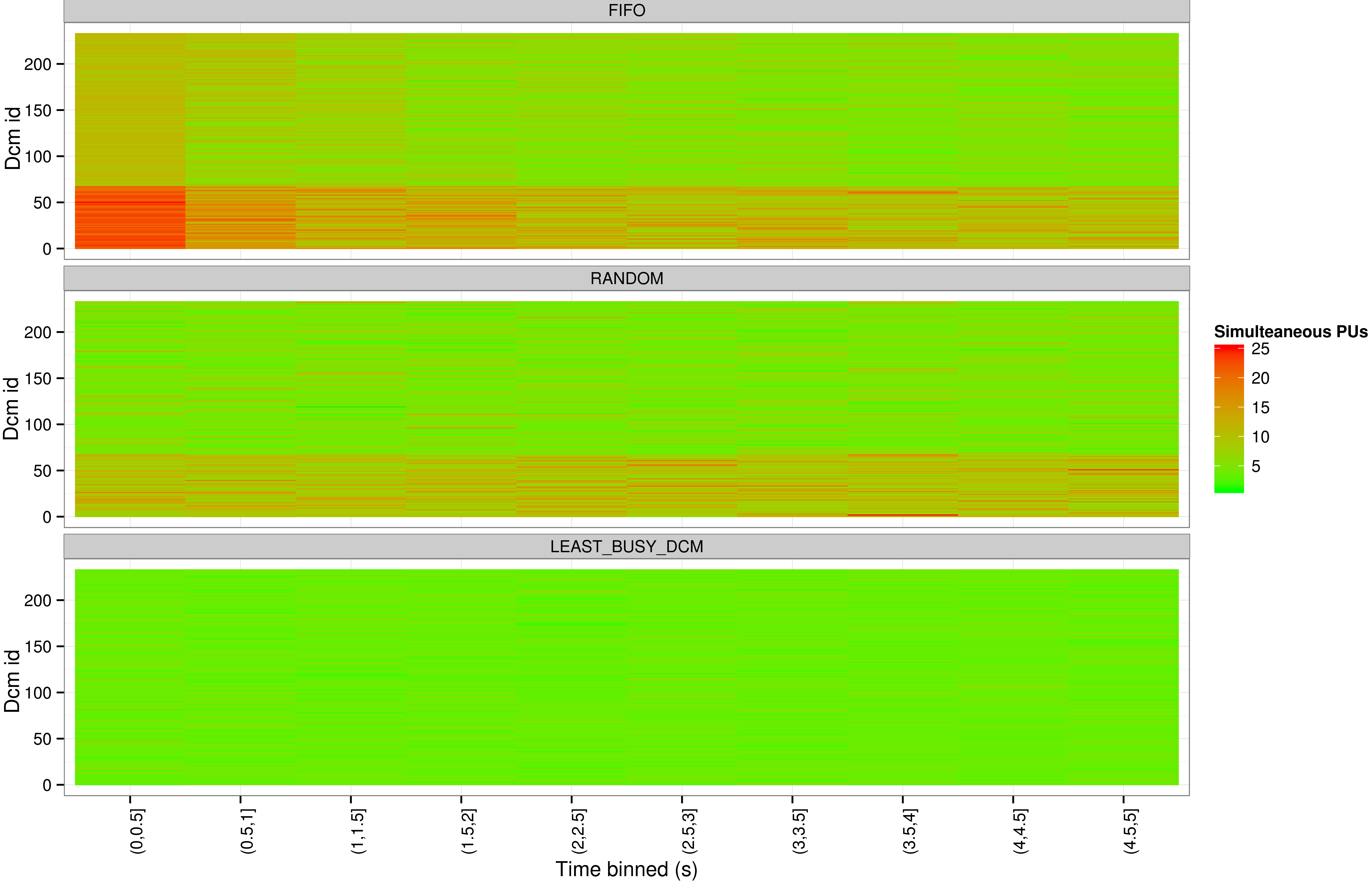
**Simulation results and validation with the real system.** To complete this *Build Cycle*, simulations were validated against the real system. We replicated experiments previously conducted in the real system, sweeping the HLTSV assignment rate parameter. 9 experiments, each simulating 60 seconds (~180K filtered events in the most stringent case), were executed in 3 different nodes, completing all simulations in ~120 min. As shown in **Fig.7** (Right) the simulation results reproduce closely the latency curve measured in the real system. The absolute latency values and network load on the simulation differ from the reality within an acceptable range (<5% difference), showing a good degree of validation.

## Third iteration: Explore and discovery of system-level behaviors

The simulation accuracy obtained on previous cycles provide us with a high confidence level on the simulated data that justifies running an *Explore Cycle* (green in **Fig 1**) to search for potential emergent behaviors.

Full system simulations at design HLTSV rates generate huge volumes of information for the 6M events filtered in 1 minute (e.g. processing times, filtering latencies, queues occupancy, link usages, farm utilization, etc.).

Some of this information is not even available in the real system, or is too difficult to gather uniformly for post-analysis goals. Simulations can provide information, at diverse levels of precision (fine or coarse grained) in a consistent format that facilitates analyses.



**Fig. 8.** Heatmap of the load in the HLT farm for different HLTSV assignment algorithms. Tile color represents the maximum amount of PUs that simultaneously processed in each DCM (230 DCM IDs in the vertical axis) during 0.5s (5 seconds binned in the horizontal axis).

Figure 8 is an example data analysis performed on the simulation results (*Compute Behavior Database,* λC). It shows how events are distributed across the farm in different time slots using various load balancing algorithms: Fist-In-Fist-Out (FIFO), discussed in this section, is currently implemented in the HLTSV for selecting the TPU node that will filter the next event, and other algorithms will be discussed in following section.

The reddish area at the bottom explains the fact that 30% of the DCMs had double the amount of PUs available for processing, thus explaining their higher load. All DCMs are heavily assigned in the first time bins and after a few seconds of execution, load becomes similar to the RANDOM algorithm (as each event filtering time differ). Another detected system-level behavior is that for any single DCM, it processes a very different amount of events along the time slots. Also, for any single time slot the amount of events process by each DCM is very uneven (i.e. the color intensities vary noticeable along any single row and along any single column).

These observations led us to infer that a potentially uneven load balancing mechanism might be the cause of overall higher filtering latencies.

Regarding the *Tools Development* phase, at this cycle we developed a set of R [31] libraries for data analysis and visualization of big amounts of logging information produced by PowerDEVS. Furthermore, the new graphical information generated through the R platform (such as heatmaps like in Fig.8) became the standard means of communication with the TDAQ team, including the capability of regenerating reports automatically when new simulations are run.

## Fourth iteration: Real system improvement proposal

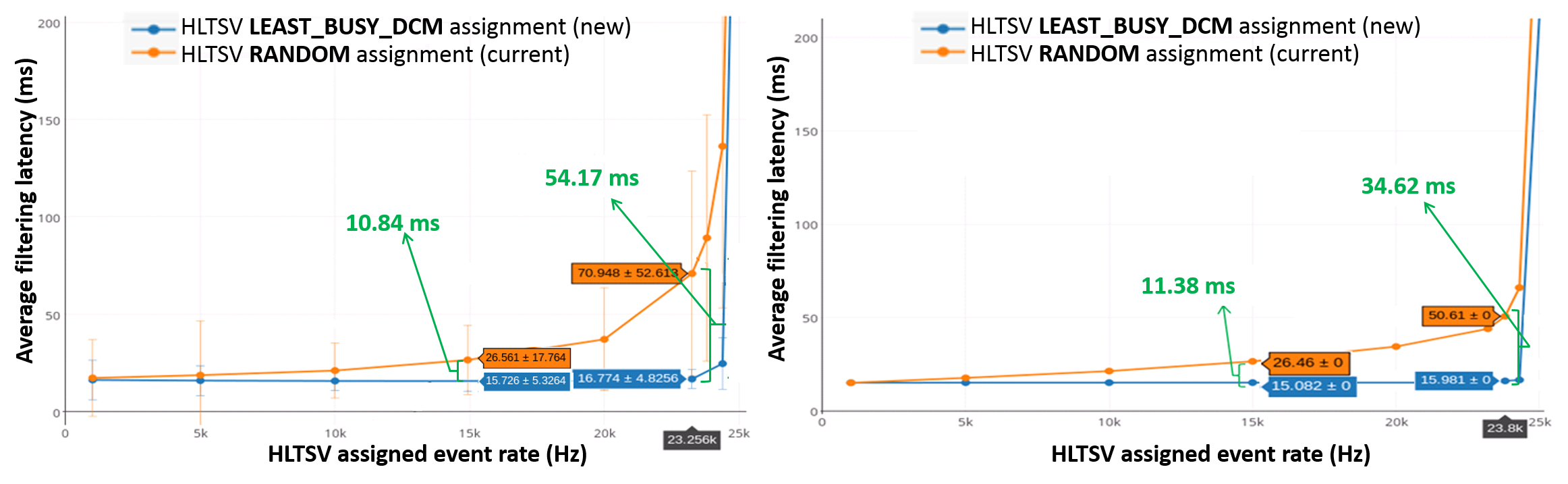
The revealed behavior discovered during the *Explore Cycle* motivated us to move forward towards a *Hypotheses Cycle* (in red in **Fig 1**) to test new load balancing proposals in a simulated domain in search for improved performance.

**Testing hypothesis on the model - Improved HLTSV load balancing proposal.** The latency's linear increase seen in **Fig. 7** is the effect of several PUs competing for the same resources (also reddish tiles in **Fig 8**). However, under the conditions of this experiment and for the designed frequency (2 kHz per rack), each DCM receives, on average, one assignment each every 50-60ms. On the other hand, minimum latency for filtering an event is ~13ms (under optimal conditions). Then, under an optimal assignment policy, it is not necessary that two PUs process simultaneously on a single DCM. However, currently such assignments are random (uniformly distributed), so there are cases were 10 to 25 PUs of the same DCM process simultaneously while other DCMs are almost idle as shown in the DCM load in **Figure 8.**

Three HLTSV assignment policies were modeled: **1)** FIFO, used by the real system, **2)** RANDOM, the HLTSV selects a random idle PU, and **3)** LEAST\_BUSY\_DCM, the HLTSV select an idle PU within the DCM with fewer busy PUs. The main idea behind LEAST\_BUSY\_DCM is to revert the uneven load detected in the *Explore Cycle* by assigning events according to the load on each DCM.

**Simulation results.** After implementing new alternatives in the model, simulations were performed to compare the RANDOM algorithm with the new proposed LEAST\_BUSY\_DCM algorithm (FIFO is omitted because after a while is equivalent to RANDOM as shown in **Fig. 8**). In order to compare, we simulated the same experiment as in the previous section (sweeping the HLTSV rate) but configuring 9 TPU racks (267 DCMs and 6408 PUs). **Fig. 8** showsthat LEAST\_BUSY\_DCM effectively balances the load of all DCMs in the farm, reducing the amount of simultaneous PUs processing in each DCM (tiles colors present more similarity along rows and columns). **Fig.9** (Right) shows simulation results comparing both algorithms. RANDOM algorithm shows the same behavior described in the previous section where FIFO algorithm was used. As for the new algorithm, the simulation shows that the average latency is kept constant and with minimum event latency (~16ms) for frequencies below 24kHz. For higher frequencies, the latency grows exponentially due to network congestion. These results suggest that, for this configuration, the new algorithm could reduce latency between two to four times.

**Implementation and Validation in the real system.** Once hypothesis are tested in the simulation, the next step in the *Hypothesis Cycle* is to implement changes to validate against the real system. It was possible to reuse some code of the logic developed for the simulation, with some minor adaptations to attain close-to-real time performance (the 100 kHz rate requirement for HLTSV is a stringent one). The same experiment was performed in the real system: HLTSV rate sweeping using 9 TPUs racks. **Fig. 9** (Left) shows the result of comparing RANDOM and LEAST\_BUSY\_DCM algorithms in the real system. With the new algorithm and rates under 24kHz the average latency is kept minimum and also shows improvements of two to four times as compared with the current FIFO algorithm, as predicted in the simulation. The simulation is thus validated, showing that the model is capable of reproducing known behaviors, representing a valuable tool to predict the impact of changes in the real system.



**Fig. 9.** Comparison of assignment policies (RANDOM and LEAST\_BUSY\_DCM).   
**Left:** Real System measurements **Right:** Simulation results

# Conclusions and Future Work

We presented an iterative process-based M&S-driven methodology and its practical application to a network engineering case study in a large-scale scientific project: the TDAQ system of the ATLAS experiment.

The methodology allowed us to fulfill the elicited requirements, obtaining well-structured incremental simulation deliverables in short times. All results are formally backed by the DEVS mathematical formalism that helped in keeping strictly separated (yet coherently integrated) system measurements, model formulations, simulation tools and simulation results.

*Build Cycle*s allowed us to build ​​a simulation model of incremental accuracy that satisfactorily reproduces real system behavior under different scenarios of interest. *Hypotheses Cycles* permitted using M&S to design and evaluate improvements in TDAQ load balance control algorithms before commissioning them into the real system, relying on accurate enough predictions produced by the simulation model. *Explore Cycles* proved efficient for discovering unthought-of system-level behavior by mining and analyzing the huge database of fine grained information produced in each simulated experiment. Using *Tools Development* phases general, reusable enhancements were implemented for the underlying DEVS-based modeling and simulation platform (PowerDEVS) and its ecosystem of helper tools (e.g. Scilab, R Studio).

Currently we are incrementing the number and variety of TDAQ scenarios, where different candidate traffic control techniques shall be studied in the search for further performance improvements (in particular looking for quick recovery times in the face of systems failures). We also plan to apply our methodology and tools to assess candidate ATLAS upgrades (planned 2018). Ongoing research is also in due course to automate parameterization-simulation-validation cycles by crawling the huge database of recorded ATLAS experiments available worldwide.

# References

1. B.P. Zeigler, “Theory of Modeling and Simulation”, John Wiley & Sons, New York, 1976
2. A.C.H. Chow and B.P Zeigler, “Parallel DEVS: a parallel, hierarchical, modular, modeling formalism”, Proceedings of the 26th conference on Winter simulation, Society for Computer Simulation International, 1994, pp. 716-722.
3. B.P. Zeigler, H. Praehofer, and T.G. Kim, “Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems”, Academic Press, 2000.
4. G. Wainer, “Discrete-event modeling and simulation: a practitioner's approach”, CRC Press, 2009.
5. G. Wainer and J. Mosterman, “Discrete-event modeling and simulation: theory and applications”, CRC Press, 2010.
6. G. AAD et al, “The ATLAS experiment at the CERN large hadron collider”, Journal of Instrumentation, S08003 , vol. 3, no. 08, 2008.
7. D. Pestre, “L'organisation européenne pour la recherche nucléaire (CERN): A succès et politique scientifique”, Vingtieme siecle, Revue d'histoire, 1984, pp. 65-76.
8. L. Evans and Philip Bryant, “LHC machine”, Journal of Instrumentation 3.08, S08001, 2008.
9. CMS collaboration, “The CMS experiment at the CERN LHC”. JINST 3, S08004, 2008, pp. 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”, Jinst 3, S08005, 2008.
12. G. Wainer and R. Castro, “A Survey on the Application of the Cell-DEVS Formalism”, J. Cellular Automata, vol. 5, no. 6, 2010, pp. 509-524
13. A.C.H. Chow and B. Zeigler, “Parallel DEVS: a parallel, hierarchical, modular, modeling formalism”, Winter simulation, 1994.
14. F. Bergero and E. Kofman, “A vectorial DEVS extension for large scale parallel system modeling and simulation”, Simulation 90.5, 2014, pp. 522-546.
15. F. Bergero and E. Kofman, “PowerDEVS: a tool for hybrid system Real-time modeling and simulation.”, Simulation 87, no. 1-2, 2011, pp. 113-132.
16. R. Castro, “Integrating tools for modeling, simulation and control of Data Networks”, PhD Thesis, FCEIyA, National University of Rosario, Argentina, 2010, pp. 135.
17. OMNeT++ Community, “OMNeT ++ Discrete Event Simulation System”, 2004, www.omnetpp.org.
18. T. Issariyakul and E. Hossain, “Introduction to Network Simulator NS2”, Springer, 2008.
19. X. Chang, "Network With OPNET simulations.", Proceedings of the 31st conference on Winter simulation: Simulation --- a bridge to the future-Volume 1. ACM, 1999, pp. 307-314.
20. F. Cellier and E. Kofman, “Continuous System Simulation”, Springer Science & Business Media, 2006.
21. R. Castro and E. Kofman, “An integrative approach for hybrid modeling, simulation and control of data networks based on the DEVS formalism”, Modeling and Simulation of Computer Networks and Systems, Elsevier, DOI: 10.1016/B978-0-12-800887-4.00018-3, 2015.
22. P. Kruchten, “The Rational Unified Process: An Introduction”, Addison-Wesley Professional, 2004.
23. K. Beck, “Extreme Programming Explained: Embrace change”, Addison-Wesley Professional, 2000.
24. S.R. Palmer and M. Felsing, “A practical guide to feature-driven development”, Pearson Education, 2001.
25. D. Astels, “Test driven development: A practical guide”, Prentice Hall, 2003.
26. S. Kulkarni and P. Agrawal, “Analysis of TCP Performance in Data Center Networks”, Springer, 2014
27. J. Postel, "Transmission Control Protocol", STD 7, RFC 793, 1981
28. S. Ha, I. Rhee and L. Xu. "CUBIC: a new TCP-friendly high-speed TCP variant.", ACM SIGOPS Operating Systems Review, vol. 42, no.5, 2008, pp. 64-74
29. Red Hat Bugzilla, "TCP RTO additional Creates bug in very specific condition", https://bugzilla.redhat.com/show\_bug.cgi?id=1203742, 2015
30. I. Antcheva, M. Ballintijn, B. Bellenot, M. Biskup, R. Brun, N. Buncic, Ph Canal et al., "ROOT - A C++ framework for petabyte data storage, statistical analysis and visualization.", Computer Physics Communications, vol. 182, no. 12, 2011, pp. 2499--2512.
31. Core R Team, "R Language Definition.", CRAN, 2000.
32. B. Gonçalves and F. Porto, “Managing Scientific Hypotheses as Data with Support for Predictive Analytics”, Computing in Science & Engineering, vol. 17, no. 5, 2015, pp. 35-43.

1. This is an adapted/extended version of the DEVS Formal Framework, using concepts from [1,4 and 5]. [↑](#footnote-ref-1)