MARS: A Flexible Real-time Streaming Platform for Testing Automation Systems

Raphael Eidenbenz, Alexandru Moga, Thanikesavan Sivanthi, Carsten Franke
ABB Corporate Research
Baden-Dättwil, Switzerland
firstname.lastname@ch.abb.com

***Abstract*—Trends in industrial automation systems are placing more importance on using streams of digitized data to perform various automation functions in real-time, e.g., power, process, and factory automation. To ensure high reliability and availability, individual devices or (sub-)systems thereof need to be tested with respect to their expected real-time behavior in the system context at various stages during product and project life cycle. In this paper, we introduce a real-time streaming platform called MARS that provides the means to monitor (M) high frequency data streams, analyze (A) them with respect to their properties, record data traffic (R) based on user-specified rules, and simulate data traffic (S) and in general the behavior of certain functions of one or more devices according to highly customizable scenarios. We present the design principles of MARS, the real-time software architecture, and evaluation results.**

I. Introduction

Analog inputs and outputs in industrial automation systems (IAS) are more and more replaced by digital communication, which allows devices to transmit and receive messages to and from other devices via a dedicated communication network. Since such devices are often produced by different vendors and configured independently many IASs are only complete once commissioned on site. Since resolution of integration errors at such a late stage is very costly, earlier stand-alone and factory acceptance tests should already be done in full system context, even if the system is incomplete. This imposes the need for a platform that supports functional behavior simulation of unavailable devices, and provides monitoring, recording, and analysis capabilities for validation of devices under test. Such a platform has to be flexible enough to allow rapid configuration of different test scenarios. Furthermore, the solution needs to provide concurrent execution of different testing scenarios in certain logical and temporal order with real-time guarantees.

This paper introduces such a platform, called *MARS* (Monitoring, Analysis, Recording, Simulation), which builds on the principles of data stream processing (SP) to allow for flexible real-time simulation and hardware-in-the-loop (HIL) testing of IASs with sub-millisecond accuracy. MARS leverages SP concepts to provide a natural way of composing test scenarios for IASs as streaming applications. Furthermore, the data stream orientation enables scalability and flexibility while handling real-time workloads on commodity hardware.

The paper is structured as follows. Section II describes the principles behind MARS. Section III introduces the SP concepts and the runtime operation underpinning the proposed MARS platform. We then present experimental evaluations of a MARS implementation on Linux with real-time capabilities in Section V, compare the MARS approach to related work in Section VI, and draw conclusions in Section VII.

II. Principles of MARS

MARS is based on a few principles that make it a highly flexible, extensible and customizable platform for testing of IASs, namely, integrated testing, real-time operation, data stream orientation and flexibility.

A. Integrated Testing Functionality

MARS integrates monitoring, analysis, recording, and simulation capabilities into one platform, on the same device or in the same cluster of devices. This is the governing principle of MARS, and offers the following benefits:

- *Direct correlation of events*, such as correlating the stimulus and the response in a test scenario (e.g., measuring round-trip times), is straightforward as the simulation producing the stimulus and the monitoring of the (expected) response are carried out in the same environment, thus benefiting from the same notion of time and a naturally correct ordering of events.
- The collocation of testing capabilities significantly broadens the range of tests that can be executed without specially-tailored setups containing separate simulators, recorders, monitors, or offline analyzers. Thus, test engineering and execution is *more efficient*; tests can be re-used and automated more easily.
- Finally, the integrated approach enables *dynamic simulation*: in order to adapt depending on the behavior of the system under test, the monitoring and analysis functionality can be used to provide feedback to the simulator which can modify its behavior according to the observed changes.

B. Real-time Operation

Many IAS devices have strict timing requirements. For instance, controllers can employ periodic execution cycles in the range of milliseconds to microseconds, and for some domains even nanoseconds. A testing infrastructure thus needs to provide simulation and measurement accuracy in the same order of magnitude. While for accuracies below microseconds, specialized hardware is needed, such as FPGAs or DSPs, commercial off-the-shelf (COTS) computers achieve accuracies in the microseconds, if they are run with a real-time operating system (RTOS), e.g., Linux with Preempt-RT patch. As MARS targets below millisecond accuracy, we rely on

978-3-9815370-8-6/17/$31.00

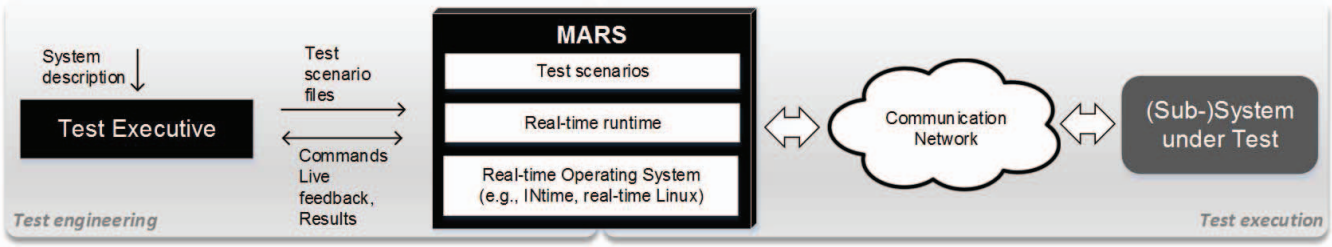

Fig. 1. MARS overview.

RTOS environments on CPU-based computers to provide real-time capabilities by means of reliable preemption based on thread priorities. MARS leverages the preemption mechanism to support different levels of criticality for handling data and processing. Thus, we can address variable timing requirements, as typically present in IASs.

C. Data Stream Orientation

MARS is designed as an *SP platform*. Conceptually, a *test scenario* consists of a set of configured instances of processing elements, which we call *layer objects (LO)*. LOs are interconnected by *data streams*. Streams can carry raw Ethernet, signal, processed, and protocol-formatted data but also events, or logging messages. Nevertheless, all stream items are timestamped according to the same time reference.

The streaming approach allows for a natural engineering of test scenarios, as the functionality of most IASs is centered around processing sensed signals and creating control messages. Moreover, the MARS platform interfaces with the System under Test (SuT) via Ethernet in order to support HIL testing, and thus all stimuli produced by MARS as well as the responses from the SuT are essentially Ethernet data streams. MARS currently targets only testing of devices where all relevant input (and output) is Ethernet. However, MARS could be easily extended to also support other types of data inputs and outputs by implementing the corresponding I/O handlers, e.g., analog or binary signal inputs or outputs.

D. Flexibility

MARS achieves a high level of flexibility and extensibility by offering three degrees of freedom in specifying and configuring test scenarios, as follows:

1. The common set of LOs can be extended by *domain-specific catalogs*, which contain a set of LO types, including loadable implementation and configuration parameter descriptions. Harnessing the LOs in one or multiple such catalogs, test engineers can compose a test scenario by describing the list of interconnected LO instances, their configurations, and data streams between them directly in a *test scenario description file* or via a graphical tool.
2. MARS provides a generic *script object* LO type that can wraps and interpret a custom script. With this mechanism, custom logic that is tailored to a given test can be injected on a higher level of abstraction, without the need to implement an additional, test-specific LO.
3. *User interaction* allows users (i.e., test engineers) to interact with a running test using commands that are meant to enact a certain behavior, for instance, by modifying the parameters of a simulation.

III. MARS System Model

MARS consists of a *real-time core* (MRTC) that includes the necessary runtime to execute test scenarios on top of an RTOS running on COTS hardware (see Fig. 1). The *Test Executive* determines what the MRTC is executing: it loads the test scenario description files, user-defined scripts, and issues commands, e.g. to start a loaded scenario or to interactively adapt a running scenario. The Test Executive receives live feedback, e.g. testing events or information on test execution status, and recorded data from MRTC. Finally, MRTC interacts with the SuT (i.e., automation device or set thereof) over an Ethernet-based communication network.

The MARS system model (see Fig. 2) comprises of the data and the processing models which we describe next.

A. Data Model

LOs can consume and/or produce data in many forms. On the one hand, an LO can process *streaming data*, which can come from various sources: other LOs within the same scenario, the network, other processes on the same host or the host's file system (i.e., recorded data). On the other hand, an LO can react to certain events originating in the system, i.e., *time events*, or in the scenario itself, i.e., *actions*.

1) Data Streams: are defined as a *sequence of data items* with a partial order based on the *timestamps* present inside the data items themselves. The data items can appear in the sequence *periodically* (e.g., data resulting from sampling or from heartbeat-style of reporting) or *sporadically* (e.g., events signaling the change in state of some system variables).

2) Time Events: are callbacks that occur as a result of a timer request by an LO. Such callbacks are either fired once after a given time period (single shot timer) or repetitively with a given period (periodic timer). They are handled by the LO that has set the respective timer. For accurate time calculations, callbacks are timestamped with the event firing time.

3) Actions: are events which originate either from an LO, or from the testing engineer via Test Executive and Interactive Interface (as illustrated in Fig. 2) . Similar to time events, an action is linked to the functionality of an LO, and typically results in the modification of the LO's behavior. An action carries a type and a set of (optional) parameters, which are specific to an LO.

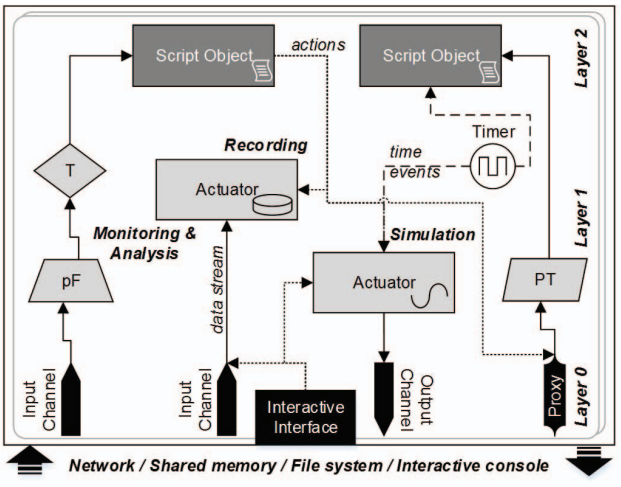

Fig. 2. MARS system model.

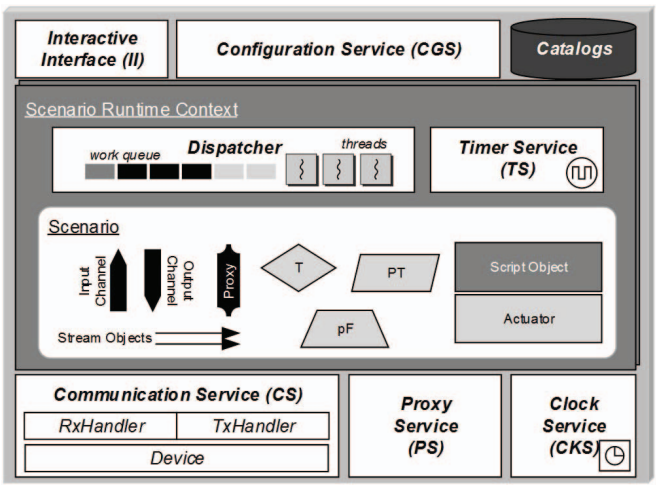

Fig. 3. MARS runtime framework.

B. Processing Model

LOs are grouped into three layers based on their role:

1) Layer 0 (L0): "I/O": $L0$ contains *input and output channels* that typically implement communication protocols or access mechanisms to different sources or destinations, e.g., file system, network, shared memory. That is, $L0$ also enables the *recording and playback capabilities* of MARS.

2) Layer 1 (L1): "Processing and Simulation": $L1$ includes the core functionality for MARS scenarios with respect to monitoring, analysis and simulation capabilities. We distinguish two different types of LOs:

- *Processors*: These are simply data stream processors, such as *pre-processing functions (pF)* (e.g., Fourier transform, Root mean square); *triggers (T)* to raise events while monitoring for special conditions; and *property trackers (PT)*, which maintain statistics on connected input streams.
- *Actuators*: These LOs typically control a certain MARS function, such as recording or simulation. An example of the latter is to generate data by sampling a simulated signal (e.g., a sinusoid wave form). The actuator may rely on time events as input to drive the data generation and it can even modify its behavior, i.e., how the signal is simulated or how the data is generated following inputs from actions.

3) Layer 2 (L2): "Scripting": This layer includes a single LO type, *script objects*, which embed and run user scripts using an embeddable scripting engine, such as the Lua[1] interpreter. A script object works with input data streams as well as time events, which are relayed to the scope of the script. The typical output of a script object is actions, which are issued towards LOs in $L1$ or even $L0$.

IV. MARS Real-time Core

Fig. 3 shows the MARS real-time core (MRTC) consisting of a modular runtime framework that handles the execution of test scenarios. Next we describe the execution model of a scenario as well as the other services inside MRTC.

A. Execution Semantics

MRTC creates a *Scenario Runtime Context* to contain all the elements of a test scenario (LOs, streams), and runs the scenario with the help of a *Dispatcher* and a *Timer Service*. The Dispatcher manages the threads that execute LO instances. That is, it executes the LO logic for consuming data, handling actions, or time events whenever new data is present on input data streams, or when an action time event has been issued to the LO by other LOs or by other MRTC components (e.g., the Timer Service for time events). The Dispatcher implements the Command pattern[2] by maintaining a queue of so-called work items that are dispatched to a set of threads. Each work item represents a *delegate* which abstracts away from the type of work. Besides data-, action- and time-related delegates, the Dispatcher can handle control delegates, subject to scenario runtime management, e.g., starting LOs during scenario deployment, stopping during scenario tear-down.

The Dispatcher threads have *real-time priorities* according to the underlying RTOS. The priorities are specified in the configuration of LOs to represent different criticalities inside a test scenario. LOs that have the same priority are executed by the same thread with that priority. Furthermore, the execution order of LOs can be *asynchronous*, i.e., an LO posts work for another LO to the Dispatcher that may be executed at a later stage, or *synchronous*, to minimize latency along a chain of LOs, i.e., they are executed back to back along the chain.

MARS is required to combine event-driven and time-driven behavior. It does so using a *tick-based approach* for executing work. That is, the timeline is divided into periods of equal size, called *ticks* (e.g., of size 50μs, 100μs). MRTC threads such as Dispatcher threads or the Timer Service thread, perform work in the beginning of a tick, or as soons as the scheduler lets them execute according to their priority, respectively. Whenever the current work is completed, a thread sleeps the beginning of the next tick. Such a tick approach provides the needed determinism with limited overhead, albeit at the expense of latency and jitter (see Section V).

Fig. 4. Single stream scenarios, deployed N times to evaluate scalability.

B. Core Services

The *Interactive Interface* allows test engineers to interact with MRTC, in particular, to load, start or stop scenarios, and even to adapt the behavior of individual LOs via actions during test execution. The *Configuration Service* manages the LO catalogs, and takes care of (re-)configuring MRTC at runtime according to the hosted test scenarios.

MRTC provides a number of utility services that can be used by test scenarios: the *Communication Service* enables transmission and reception of Ethernet network traffic. The contained TxHandler and RxHandler have direct access to the network driver to transmits and receive Ethernet packets using DMA-based swappable ring buffers. The network driver thereby reads or writes an *active* buffer; the TxHandler and the RxHandler read/write a *pending* buffer; the buffers are swapped on context switch. Based on a configurable tick, the TxHandler periodically works off all items (Ethernet packets) in a queue that have a timestamp in the past. Items are added to the queue by the testing scenario LOs and timestamped with desired time of transmission. By contrast, RxHandler is triggered by the network driver via interrupt and forwards received data to the according MARS I/O channels.

Test scenarios interact with the Communication Service using streams, just as scenario LOs connect to other LOs. The same approach was taken when designing the *Proxy Service*, which allows a test scenario to make use of external components, i.e., external processes. An example use case is to add a 3rd party communication stack to a MARS test suite.

Finally, the *Clock Service* can synchronize the used clock to an external time source, e.g., a IEEE 1588 master clock. The test scenario can thus be synchronized with the device(s) under test to achieve accurate simulation and time measurements.

V. Evaluation

To show the real-time capabilities and the flexibility of MARS, our experimental evaluation focuses on two key points: the efficiency of MRTC to handle network streaming data at scale (Section V-A); and the real-time performance thereof when confronted with realistic mixed functionality workloads (Section V-B). Beyond the results presented here, MARS has been successfully used for testing ABB devices.

We conducted experiments using two HP Z840 machines running Preempt-RT patched Ubuntu 14.04 (Linux Kernel 3.14.53-rt54) with Intel network interface cards (NICs) that support hardware timestamping, one typically sending out, one receiving network traffic. We used an LO called *Stats Writer* to measure the device internal latencies and jitter on the receiver: it compares the time of reception at the Stats Writer with the timestamp attached by the NIC to calculate latencies; it measures inter-arrival times of data items, and writes the results to file. On the sending side, we instrumented TxHandler to measure the latencies of outgoing Ethernet packets. Finally, we measured CPU load with the Linux tool *top*.

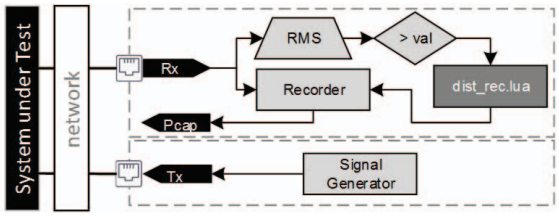

Fig. 5. Scenarios providing mixed functionality.

A. Scalability performance

To understand the performance of MRTC with respect to running tests of different scales, we designed experiments that operate on generic data and include two MARS machines, one acting as data simulator and one as data consumer (see Fig. 4). We simulate two basic types of data streams: (1) *periodic* streams to reflect data such as periodic sensor readings, heartbeats, or status information; and (2) *bursty* streams to reflect other Ethernet traffic, such as Internet traffic or cascading events in a distributed automation system.

In a first series of experiments, the data simulator generates N periodic streams at a rate of 1000 items per second using N *Generator* LOs; the consumer receives, processes, and measures the streams using N Statistics Writer LOs. In a second series, the setup differs only in that the Generator produces data items in a bursty manner. For this purpose, the generator implements the b-model algorithm by Wang et al. [1] with bias 0.8, a time window of 1 second, and total number of 1000 items per second. In both series, each generated Ethernet packet has 18 bytes and consists of the Ethernet header and a 32-bit counter, which is used to determine packet losses. Thus, one stream incurs 18 kB/s on the wire.

Fig. 6 shows the measured device-internal latency distributions on the simulator and the consumer for $N \in \{1, 5, 10, 100\}$. With large enough NIC and MRTC internal buffer sizes, we observed no losses for both periodic and bursty traffic with up to 100 streams. However, when going to 100 streams, while the majority of latencies remain low we observe high maximum latencies: the maximum transmission latency with periodic traffic reaches 92 ms, and 60.6 ms with bursty traffic; the maximum reception latency with bursty traffic 84 ms. The maximum reception latency with bursty traffic was even worse, at 1.4 s for 50 streams. The reason for the high latencies in the mentioned cases is that the system operates close to the capacity limits: even though the MRTC process consumes below 70% of the CPU on average, together with the network interrupt handling it can come close to 100% at some points of the execution. In particular, when there is a concentrated message burst concurrently on multiple streams, new data is generated faster than it can be transmitted and consumed. Without mentioned overload situations, the latencies proved to be only marginally affected by the number of streams, and remain in the range of the tick, which was 125µs for the experiments. The scenarios with bursty traffic show some more peaks in latencies, but remains within the range of two to three times the tick.

In terms of CPU load, the MRTC process showed to scale by and large linearly in N on both the simulator and the

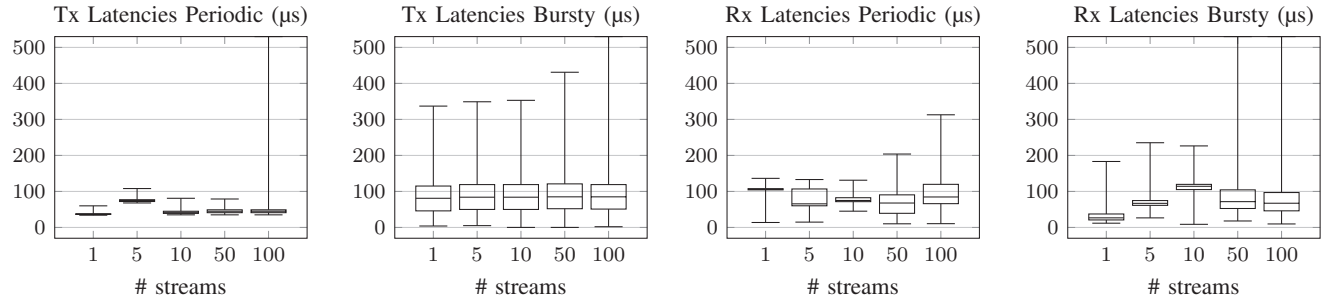

Fig. 6. Distributions of internal latencies on simulation (Tx) and monitoring (Rx) side, with periodic or with bursty traffic. The box shows the 25th percentile, the median, and the 75th percentile; the whiskers show the minimum and the maximum latency observed.

consumer side, up until the point where we have some overlead situations in the system, as observed with $N = 100$. In all experiments, the simulator scenario used less CPU than the consumer scenario: it used from about 9% at $N = 1$ to 38% at $N = 50$ on average for either type of traffic, whereas the consumer used from 14% at $N = 1$ to 44% at $N = 50$.

In terms of scalability, MARS generally shows desirable behavior, namely, that latencies are kept within bounds related to the tick, and grow only marginally with the number of streams. This is, if the test scenario does not demand resources close to hardware capacity limits. In overload situations, such as when a (too) large number of streams has to be generated or processed, or when data is produced in fast bursts, the latencies can grow beyond acceptable limits.

B. Real-time performance

We also evaluated the real-time performance with respect to a more sophisticated workload comprising of two scenarios running in parallel in the same MRTC instance. The first (Fig. 5, bottom) uses a *Signal Generator* to generate a periodic sinusoidal signal at a sampling rate of 4 kHz. The simulation is configured in such a way that every 2.5 seconds the signal's amplitude is increased above a certain threshold and then decreased to the original size the next second. The second scenario (Fig. 5, top) analyzes incoming network traffic and triggers (as an action via a Lua script) the recording of 1000 incoming Ethernet packets to a PCAP file whenever the RMS value of the incoming sinusoidal signal exceeds the threshold. In the experimental setup, we route the generated signal of the first scenario via a network switch and a separate Ethernet port to the second scenario running on the same MRTC.

We focused our evaluation on two main aspects that affect real-time performance in MARS: (i) the choice of tick length, which is used by the MRTC threads, and (ii) the priority settings for scenarios. For the former aspect, we ran the simulation scenario in isolation with different tick configurations; for the latter, we ran both scenarios concurrently on one MRTC, and varied the priority of the second scenario relative to the first. To measure the real-time performance, we investigate the effect on the jitter of the periodic sinusoidal signal, as measured on a different, connected machine.

MARS is required to be flexible enough to accommodate simulation of data streams with different characteristics. For periodic data streams, we can expect MARS's tick-based execution to introduce jitter that is relative to the tick length. In fact, the results of running the same simulation scenario with different tick settings in Fig. 7 show that there are several distinguished cases. For 10μs, the inter-arrival times cluster around 250μs, the actual period of the 4kHz signal. Such a low tick setting yields low jitter, however, at the expense of high CPU load, i.e., 58% compared to <15% for the other tick settings. Another case can be seen at 125μs which happens to be half the signal period. In such a case, we can see again clustering around 250μs but some smaller clusters appear at 100μs and 400μs. This indicates the fact that sometimes the system is pushed to delivering data in a later tick and then immediately to catch up. Yet another case is where we see only two clusters on either side of 250μs for the 100μs and 200μs tick settings. While the latter setting yields comparably high jitter, the former provides a good compromise as we can then expect inter-arrival times to be $+/-$ one tick off of the signal's period. In general, it is expected that the tick is set as part of calibration for a given domain and that it is preferably chosen so that the signal period is not a multiple of the tick.

We used the 100μs tick setting for the case shown in Fig. 5. We initially set the priority of both scenarios to 96. Fig. 8 shows the inter-arrival times between seconds 4 and 11 of the experiment. Most measured inter-arrival times fall in the same case as shown in Fig. 7 but every 2.5 seconds we see a high spike (reaching maximums of around 30ms) and subsequent drops. In fact, these spikes occur when the data is flushed to disk, and thereby preempting or postponing the data generation and transmission tasks. Consequently, we lowered the priority of the recording scenario to 50, but still observed spikes, albeit with lower values. The reason is that the data recording process was competing with the network interrupt handler thread which has priority 50 by default. Lowering the priority further to 20 yields no difference between the simulation scenario with or without the concurrent recording scenario. Thus, setting priorities correctly for scenarios or individual LOs enables timely concurrent execution of mixed scenarios also if I/O operations are involved.

VI. Related Work

Real-time digital simulations are widely used during rapid prototyping and product development phases in many different domains, e.g. aerospace, automotive, electrical, and defense

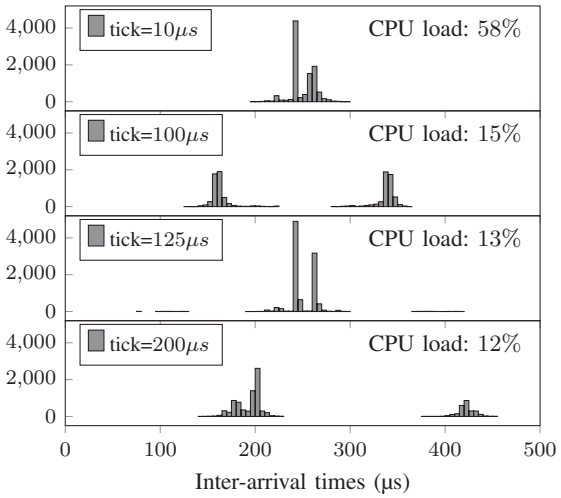

Fig. 7. Histogram of inter-arrival times for a periodic 4 kHz signal vs. tick.

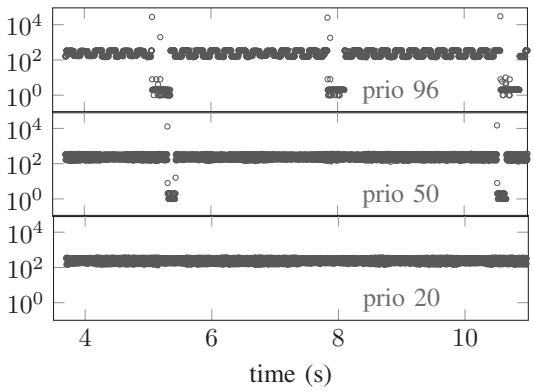

Fig. 8. Inter-arrival times (in μs) for periodic traffic vs. recording priority in a typical mixed functionality scenario.

industries [2]. Many commercial vendors such as RTDS, Opal-RT, and dSPACE[3] have developed real-time digital simulation solutions for these industrial segments. RTDS uses digital parallel processing hardware for performing real-time power system simulations. Opal-RT uses COTS PCs and FPGA-based reconfigurable I/O for real-time simulation. dSPACE offers hardware-in-the-loop simulators for testing of mechatronic systems. These real-time digital simulation platforms use specialized hardware such as FPGAs or DSPs and are not flexible enough to build adaptable and extensible simulations. The MARS architecture provides this flexibility and extensibility on COTS hardware. There are also several testbeds developed by the scientific community for testing of industrial control systems. A survey of these testbeds is presented by Holm et. al [3]. These testbeds are mostly developed for education purposes, for performing vulnerability analysis or for testing the defense mechanisms of industrial control systems.

Real-time digital simulation is also widely used in analysis and testing of power systems. Blair et. al [4] present a model-driven approach to derive a communication stack for real-time testing of protection and control schemes. They use RTDS hardware for physical process simulation. Mo et. al [5] present an evaluation platform based on specialized hardware for measuring the performance of protection relays under different communication setups. Zadeh et. al [6] discuss a test setup that focuses on evaluating performance of a new busbar protection algorithm using playback data together with background traffic. Bo et. al [7] investigate the application of integrated protection schemes in substations using RTDS. Almas and Vanfretti [8] focus on testing a protection relay using Opal-RT for physical process simulation. The testing of protection and control devices under abnormal system conditions using RTDS is presented in [9], [10]. These setups mainly focus on testing of individual devices and most of them are based on specialized hardware. MARS on the other hand can perform testing of individual devices as well as subsystems on a COTS PC with simulation and monitoring below millisecond accuracy, which is sufficient for the needs of many industrial automation systems.

[3]www.rtds.com, www.opal-rt.com, www.dspace.com

VII. Conclusions

The design principles of MARS, namely *real-time operation*, *data stream orientation*, *flexibility*, and *dynamic user interaction* make it a highly flexible, extensible and customizable platform for HIL testing of industrial automation system devices. With sub-millisecond timing accuracy the platform achieves the simulation and monitoring accuracy that is necessary for testing industrial automation systems.

Powered with the flexibility of MARS, users could also push the system into overload or simply expose performance caveats. Future work shall address this issue by providing means to predict the needed memory and computational power for a given scenario, and enable an automatic deployment of a testing scenario onto multiple MRTC instances, potentially distributed onto multiple devices.

References

[1] M. Wang, T. Madhyastha, N. H. Chan, S. Papadimitriou, and C. Faloutsos, "Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic," in *ICDE*, pp. 507–516, 2002.
[2] P. Sarhadi and S. Yousefpour, "State of the art: hardware in the loop modeling and simulation with its applications in design, development and implementation of system and control software," *International Journal of Dynamics and Control*, vol. 3, no. 4, pp. 470–479, 2015.
[3] H. Holm, M. Karresand, A. Vidström, and E. Westring, "A survey of industrial control system testbeds," in *Secure IT Systems*, Springer Science & Business Media, pp. 11–26, 2015.
[4] S. M. Blair, F. Coffele, C. D. Booth, and G. M. Burt, "An open platform for rapid-prototyping protection and control schemes with IEC 61850," *IEEE Trans. on Power Delivery*, vol. 28, no. 2, pp. 1103–1110, 2013.
[5] J. Mo, J. C. Tan, P. A. Crossley, Z. Q. Bo, and A. Klimek, "Evaluation of process bus reliability," in *DPSP*, 2010.
[6] M. Zadeh, T. Sidhu, and A. Klimek, "Implementation and testing of directional comparison bus protection based on IEC 61850 process bus," *IEEE Trans. on Power Delivery*, vol. 26, no. 3, pp. 1530 – 1537, 2011.
[7] Z. Q. Bo, A. Klimek, Y. L. Ren, and J. H. He, "A real time digital simulation system for testing of integrated protection schemes," in *POWERCON and IEEE Power India Conf.*, pp. 1–5, 2008.
[8] M. S. Almas and L. Vanfretti, "Performance evaluation of protection functions for IEC 61850-9-2 process bus using real-time hardware-in-the-loop simulation approach," in *CIRED*, pp. 1–4, 2013.
[9] R. Kuffel, D. Ouellette, and P. Forsyth, "Real time simulation and testing using IEC 61850," in *MEPS*, pp. 1–8, 2010.
[10] D. S. Ouellette, M. D. Desjardine, and P. A. Forsyth, "Using a real time digital simulator to affect the quality of IEC 61850 GOOSE and sampled value data," in *DPSP*, 2010.