Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

A DISTRIBUTED SIMULATOR PLATFORM FOR RAPID INDUSTRIAL USER EXPERIENCE PROTOTYPING

Roberto S. Silva Filho
Alexander K. Carroll

GE Global Research
Intelligent Industrial Experiences Lab
2623 Camino Ramon
San Ramon, CA, USA 94583

James D. Brooks

GE Global Research
Industrial Outcomes Optimization
1 Research Circle
Niskayuna, NY, USA 12309

ABSTRACT

The research and development of novel user experience concepts in well-regulated industrial domains face different challenges. Systems in these domains often require backward compatibility and integration with legacy sub-systems and protocols. They must comply with well-defined procedures and standards, and must pass through stringent evaluation processes involving actual users under realistic conditions and scenarios. As a consequence, prototyping and simulations are extensively used. During product development, the level of fidelity of a simulation prototype will directly impact the quality of end-user feedback, minimizing expensive rework of UX in later stages of a project. This paper describes the Distributed Industrial Simulation Platform (DISP), a simulation framework developed within GE that facilitates the rapid prototyping and evaluation of novel industrial UX systems. We present the DISP design and main services showing how it has been used in support of the development and simulation of two UX prototypes in the railroad transportation domain.

1 INTRODUCTION

Simulations and prototypes are key tools of choice in early stages of industrial product development. In particular, they allow designers to evaluate new user experience (or UX) concepts before they are developed into production systems. The higher the fidelity of prototypes and experiences at this stage, the better the quality of the user feedback collected, and higher are the chances of meeting end-users needs.

The development of high-fidelity simulated user experiences in the industrial domain is non-trivial. The first stage in the development of a new operations concept involves discovery and observational research. Frequently, a co-design workshops are conducted with end-users to identify their typical workflow, pain points and possible technology-driven solutions. At this stage, low-fidelity story boarding and wireframe prototypes are used to explore and evaluate several potential solutions and UX concepts (Kim et al. 2017)(Levulis, Kim, and DeLucia 2016). Soon after this stage, there is a need for higher-fidelity system prototyping to validate and refine these concepts. Prototypes allow actual users to experience the technological solutions in situations close to their everyday work setting. The closer the prototype is to the proposed solution, the better the design feedback on the concept will be. Two main options are generally available in the production of high-fidelity prototypes: the development of full-fledged simulations, or the rapid prototyping of systems out of existing parts.

Whereas the development of simulations using different of-the-shelf simulation packages is an option (Boer, Bruin, and Verbraeck 2006), they provide poor or no support for the rapid prototyping of domain-specific Human-machine Interfaces (or HMI). Instead, they focus on the abstract modeling and simulation of key aspects of the system underlying controls and interaction. Few of them provide mechanisms to facilitate the integration of legacy systems. Finally, there is poor support for end-user data analytics (Poria

et al. 2015), often requiring UX designers to manually gather and analyze multi-modal data from both systems logs and user study sessions (Lahat, Adali, and Jutten 2015).

As a consequence, many UX HMI industrial prototypes opt to design around existing systems, relying on the integration of simulations, with existing subsystems and physical HMI controls. While this improves reuse and provides high-fidelity of HMI controls, it many times comes with high integration costs due to the diversity of protocols and subsystems. In this paper we describe our approach to facilitate the rapid prototyping of UX concepts by discussing the design and implementation of an infrastructure that addresses the main concerns involved in the integration of simulation models with existing industrial subsystems and UI components.

2 DESIGN GOALS

In order to facilitate the rapid prototyping and evaluation of novel UX concepts in industrial domains, we have developed the Distributed Industrial Simulator Platform (DISP) which provides a common component framework and additional services for the development of human- and system-in-the-loop distributed simulations. DISP was developed as part of GE's Global Research next generation railroad program which includes the use of current HMI technology, advanced networking, controls, and collaboration technology in enhancing today's railroad operations.

DISP was designed according to the following architectural decisions: Distributed Component Model; Message-driven communication; Common extensible data model; Multi-platform support; and Service-oriented extensibility. These design decisions were them implemented in Java, in the form of distributed services, reusing the communication capability of ZeroMQ middleware (Hintjens 2013). In this section, we will discuss these decisions.

2.1 Distributed Component Model

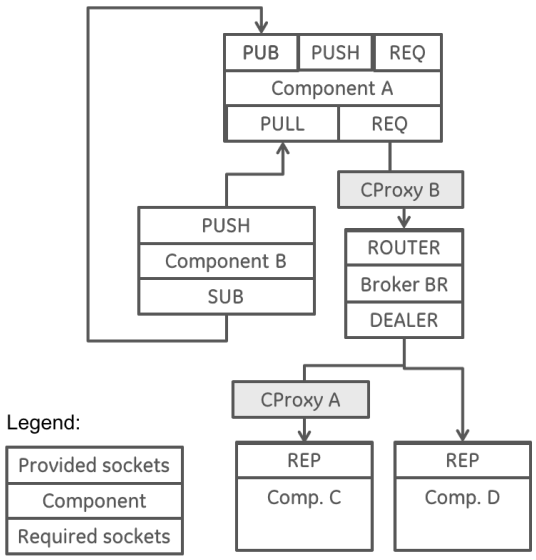


Figure 1 Component model example

Industrial systems such as railroads are typically built as networked systems of systems. This characteristic allows subsystems to be independently developed and maintained. In particular, subsystems can be independently tested and evaluated using a combination of hardware-in-the-loop and software-in-the-loop configurations. DISP component model supports the definition of component interfaces and a message-oriented communication channels between components. This model supports co-simulation scenarios

(Fitzgerald and Pierce 2014) where both continuous and discrete subsystem simulators are integrated using contractual parameters and protocols defined for each pair of components. It also facilitates the integration of actual subsystems, executing in heterogeneous networked hosts.

The DISP component model also supports the notion of proxies (grayed boxes in Figure 1). Proxies are specialized queues that modulate the communication between components, allowing the simulation of different reliability and throughput characteristics. Proxies will be discussed in more detail in section 3.5 .

2.2 Message-driven Communication

The communication between DISP simulator components is performed using networked message queues that support both unicast and multicast communication including: request-response, producer/consumer and publish/subscribe communication styles. In particular, we use ZeroMQ (Hintjens 2013) as our messaging middleware. ZeroMQ sockets implementing different types of queues including publish/subscribe, request/response and other queue variations. Components communicate via complementary pairs of sockets as illustrated in the example of Figure 1.

The use of distributed message queues as component connectors decouples the many subsystems involved in complex simulations, allows for multiple producers and consumers, and enables the dynamic exchange and restart of components with minimum impact on the overall simulation system. From a system design and implementation, it also allows for flexible deployment configurations, and facilitates both reuse and refactoring. Finally, queues facilitate the simulation of different network communication characteristics via proxies as will be later discussed in section 3.

2.3 Common Extensible Data Model

In DISP, both data and control messages exchanged by the different components of the system are represented as JavaScript Object Notation (or JSON) objects. This uniform data representation allows different application-specific message types to be used, and permits the implementation of different protocols. JSON is easily serializable as character streams which facilitate the integration of heterogeneous legacy systems via protocol wrappers and adapters that convert native messages and data to and from JSON messages.

2.4 Multi-platform

By combining a common JSON message & data representation over ZeroMQ message queues, DISP facilitates the integration of components implemented in different programing languages and legacy environemnts. Currently, ZeroMQ is supported in over 50 different programming languages and platforms including: Python, Java, PHP, Ruby, C, C++, C#, Erlang, Perl, and others. JSON message processing is also widely supported in these environments, which facilitate the conversion of messages to and from native data types.

2.5 Service-Oriented Extensibility

DISP comes with different auxiliary services that facilitate the development of distributed component-based simulations including: directory, orchestration, data gathering, time synchronization, communications (or comms), . While the use of these services is optional, they provide “add-on” solutions to common simulation and distributed systems problems, and facilitate the development of new simulator experiences.

3 DISP SIMULATION SERVICES

DISP component model is augmented with services that facilitate the implementation of complex distributed simulators. DISP main services are implemented in Java using the Java binding of ZeroMQ.

3.1 Directory Service

In distributed architectures such as DISP, there is a need for components to advertise their presence, locate, and bind to one another before and during simulations. As the simulator complexity increases, the management of component configurations becomes important. Changes in network topology, host names, and number of components need to be supported. A scalable approach to this problem is to use a directory service (Coulouris et al. 2011).

The DISP directory service (DS) provides a logically centralized database of component meta-data. It allows components to create, read, update and delete DS entries, allowing components to find one another at runtime. DS database entries specify the component name, list of provided and required sockets and other information such as the host where the component is located.

Another common problem in distributed systems is the dissemination of information such as the arrival and departure of components. The DISP DS provides a subscriber API that allows interested components to be notified when entries are created or modified.

3.2 Launcher

The launcher service automates the bootstrapping of complex distributed simulations using architecture description files, which provide information similar as that stored in the directory service. The simulation launcher starts individual components of the simulator, according to their dependency graph, registering them with the directory service. In order to support automatic activation of components and services, each host of the distributed simulation system runs a copy of the simulation launcher. A master simulation launcher coordinates with all launchers in the system to automatically start local installations of components as instructed by the master launcher. The launcher may be configured to reuse components that are already loaded and registered in the DS, starting only components that are not currently present in the system. Once loaded, simulations are started using the orchestration service described later in this section.

3.3 Orchestration Service

A common problem in distributed simulations is the need for a central control, where the components of the system can be jointly started, stopped, paused, and resumed. The orchestration service (or OS) provides a logically centralized service that receives commands from a manned simulation console, or automatic components as the Launcher, and broadcasts these commands to all components of the system at once. It does so by providing a PUB port that can be subscribed by system components.

3.4 Simulation Console

The simulation console provides a simple user interface for users to interact with the orchestration, directory, analytics and other services of the system. It allows facilitators controlling the simulation to perform basic operations such as sending start/stop commands, detecting the arrival/departure of components, or launching new simulations. The console also provides an interface to inspect and interact with the DS, which has proven very useful for developing and debugging of these distributed systems.

3.5 Comms Service

A common problem in human-in-the-loop distributed simulations is the need to reproduce less than perfect real-world network characteristics found in many industrial sites, e.g. power plants, rail roads, ships, factories, etc. The communications service (also referred to as Comms Service) supports the simulation of degraded network characteristics, using existing local networks. It does so through the use of proxies that intermediate the communication between components in the system. Proxies can be automatically or manually created and installed between communicating components. They allow the reproduction of different reliability, latency and data throughput conditions allowing, for example, the simulation of satellite

link, 3G and 4G cellular networks conditions, and various types of jitter and bandwidth limitations found in industrial networks. Proxies also provide APIs that allow the dynamic tuning of communication parameters at runtime. For example, consider the simulation of a Locomotive Cab experience as it moves through a transportation network. As the train leaves the city and moves through rural areas, its connectivity to the controls room may change from using a urban cellphone network to using high-latency satellite link.

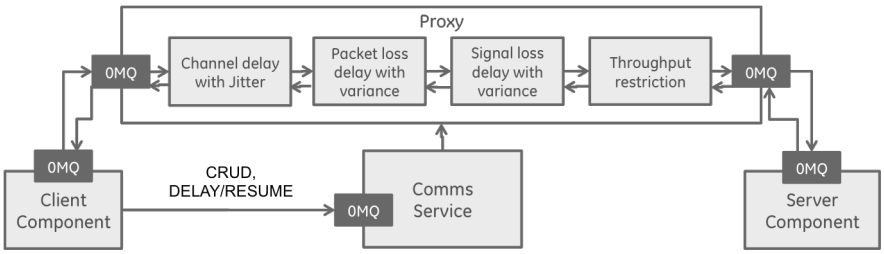


Figure 2 General Pipeline of a Comms Proxy

As shown in Figure 2, proxies are programmable. Messages pass through a pipeline of modulators that operate according to parameters that may change over time. Basic configuration parameters include: jitter, delay, signal loss probability and throughput. Messages sent and received by each component are delayed based on these parameters. Note that both packet and signal lose conditions are all simulated as different message delays.

3.6 Data Capturing & Analytics Service

In any simulation, the monitoring of the system & users behavior provide invaluable information for the calculation of different usability and performance metrics and the ultimate validation of end-user experience hypothesis. For example, did the users' performance improve with the new HMI? Did the users react to the alarms within expected time limits? In order to answer questions such as these, different modalities of logs are produced. E.g. system-level, UI-level, as well as audio and video recording of end-users' interaction with the HMI. The amount data produced in a typical end-user study can quickly become overwhelming, making its analysis tedious, time consuming, and error prone.

In order to facilitate the multi-modal data capture and analysis of simulation sessions, we have developed the DISP data recording & analysis service. This service provides a single API for capturing, aggregating and correlating multiple data and log streams. The single point of entry allows these streams to be synchronized with respect to real-time and simulation meta-data and timeframes. At the core of this service is a complex event processing and inference engine that allows the detection of successful and unsuccessful sequences of events involving multiple log streams. In this implementation, we used JBoss Drools (http://www.drools.org) as the inference engine of the system. For example, by correlating events for the UI and the system components, it is possible to detect when users switch tasks, get stuck on certain parts of the UI, switch attention, system response time, and many other KPIs.

3.7 Time Service

When designing the DISP framework, we were pragmatic with respect to timing. We synchronize system components with respect to a central time reference (the time service), and adopt conventions such as requiring time to be represented as epoch numbers throughout the simulation. Similar to the orchestration service, the time service provides a PUB socket allowing components of the system to subscribe to clock ticks and commands including fast forward and fast backwards, ticks by a multiplication factor, move to a certain time, pause and resume.

3.8 Collaboration services

While most simulation data and control messages can be transmitted using message queues, voice and video communication usually requires real-time guarantees not supported by these channels. The collaboration service supports SIP signaling and multi-media traffic control in scenarios that utilize video and audio communication between participants, and allows the recording of these interactions for offline analysis.

4 CASE STUDIES

We have used DISP framework for rapid prototyping and evaluation of two UX concept prototypes.

4.1 Next Generation Train Dispatching Simulator

Current freight train operation involves the continuous communication between the locomotive engineer, who pilots the train, with different train dispatchers, that control the inbound and outbound traffic of trains in different territories along the rail network. This communication is currently performed mainly using radio, and is supported by train location data gathered from wayside systems such as Positive Train Control (PTC). In this project, we implemented and tested different UX design concepts for the next generation control center HMI. The goal was to improve situation awareness of remote dispatchers when talking to individual train engineers. In particular, we designed a dashboard where track, individual train conditions, weather and alerts are shown in context. During the simulated scenarios using the new UI, different hypothesis were tested. For example, the ability of remote dispatchers in understanding the current train situation during different simulated situations as measured by analyzing radio communication content and task performance.

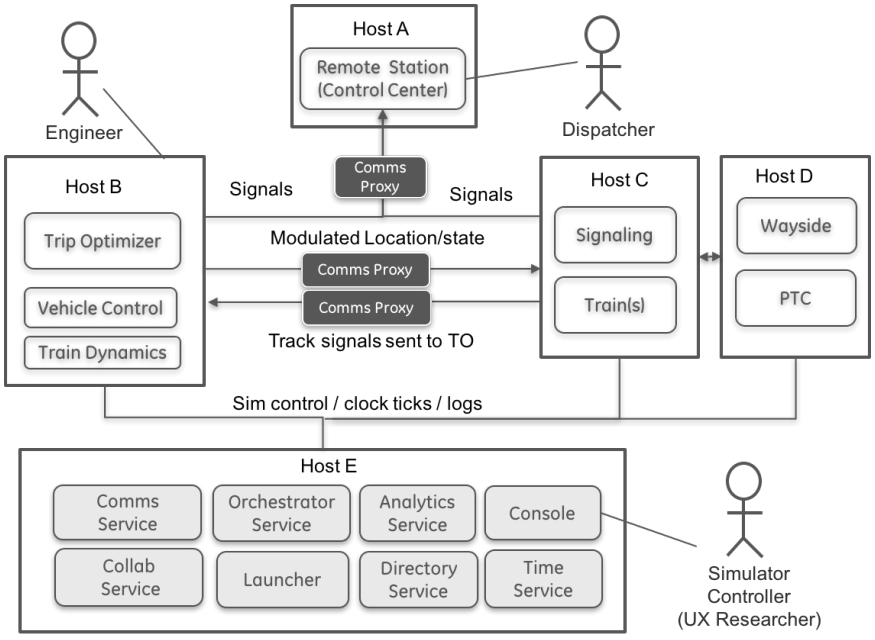


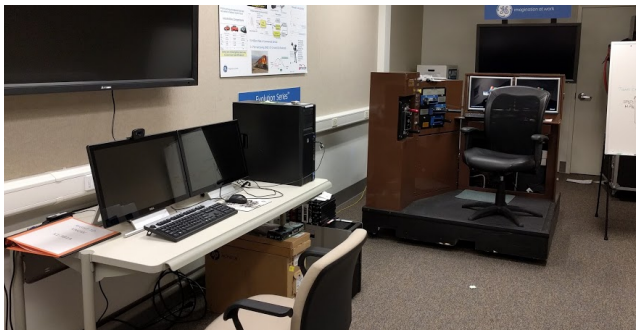
Figure 3 Railroad network simulator architecture with engineer, remote dispatcher UX researcher

This study required the simulation of both the locomotive cab experience and the railroad control center HMI as seen in Figure 4. The locomotive cab experience was developed combining physical controls typically found in trains, with touch glass UI enhanced with advanced communication controls. In particular, we reused GE's Trip Optimizer speed control system (or TO) (Houpt et al. 2009), a complex sys-

tem-in-the-loop component that can automatically control the operation of locomotives based on different track and trip parameters. TO interfaces with different on-board sensors and wayside systems such as PTC. PTC provides automatic speed limits and stops the train in case of movement violations in existing railroads, including potential train-to-train collisions.

A diagram with the integration of these systems is shown in Figure 3. We use solid rectangles to represent different hosts in a network, and rounded rectangles to represent main DISP-enabled software components. Notice the use of communications proxies between the components. In our study, proxies execute in the same host as the DISP services (host E), but are sown outside that box to emphasize their role as communication interceptors. Proxies are used to mimic real-world communication delays in transmitting train telemetry and position to the remote operations room. We use an Anylogic-based simulator to represent the rail network signaling (tracks, switches) and the traffic (other trains). Integration with track wayside systems and PTC was also supported. They enhance the rail network simulator with the ability to detect trains speed and position, and to send signals to the simulated train in case of excess of speed (PTC).

(a) New in-cab HMI



(b) Operations Room (Left), and In-cab experience (Right)

Figure 4 Remote operations room & new in-cabin HMI experiences

Several DISP services were used. The Directory Service is used by the system components to locate one another. Orchestration Service is used to coordinate the start, pause, stop of simulation via the simulator controller console. The Time Service provides the common simulation time, and the Comms Service provides audio communication between the engineer in the cabin and the remote operator.

In this setting, different UI components are integrated. Two workstations are used as shown in the right side of Figure 4 : the first simulates the remote operations room experience (left) and the second one (right) provides the locomotive cab experience. The rail network simulator and simulation services execute in their own hosts. Note that for the cab experience we integrate physical controls with a UI developed to provide train gauges and communication capability. Also note that, when performing tests with end-users, both the cab and the remote operations room are set up in separate locations.

The glass cab experience (left side of Figure 4) and the remote operations room use a custom UI developed for this project. This integrated system is heterogeneous in nature. The user interface is developed using a combination of HTML5 and .NET technology, the rail network simulator is implemented in Java (Anylogic), the Trip Optimizer (or TO) system is developed in C and Matlab, and the simulation services are implemented in Java.

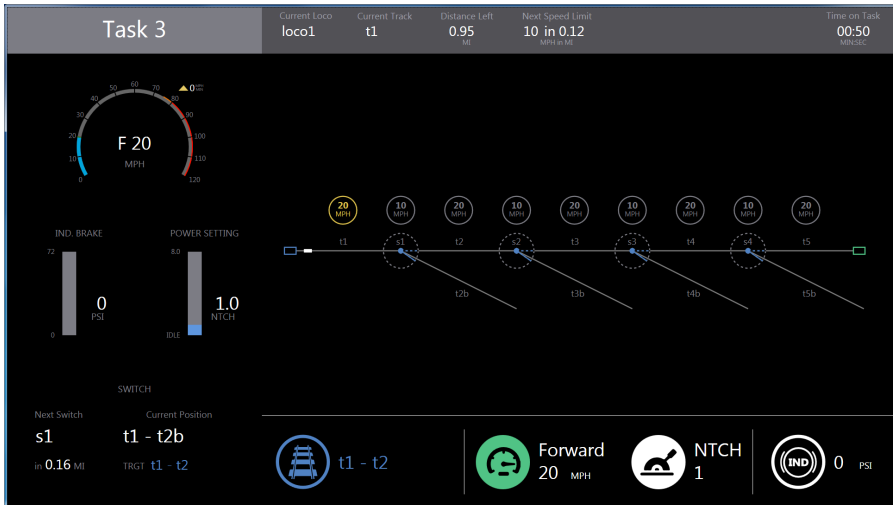
4.2 Yard Remote Control Simulator

In this second use case, our goal was to study the efficacy of an updated version of existing remote control system for locomotives in a yard. The new HMI provides real-time information about train, including its position, power settings, and track speed limits as shown in Figure 5. We also tested alternate forms of control input including touch screens and physical control units (existing and new). The combination of operator UI and control unit allows the operator to simulate remotely driving the train in the yard.

Two control options:

a) New touch-based control

b) Existing physical control



Operator View

Figure 5 Next generation yard remote control UX

The prototype integrates different components: the yard simulator with different tracks and locomotives (including simulation of the physics and control system behavior), the yard visualization, that allows the operator to see the current train speed, position, and controls state (throttle and brakes), and the DISP framework services (Comms, Console, Launcher, Directory, Time and orchestration services). The remote control and simulator are interconnected through proxies that restrict traffic using the latency of the local network of the yard. These are illustrated in Figure 6.

As with the train dispatching simulator, the architecture of the system is distributed. Three computers are used in the simulation including one host for the rail network simulator, another mobile device (tablet) for the yard pilot UI, and a third one for the simulator services. They are integrated through a local area network.

5 DISCUSSION

Both prototypes presented in this paper were developed using three distributed teams: the UI development team located in Mexico, the DISP framework team located in California, and the simulation and controls team located in New York state. During the development of the prototypes we observed the following benefits and costs of the use of DISP framework.

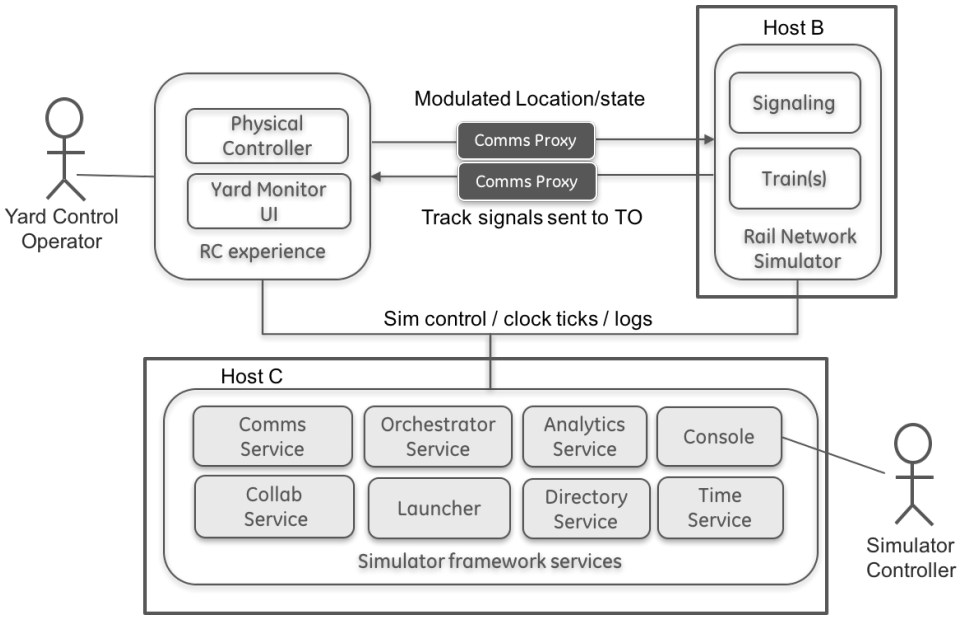


Figure 6 Yard Pilot Architecture

5.1 Benefits

Independent development. By architecting the system in terms of independent components (UI, Track & Train Simulators, and Services) with well-defined interfaces (message contracts, data format and protocols), the distributed teams were able to successfully test their individual components in semi-isolation. Temporary stubbed systems were used in early development stages, later on replaced with actual components with minimum rework.

Distribution flexibility was another observed benefit. When necessary, individual components could be deployed together in the same host (e.g. during testing), or in servers available to all users during development. This facilitated the integration tests between developers, and solved past problems related to licensing and integration of off-the-shelf simulators such as Anylogic, that must execute in a single Windows host shared by our team.

Reuse. The componentization and separation of concerns into a set of networked components improved the reuse of these different sub-systems. By keeping messages and interfaces constant, minimal adaptation was required, for example, to transition one network simulator from one case study to the next.

Interoperability. The use of ZeroMQ queues, which supports different programming platforms, together with the adoption of common message formats largely facilitated the integration of these systems. We experienced minimal software development issues related to message parsing and communication.

Performance. The use of distributed ZeroMQ sockets had little impact in the performance of the system. During development, we performed several tests using geographically distributed components, with minimal delays in communication.

Simulation Services facilitated distributed development. Services such as the simulation console, Orchestration, Time and Analytics address common problems in distributed simulations. As such, they were widely used, and have reduced the burden of individual component developers, simplifying their code. For example, an early implementation of the rail network simulator suffered from different synchronization problems. Trains were required to wait for simulation state changes before publishing their position to the track. Startup order was a problem for later trains. Reset of simulator required the restart of components individually, and logs were scattered over different hosts. With the use of a central orchestrator service, log, and a directory services, the components can now find one another, and produce synchro-

nized logs. They all now wait for the start/stop simulation common console signals. As a result, the railroad simulation code was largely simplified.

Overall network simulation was simplified. The use of proxies for communication between components provide an elegant and modular way for simulating different network conditions. The implementation of this concerns in the middleware removed the network traffic concern from the UI and simulator developers largely simplifying the code.

Faster data analytics. Finally, the ability to aggregate and analyze logs, detecting abnormal sequences of events significantly reduced the time devoted to data analysis from weeks to days. By analyzing test run data during final integration, we were able to improve the quality of the data captured, as well as identify internal simulator and control interaction issues.

5.2 Costs & limitations

While DISP component model presented the benefits above, we observed some new problems induced by this distributed simulation model as follows.

Global system state. The need for a common view of the global state of the system can hinder the adoption of a distributed model, requiring careful design and planning. A balance between gathering information from individual components and having it centralized must be achieved. E.g. after different interactions, we adopted a hub-spokes topology where the track simulator communicates directly with train and PTC components. The remote operations center UI then gets aggregated information about track and trains from the simulator component. i.e. the simulator provides the ground truth for the state of the simulation. This prevented data replication and simplified the development of the UI.

Inversion of Control. As with the data, distributed control is not easily understood by simulator and UI designers. While the orchestration service simplified the problem of synchronizing the start and stop of simulation, it required a change of mindset in the developers, as well as some adaptation from current components. For example, the simulator component had to be adapted to accept control messages form the orchestration service, instead of relying on its own simulation control panel. While this required additional development effort, the effort paid off in the second prototype's development due to the simplified implementation in the cleaner architecture.

Integration costs. Reused components such as the Trip Optimizer had to be adapted in order to produce and consume messages through ZeroMQ. In particular, they had to be wrapped using a multithreading interfaces, supporting multiple subscribers, which required additional development in different environments like C and Matlab (used by the Trip Optimizer). This adaptation had minimal impact on the core component code, limiting the impact and costs of integration.

Overall, however, we found the benefits of the approach significantly outweighed its costs, in particular during situations where component reuse across different simulations are required, or where the redevelopment of components would have been time or cost prohibitive.

6 RELATED WORK

Different distributed simulation approaches exist in the literature (Boer, Bruin, and Verbraeck 2006), however, they did not meet the specific requirements of interactive system- and human-in-the-loop simulation systems such as the ones discussed in this paper.

For example, an architecture for modular distributed simulation is presented by (Scerri et al. 2010). It is concerned with synchronization of state on parallel computation systems used for high-performance simulations. Our approach focus on the development of high-fidelity simulators for industrial applications where simulated components are mixed with existing industrial systems. In particular, we are interested in the simulation and analysis of new user experiences in these domains.

ROS (Quigley et al. 2009) as an example of event-driven middleware for integration of components in robotics systems. Our approach has similarities with ROS but support a richer message model based on

ZeroMQ queues, together with additional services such as logging analytics, comms, orchestration, and timing that are tailored to the needs of distributed interactive simulation systems.

Platforms for hardware in the loop simulations have also been proposed (Ingalalli, Satheesh, and Kande 2016). To the best of our knowledge, these are mainly low-level electric circuit simulators that test the integration of components designed in software with physical hardware components, and do not support the interactive simulation needs of systems-of-systems.

The usability monitoring capability of the log service has similarities with EDEM (Expectation-Driven Event Monitoring)(Hilbert and Redmiles 1998). We build upon the EDEM approach by adding multi modal support.

For the analytics service data analysis UI, we adopted some of the multimodal data analysis concepts of systems such as ChronoViz (Fouse et al. 2011) and (Weibel et al. 2013). Our system, however, focus on the specific needs of simulation systems, with particular focus on analysis of UI and simulation logs.

The IEEE Distributed Interactive Simulation standard and the more recent HLA (High-Level Architecture Simulator) (IEEE 2016) prescribe a set of protocols and data formats for building distributed simulators in different domains that if implemented would enable the interoperability of simulator components. In spite of its usefulness, we find these one-size-fits-all approaches to be overly complex, in particular, there is a lack of reference implementations of these standards we could readily use. As a result, we adopted a more pragmatic and specific approach based on high-level component models that builds upon a very capable message middleware.

7 CONCLUSIONS

In this paper, we described the DISP, a common framework for the development of high-fidelity human- and system-in-the-loop distributed simulations. DISP integrates a component model and a set of services that address common problems in the implementations of these simulated experiences. We demonstrate our approach through two use cases where DISP facilitated the development of simulations in the transportation domain.

The relative simplicity of the model, based on message queues, standardized message representations and communication proxies, together with the reuse achieved by componentizing existing systems have helped our teams to reduce the development effort of distributed simulations.

ACKNOWLEDGEMENTS

We thank all the members of our team for their contributions to this project, in particular: Hullas Sehgal, Uriel Osequera, Shanshan Wang, So Young Kim, Neeraja Subrahmaniyan, Aristotelis Thanos, and Maria Manrique .

REFERENCES

Boer, C. A., A. D. Bruin, and A. Verbraeck. 2006. “Distributed Simulation in Industry - A Survey Part 2 - Experts on Distributed Simulation.” In *Proc. of the 2006 Winter Simulation Conference*, 1061–68.

Coulouris, George, Jean Dollimore, Tim Kindberg, and Gordon Blair. 2011. *Distributed Systems: Concepts and Design*. 5 edition. Boston: Pearson.

Fitzgerald, John, and Kenneth Pierce. 2014. “Co-Modelling and Co-Simulation in Embedded Systems Design,” 15–25.

Fouse, Adam, Nadir Weibel, Edwin Hutchins, and James D. Hollan. 2011. “ChronoViz: A System for Supporting Navigation of Time-Coded Data.” In *CHI ’11 Extended Abstracts on Human Factors in Computing Systems*, 299–304. CHI EA ’11. New York, NY, USA: ACM.

Hilbert, David M., and David F. Redmiles. 1998. “An Approach to Large-Scale Collection of Application Usage Data over the Internet.” In *Proc of the 20th Intl Conf on Software Engineering*, 136–145. ICSE ’98. Washington, DC, USA: IEEE Computer Society.

Hintjens, Pieter. 2013. *ZeroMQ: Messaging for Many Applications*. 1st ed. Sebastopol, CA: O’Reilly Media.

Houpt, P. K., P. G. Bonanni, D. S. Chan, R. S. Chandra, K. Kalyanam, J. D. Brooks, and C. McNally. 2009. "Optimal Control of Heavy-Haul Freight Trains to Save Fuel." In *9th International Heavy Haul Association Conference*, 1033–40. China.

IEEE. 2016. "IEEE SA - 1516-2010 - IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-- Framework and Rules." Accessed November 1. https://standards.ieee.org/findstds/standard/1516-2010.html.

Ingalalli, A., H. Satheesh, and M. Kande. 2016. "Platform for Hardware In Loop Simulation." In *2016 International Symposium on Power Electronics, Electrical Drives, Automation and Motion*, 41–46.

Kim, So Young, Jennifer Cooper, Alexander Carroll, and Sundar Murugappan. 2017. "Purple Sky Framework Towards the Flight Deck of the Future Experience: Through Co-Design, Rapid UX Prototyping, and User Testing." In *Advances in Human Aspects of Transportation*, edited by Neville A. Stanton, Steven Landry, Giuseppe Di Bucchianico, and Andrea Vallicelli, 484:851–62. Cham: Springer International Publishing.

Lahat, D., T. Adali, and C. Jutten. 2015. "Multimodal Data Fusion: An Overview of Methods, Challenges, and Prospects." *Proceedings of the IEEE* 103 (9): 1449–77.

Levulis, Samuel J., So Young Kim, and Patricia R. DeLucia. 2016. "Effects of Touch, Voice, and Multimodal Input on Multiple-UAV Monitoring During Simulated Manned-Unmanned Teaming in a Military Helicopter." in *Proc. Human Factors and Ergonomics Society Annual Meeting* 60 (1): 132–132.

Michel, Fabien, Jacques Ferber, and Alexis Drogoul. 2009. *Multi-Agent Systems and Simulation: A Survey From the Agents Community's Perspective*. CRC Press - Taylor & Francis.

Poria, Soujanya, Erik Cambria, Amir Hussain, and Guang-Bin Huang. 2015. "Towards an Intelligent Framework for Multimodal Affective Data Analysis." *Neural Networks: The Official Journal of the International Neural Network Society* 63 (March): 104–16.

Quigley, Morgan, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. 2009. "ROS: An Open-Source Robot Operating System." In *ICRA Workshop on Open Source Software*, 3:5. https://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf.

Scerri, David, Alexis Drogoul, Sarah Hickmott, and Lin Padgham. 2010. "An Architecture for Modular Distributed Simulation with Agent-Based Models." In *Proc of the 9th Intl Conf on Autonomous Agents and Multiagent Systems: Volume 1*, 541–548. AAMAS '10. Richland, SC: Intl Foundation for Autonomous Agents and Multiagent Systems.

Weibel, Nadir, Shazia Ashfaq, Alan Calvitti, James D. Hollan, and Zia Agha. 2013. "Multimodal Data Analysis and Visualization to Study the Usage of Electronic Health Records." In *Proc of the 7th Intl Conf on Pervasive Computing Technologies for Healthcare*, 282–283. PervasiveHealth '13. ICST, Brussels, Belgium, Belgium: ICST (Inst. for Comp. Sci, Social-Inf and Telecom. Eng).

AUTHOR BIOGRAPHIES

ROBERTO S. SILVA FILHO is a Lead Scientist at GE Global Research in San Ramon, CA. He holds a Ph.D. in Information and Computer Sciences from University of California, Irvine. His research interests include software engineering and middleware, mobile computing, HCI, and their industrial applications. His e-mail address is silva_filho@ge.com.

ALEXANDER K. CARROLL is a Senior Scientist at GE Global Research in San Ramon, CA. He holds a MS in Computer Science and Engineering from the University of Washington. His research interests include HCI, distributed systems architectures, and HMI in industrial contexts. His e-mail address is alex.carroll@ge.com.

JAMES D. BROOKS is a Lead Scientist at GE Global Research in Niskayuna, NY. He holds a Ph.D. in Decision Sciences and Engineering Systems from Rensselaer Polytechnic Institute. His research interests include human-automation interaction and the design of large-scale distributed control systems. His e-mail address is brooksja@ge.com