



ARTICLE

Distributed Simulation Platforms and Data Passing Tools for Natural Hazards Engineering: Reviews, Limitations, and Recommendations

Lichao Xu¹ · Szu-Yun Lin¹ · Andrew W. Hlynka¹ · Hao Lu² · Vineet R. Kamat¹ · Carol C. Menassa¹ · Sherif El-Tawil¹ · Atul Prakash² · Seymour M. J. Spence¹ · Jason McCormick¹

Accepted: 7 June 2021
© The Author(s) 2021

Abstract There has been a strong need for simulation environments that are capable of modeling deep interdependencies between complex systems encountered during natural hazards, such as the interactions and coupled effects between civil infrastructure systems response, human behavior, and social policies, for improved community resilience. Coupling such complex components with an integrated simulation requires continuous data exchange between different simulators simulating separate models during the entire simulation process. This can be implemented by means of distributed simulation platforms or data passing tools. In order to provide a systematic reference for simulation tool choice and facilitating the development of compatible distributed simulators for deep interdependent study in the context of natural hazards, this article focuses on generic tools suitable for integration of simulators from different fields but not the platforms that are mainly used in some specific fields. With this aim, the article provides a comprehensive review of the most commonly used generic distributed simulation platforms (Distributed Interactive Simulation (DIS), High Level Architecture (HLA), Test and Training Enabling Architecture (TENA), and Distributed Data Services (DDS)) and data passing tools (Robot Operation System (ROS) and Lightweight Communication and Marshalling (LCM)) and compares their advantages and disadvantages. Three specific limitations in existing platforms are identified from

the perspective of natural hazard simulation. For mitigating the identified limitations, two platform design recommendations are provided, namely message exchange wrappers and hybrid communication, to help improve data passing capabilities in existing solutions and provide some guidance for the design of a new domain-specific distributed simulation framework.

Keywords Civil infrastructure · Data passing tools · Distributed simulation platforms · Hybrid communication · Message exchange wrapper · Natural hazards

1 Introduction

As in many other fields, numerical simulation models in the natural hazards research area have primarily evolved along separate disciplines. For example, in earthquake engineering, several models have been developed to simulate various effects of an earthquake on civil infrastructure (Xiong et al. 2016). An example in the fire propagation area is National Institute of Standards and Technology (NIST)'s Fire Dynamics Simulator (FDS) and Smokeview (Kerber and Milke 2007). In hurricane research, there exist some public models such as the Florida Public Hurricane Loss Model (FPHLM) (Chen et al. 2009) and commercial models from AIR Worldwide (AIR) (AIR Worldwide 2020), Applied Research Associates (ARA) (ARA 2021), and Risk Management Solutions (RMS) (Risk Management Solutions 2007). Similarly, several models exist in hazard-related areas such as wind (Lin et al. 2018), tsunami (Zobel et al. 2006), flood (Ginting and Mundani 2019), power system (Wang et al. 2015), transportation (Barrett et al. 2010), human response under disasters (Jain and

✉ Vineet R. Kamat
vkamat@umich.edu

¹ Department of Civil and Environmental Engineering,
University of Michigan, Ann Arbor, MI 48109, USA

² Department of Electrical Engineering and Computer Science,
University of Michigan, Ann Arbor, MI 48109, USA

McLean 2003; Bunea et al. 2016), and to a lesser extent, evacuation plans (Xie et al. 2016), emergency response training (Liu et al. 2007), and post-disaster recovery (Fiedrich 2006).

However, extreme natural hazards, such as earthquakes, tornadoes, floods, and hurricanes, often induce complicated interdependencies between the built environment (for example, buildings and bridges), critical infrastructure systems (for example, roadways and communications), and social and non-physical systems (for example, politics and economics). As such, hazards simulation and disaster science, more broadly, are highly multi-disciplinary research areas. Many U.S. government documents (NRC 2011a, 2011b; NIST 2016a, 2016b) and researchers (Koliou et al. 2017; Mitsova 2018) have called for the development of comprehensive frameworks that can integrate the efforts from different sub-fields and enhance interdisciplinary collaborations between natural hazard researchers.

In order to deal with this lack of compatibility, one promising and practical strategy is to modularize each discipline-specific computational model and then integrate them with distributed simulation platforms such as Distributed Interactive Simulation (DIS) (D.S. Committee 2012), High Level Architecture (HLA) (HLA Working Group 2010), Test and Training Enabling Architecture (TENA) (Powell and Noseworthy 2012), and Distributed Data Services (DDS) (OMG 2015), or data passing tools such as Robot Operation System (ROS) (ROS.org 2018) and Lightweight Communication and Marshalling (LCM) (Huang et al. 2010). Addressing this problem using such an approach is commonly referred to as distributed simulation.

The current state of affairs in this field is that since each domain has been evolving separately, most of the existing integrated simulations are developed upon and limited to domain-specific simulation environments and lack the benefits of interoperability, reusability, and scalability provided by the generic simulation platforms listed above. For example, in earthquake engineering, Integrated Earthquake Simulator (IES) (Hori and Ichimura 2008; Hori 2011) was originally developed to seamlessly integrate analysis models and simultaneously analyze almost all processes involved in earthquake disasters in Japan. However, even for a similar simulation, a new version of IES had to be developed separately for the Istanbul, Turkey earthquake due to differences in numerical analysis methods and available urban information (Sahin et al. 2016). Moreover, IES is sequential and thus inconvenient to be integrated with other simulators with different simulation resolutions such as dynamic debris and transportation systems for timestep-wise coupling simulation (Sahin et al. 2016). Similarly, Miles and Chang conducted simulations to study the interactions that occur between various entities

during disasters (Miles and Chang 2006). However, the proposed approaches do not support cascading or parallel disaster events. In addition, without using distributed simulation, these simulators need to run on a single machine with limited processing power, which usually limits the scale of the problem that can be simulated.

Some researchers have realized the necessities and benefits of the distributed simulation platforms or tools and started to use them in the hazards engineering field. For example, Mandiak et al. developed a disaster monitoring interface and integrated it into an HLA-based earthquake simulation for post-disaster data fusion (Mandiak et al. 2005). Fiedrich proposed a distributed simulation system based on HLA that focused on resource management issues during disasters (Fiedrich 2006). To improve people's emergency response, Liu et al. demonstrated an emergency training simulation achieved by HLA (Liu et al. 2007). Nan and Eusgeld developed an HLA-compliant simulation testbed and demonstrated that HLA is a viable option to simulate and capture interdependencies among simulators (Nan and Eusgeld 2011). More recently, Lin et al. proposed to model interdependent effects in natural hazards and implemented an example application in wind engineering (Lin et al. 2018).

Due to the limitations in distributed simulation platforms, the nontrivial gaps between the simulation tools and domain knowledge, as well as the difficulty of handling multiple disciplines, most models of disaster scenarios have focused on the interactions that occur between two or, at most, three related systems. During disasters, there are usually more factors that interact with each other. In order to facilitate the development of compatible domain simulators and the large-scale simulation incorporating deep interdependencies between multiple simulators, this article surveys the main existing generic distributed simulation platforms (DIS, HLA, TENA, and DDS) and data passing tools (ROS and LCM) that are critical for interdependent study in natural hazards engineering.

More broadly, these simulation tools can also benefit various simulations in civil engineering (Kamat and Martinez 2002; Azar and Menassa 2010; Dong and Kamat 2010) by expanding the simulation scale and increasing simulation resolution. By jointly using different sources of models from researchers, model vendors, or individual developers, they can also provide more detailed and additional types of information compared to the loss modeling framework from the OASIS team (Team Oasis 2021) or the Karen Clark & Company (2020) that is largely driven from loss estimation requirements for insurance businesses.

The strengths and weaknesses of each representative simulation tool are identified to guide researchers or simulation engineers to choose the appropriate tools for their specific applications while being aware of the limitations.

After the systematic review of the distributed simulation tools, the key limitations to the current existing distributed simulation tools are summarized to highlight the specific needs in natural hazards engineering. Finally, based on a synthesis of the gathered information, two platform design recommendations are provided, namely message exchange wrappers and hybrid communication, to help further improve data passing capabilities in existing solutions and provide some guidance for the design of a new simulation framework.

2 Existing Distributed Simulation Platforms

Distributed computing emerged about 40 years ago when the U.S. Department of Defense (DoD) started developing communication protocols to enable interactive simulations involving various types of weapon systems. Among the distributed simulation platforms developed were DIS (D.S. Committee 2012), HLA (HLA Working Group 2010), and TENA (Powell and Noseworthy 2012). Besides military training and simulation, they have also been utilized in marine simulators (Yong and Jin 2000), space projects (Arguello and Miró 2000), infrastructure system simulation (Grogan and De Weck 2015), and virtual testing (Dai et al. 2011). Independently driven by the challenges of conducting real-time sensing, information fusion, and control in robots, researchers in robotics engineering developed low-latency data passing solutions. For example, ROS (ROS.org 2018) and LCM (Huang et al. 2010) have been developed and widely used in real-time robotics applications. Due to their ease of use and high efficiency, researchers have started exploring their applications in distributed simulations for modeling coupling interactions between building energy consumption and human comfort (Thomas et al. 2017) and interdependent effects in natural hazards (Lin et al. 2018; Lin et al. 2019).

In recent years, due to the rising interest in the extension of Internet connectivity, many solutions have been proposed to address the emerging need for Internet-of-Things (IoT) applications. Among such work, IoTivity (IoTivity 2018), which uses a constrained application protocol (CoAP) as its software protocol, is mainly focused on device-to-device connection. DDS (OMG 2015) is a more general data communication protocol and standard developed by the Object Management Group (OMG), which is suitable for all kinds of connections in IoT applications. Even though DDS was developed for real-time operations, it provides features such as API Standard, Data Modeling Standard, Quality of Service, and Time Management, which are comparable to HLA, and thus also suitable for simulations. The remainder of this section reviews the two categories of data passing tools for distributed simulation:

standards and standard-based solutions and standalone tools.

2.1 Standards and Standard-Based Solutions

This section reviews the standards and standard-based solutions, including DIS, HLA, TENA, and DDS.

2.1.1 Distributed Interactive Simulation (DIS)

The early efforts of the U.S. defense community to address the need for networked multi-user simulation led to the SIMNET (Simulation Networking) project (Miller and Thorpe 1995). For about a decade, SIMNET formed the technological foundation for many of its descendants and was the origin of a sequence of IEEE standards. One of SIMNET's derivatives, the DIS protocol, was published as an industry standard from 1993 to 1998 by IEEE (DIS Steering Committee 1998). The standard was considered dominant until a new standard (IEEE std 1278.1-2012 (D.S. Committee 2012)) was released. As related research in hazards engineering, it was planned to be used in a future version of evacuation simulation in fire disasters (Ren et al. 2007). For legacy reasons, DIS is still used today in some modern simulations.

The DIS protocol is designed to be a message passing standard (not an existing software or package) that specifies message types and the procedures to transmit the messages across a network of different simulators. If it is followed correctly, compliant simulations are capable of sending and receiving messages to and from any other compliant simulation, even if the local DIS implementations that run on different hosts are diverse. More specifically, DIS adopts a communication pattern for message exchange with point-to-point communication via User Datagram Protocol (UDP) as shown in Fig. 1. The message format is well specified and referred to as protocol data unit

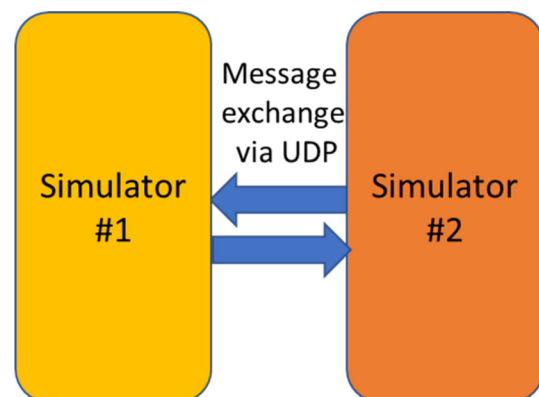


Fig. 1 Point-to-point message exchange via user datagram protocol (UDP) in distributed interactive simulation (DIS)

(PDU), which consists of an entity ID, entity type, and any expected values a simulation requires to function, represented in binary format. The standard defines exactly what variables can be present. Values like “position,” “orientation,” and “collision” all take a certain number of bits and have pre-defined limits to the range of values they can contain.

It is presumed that each simulation is capable of encoding and translating these values to binary format, knowing in advance the exact location and number of bits in which each value exists (as defined by the standard). Therefore, a set of PDUs used in different fields for different purposes are predefined in the standard and only these PDUs are available to simulation engineers. Such an approach is very inflexible. If a custom type of PDU is required, it has to be included in the standard first and only then can it be used in simulations. For example, in order to simulate the effect of wind on multiple buildings, a scenario simulator needs to be set up first to send scenario information to other simulators such as wind generator simulator, structure analysis simulator, and damage simulator. The scenario information needs to include building location, geometry, and material types, and thus needs a more complex data structure than what DIS provides in its PDUs. Therefore, it is a significant challenge to create such a simulation with DIS.

In point-to-point communication (Fig. 1), there is no middleware or center server maintaining message exchange. Instead, each message sender would manually connect to its receivers using their network Internet Protocol (IP) addresses. This makes it only suitable for its original focus, that is, individual weapon simulation for military training and real-time wargaming but not scale well for the aggregate level simulation of a battlefield. Another problem is that although the standard describes in great detail the format of data being sent over a network, it does not specify how exactly network communication should be implemented and is open to any implementation (typically hidden to an end user). This further leads to the following two disadvantages: (1) it is up to users to create their own communication tools by following the standard; and (2) users must be capable of creating the tool themselves or must be able to obtain a premade solution (open-source or commercial) such as Open DIS (McGregor et al. 2008) and VR-Link (VT MAK 2018).

Knowledge of the data format in advance is the simplest manner to maintain consistency across simulations. However, DIS puts full responsibility on the user to correctly implement its standard. Inflexibility in the data format and consequences for peer-to-peer network connections make it scale poorly for different use cases and difficult to implement in the case scenarios involving multiple simultaneous simulations that are quite common in a disaster scenario.

Distributed Interactive Simulation also does not encompass other important features, such as time management and network management that would be desirable when multiple simulators are involved in a simulation. Newer standards attempted to address these drawbacks. Among them are Common Training Instrumentation Architecture (CTIA), Aggregate Level Simulation Protocol (ALSP), HLA, and TENA. Among these, HLA and TENA are the most widely known.

2.1.2 High Level Architecture (HLA)

HLA was developed by the U.S. DoD and the Defense Modeling and Simulation Office (DMSO) in 1995 (Hollenbach 2009) based on experience with DIS and the desire to develop a high-level simulation architecture that would facilitate interoperability and reusability of distributed simulation components. It became a DoD standard in 1998 (The U.S. DoD HLA 1.3 specification) (Dahmann et al. 1998) and an IEEE standard in 2000 (HLA Working Group 2000), and then evolved again to its latest version in 2010 (HLA Working Group 2010), and continues to be an active standard as of 2019. High Level Architecture was once widely used in distributed simulations, including some natural hazard simulations where it was utilized to model interdependencies between critical infrastructure systems (Fiedrich 2006; Eusgeld and Nan 2009; Eusgeld et al. 2011) and disaster responses (Liu et al. 2007; Hwang et al. 2016).

High Level Architecture has some advantages over DIS. First, HLA-compliant software uses an application programming interface (API), which in turn can be used by a simulation member, called a federate application in HLA. This facilitates connections between federates. The API includes functionality to control time management and syncing data exchange between different simulations. Second, unlike DIS where the data structure has to be predefined in the standard, by invoking the HLA Object Model Template (OMT), a user is allowed to model data as an object instance or an interaction (also called HLA objects) that includes the data (attributes and parameters, separately) to be exchanged among the federates in a federation execution at design time. This is clearly more flexible than the DIS alternative for a disaster simulation where an interaction instance can be conveniently used to model an earthquake event, an evacuation event, or a recovery event, and its effect can be reflected in some object instances used to model buildings or lifelines. Third, in terms of how data are exchanged, instead of using the point-to-point communication in DIS (Fig. 1), HLA routes data as HLA objects via a middleware (called runtime infrastructure (RTI)) using a Publish/Subscribe (P/S) pattern (Fig. 2). In this way, the sender and receiver federates

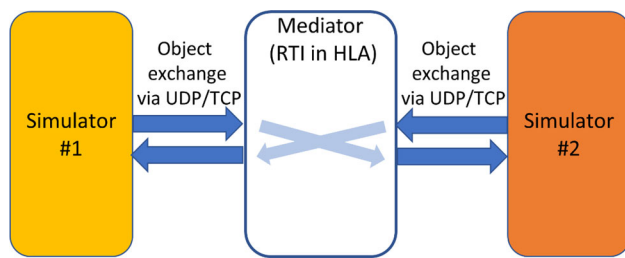


Fig. 2 Mediator-based object exchange via user datagram protocol/transmission control protocol (UDP/TCP) in high level architecture (HLA). RTI: runtime infrastructure.

just need to declare what data they need and what data they provide, without the requirement of knowing about other federates. This feature further improves the reuse of each federate by decreasing the coupling among different federates in a federation, and makes it scale better for systems with a large number of simulations. In other words, the federates each connects to a single point, rather than to each other.

Although HLA is more general than DIS, it still suffers from multiple flaws. It is defined as a simulation systems architecture framework (HLA Working Group 2010), not a software or an implementation. Therefore, HLA software must be able to “connect” to the RTI. However, it does not specify how the connection works, leaving the implementation up to the creator of the HLA compliant software. Similarly, time-management is described as a function that must exist, but how it functions can be unique in many implementations. Indeed, its very existence is all that is needed to be compliant. For example, if the wind-building simulation mentioned in Sect. 2.1.1 is created in HLA with different RTIs, the simulation’s efficiency can be quite different depending on detailed RTI implementations. In practice, users usually have to try different RTIs to get satisfactory efficiency performance. Moreover, having to compile a data format allows efficiency to be maintained as to the number of bytes in each message but requiring compilation on a user’s local machine every time that the data format and content changes is onerous.

Commercial HLA software packages available for use are CERTI (ONERA 2018), Portico (Calytrix Technologies 2018), MAK (MÄK Technologies 2018), and Pitch (Pitch Technologies 2018). While there are multiple open-source solutions, as of 2019, many of them have been discontinued or are unobtainable, or they are not 100% compliant with HLA standards. It is not trivial to program HLA software from scratch, as there are six separate management systems (federation, declaration, object, ownership, time, data distribution) (HLA Working Group 2010) that have their own specification section, each lengthy, but lacking in the detail required for systematic implementation. The benefit of the standard is that a prepared

simulation can be compatible with any compliant HLA software. However, generally, only one vendor’s HLA software can be used in a federation to allow compatibility with the RTI and local federate’s API. In order to help ensure compliance and encourage adoption, the U.S. DoD offered a public service to check if a new implementation met the HLA standard, but this service was later abandoned when the original website shifted domains (Behner and Lofstrand 2017).

High Level Architecture is still in use, but interest in it has decreased since its inception. The standard’s ambiguity and a lack of easily available implementations have made it difficult for newcomers to utilize HLA in practice.

2.1.3 Test and Training Enabling Architecture (TENA)

Test and Training Enabling Architecture was introduced by the U.S. DoD. Designed after HLA, TENA’s development traces as far back as 1998 (Cozby 1998) and continues to be maintained as of 2018. Similar to HLA, TENA allows for the development of individual simulations interoperable with each other for distributed systems, saving time and money in the development process. As a tool, its functionality is revised based on early user feedback, but its core intentions drive its development.

The architecture of TENA consists of TENA-compliant applications and simulations, TENA Middleware, and TENA Utilities, including a gateway accessible by non-TENA systems. The middleware acts as a communication channel, where data must be formatted according to a TENA Object Model. One advantage of TENA over HLA is that it allows data to be exchanged between custom object models as long as they are used by two or more simulators. However, custom object models in HLA usually risk being incompatible with other simulators that can only recognize models from HLA’s predefined model set.

Different from HLA and other standards, TENA is not intended to act as a professional standard document (Powell and Noseworthy 2012), but as a standard tool. Monitored closely by its development group, access to TENA and its documentation requires applying for a free account through an online portal. While this process is open to all applicants, the existence of a screening process that asks for contact information, project intentions, and grant usage makes it difficult for researchers to test TENA’s functionality to confirm it meets their requirements. In addition, since TENA places more emphasis on real-time applications, it has only provided partial time management ability compared with HLA, which makes it less convenient for simulations that need complex time synchronization. Therefore, even though TENA was made explicitly to overcome some limitations in HLA, its inaccessibility and the fact that some of its functionality and

design are still in development, make TENA a difficult choice for practical use cases in the near future.

2.1.4 Data Distribution Service (DDS)

The DDS is a standard for data communication between distributed machines and software for real-time systems (OMG 2015). Unlike the previous solutions in this section that are oriented for distributed simulation and created by government entities, DDS was originally designed for real-time distributed operational systems and developed by a professional non-profit collective called Object Management Group since 2004 (OMG 2004). The DDS is not a single tool or software solution, so users must utilize accessible documentation on the standard to prepare their own, or must use existing solutions, including RTI Connext (Real-time Innovations 2018) or OpenDDS (OMG 2018). The intention is that all DDS solutions follow the specification carefully, such that each user can use any vendor's DDS solution and be interoperable with each other on the network. Professional demonstrations have been given to show this interoperability to be possible with DDS software from different vendors. It has been used for disaster management in natural hazards engineering (Lazarov et al. 2015; Ray et al. 2017).

Best suited for Internet-of-Things (IoT) applications, DDS is flexible for use across a variety of domains jointly with HLA or as a replacement. In DDS, data are pre-defined as a message format in a struct-like file suffixed with .idl and compiled with the DDS software to make it recognizable when it is written to or retrieved from the DDS global data space with a DataWriter or a DataReader. Using a specially designed topic-based P/S mechanism to share data within a domain participant, DDS does not depend on any global knowledge and supports fully dynamic discovery and matching of different DataWriters and Data Readers, which is more flexible than HLA that still requires static declaration in Federation Object Model (FOM) even though different publishers and subscribers can be matched dynamically. It also provides richer (22 versus 2 in HLA) quality of service (QoS) policies that help to control local and end-to-end properties of DDS entities. Conversely, since it was originally designed for real-time application in distributed operational systems, the main disadvantage of DDS is that it does not explicitly provide time management mechanisms for different types of time advancement controls as HLA does.

While DDS is not as feature-complete as HLA, its simplified standard makes it more accessible for users. Not requiring a single access point (like HLA's RTI) makes it less prone to slow-down from the RTI's perspective when adding more simulations, and the complexity of connections is handled internally without the user's concern.

Compiling a data format ahead of time is still a limitation that TENA sought to overcome but doing so enables DDS to maintain optimal speed in data communication.

With each of these solutions, the need to study lengthy standards and rules together with the need to deal with the compatibility of legacy standards, often make them difficult to use. However, these solutions provide instructions that others can follow, putting the responsibility on individual users to ensure that their own local simulation is built correctly to be compliant without worrying about how other simulations might function.

2.2 Standalone Tools

This section reviews two standalone data passing tools, ROS and LCM.

2.2.1 Robot Operation System (ROS)

Unlike the approaches listed above, which are either designed for distributed simulation or distributed operational systems, ROS is a robotics middleware that provides services including hardware abstraction, low-level device control, package management, message passing, and so on (ROS.org 2018). Such a design makes it possible for robotic engineers to quickly and conveniently build up a robot by taking advantage of many existing hardware drivers and implemented algorithms distributed as ROS packages (Xu et al. 2018; Xu et al. 2019).

While ROS is best suited for applications in robotics, its message passing design based on a Publish/Subscribe communication can be applied to simulation applications in natural hazards scenarios with some benefits. More specifically, the three different patterns of data exchange supported by ROS all have their corresponding applications in distributed simulations. In most cases, the output of one node needs to take as input the runtime outputs of some other nodes and in turn, its output can be used as part of the input for other nodes. Such input and output information generally needs to be exchanged continuously at a small timestamp and can be modeled as messages in ROS.

A message in ROS is a data structure that can be defined flexibly in a .msg file by following a syntax similar to C structs. Most messages have a header field that is filled with a simulation timestamp by ROS and is used for time management. Once a message is compiled with ROS, by importing or including its bindings, a node can encode and retrieve information into and from the corresponding message automatically with ROS. Another type of data exchange is conducted in the Request/Response way where the data structures in a request and a response are formatted together as service in ROS and are defined in a .srv file by following similar syntax to ROS messages. In this pattern,

by agreeing upon the same *srv*, a client provides the required input for the request and requests a server to give a response based on the request. The returned response depends on the implementation on the server side.

For a distributed disaster simulation, this is suitable for acquiring the global configuration (such as the scenario information in the wind-building example in Sect. 2.1.1) from the server or for commanding some other nodes to behave in a specific way. However, for the latter use, if the server takes a significant amount of time to perform the requested action or does not respond to the request, the client would not receive any feedback and thus know nothing about the status of the server. This lack of knowledge of the status of the server can be solved by using the *actionlib* pattern. This pattern specifies the formats of the goal (the result and the feedback message) in an action file in a similar way to ROS *msg* and ROS *srv*. In this way, after a client sends out an action request to a server, it can keep listening to the feedback from the server and make further decisions based on the feedback. This approach is extremely useful for distributed disaster simulations whose nodes are simulating reality at different time scales (such as an earthquake node and a recovery node). Those nodes that run faster can request the others to catch up via an action request.

For its wide use in the robotics community, ROS is well documented, and it is easy to access help from different technological forums. Despite the above advantages it also has some drawbacks. Since it was not specially designed for simulation purposes, it lacks implementation of time management and quality of service (QoS) policies compared to other simulation-oriented approaches. Besides, it does not provide a convenient way to set up connections among different nodes. Each node needs to explicitly specify the topics the node subscribes to and the way the node wants to receive the messages on these topics. Therefore, when a node is used together with a simulator, the code for message communication is usually interspersed with the code for simulator functions. This lack of convenient communication interface makes it scale poorly as the number of nodes increases, limiting its suitability for large-scale simulations.

2.2.2 Lightweight Communication and Marshaling (LCM)

Lightweight Communication and Marshaling is another data passing tool oriented for real-time robotics applications (Huang et al. 2010). It has been applied to disaster simulations (Lin et al. 2018; Lin et al. 2019) recently owing to its beneficial features including low-latency, platform and language independence, and publish/subscribe data transmitting scheme. As a lightweight solution, it is mainly comprised of three functionalities—message type

specification, message marshaling, and message communication—and, despite what its name suggests, some analysis tools. In LCM, the data to be transmitted over a network need to be first structured as a message type, by following its specific type specification language whose syntax is very similar to C structs.

After the message type is well defined, the provided *lcm-gen* tool is invoked to generate its language-specific bindings that can be further included or imported in custom simulators to use the corresponding message. Such bindings can be generated to support multiple languages (C, C++, C#/.NET, Java, Lua, and Python) on different platforms (Linux, OS X, Windows, and any POSIX-1.2001 system), which is very convenient for simulator developers with different preferences. In the actual communication, a message is marshaled by attaching to it a fingerprint derived from its channel name and message type and routed from its sender to its receivers with a Publish/Subscribe pattern. LCM uses multicast UDP based peer-to-peer communication, in which there is no mediator and each simulator is both a sender and a receiver.

In LCM, messages are routed to all the LCM subscribers that are in the same multicast group and each subscriber further selects the messages it is expecting based on the channels to which it has subscribed. As shown in Fig. 3, for any simulator *#i* in the multicast UDP group, its LCM subscribers receive all the messages published within the same group. After receiving these messages, its subscribers automatically select the messages published to the channels that they have subscribed to by dropping all the other messages, such that the simulator *#i* can work with the messages it is interested in by just subscribing to the appropriate channels.

Compared to ROS, LCM also provides some useful tools (logging, replaying, and inspecting traffic) to help with debugging during development as well as help inspect and analyze the simulation during testing. As a pure data passing tool, LCM provides great flexibility for further development of different features by users. However, as a

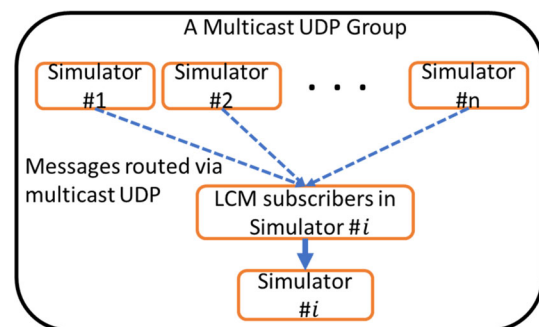


Fig. 3 Message exchange via multicast user datagram protocol (UDP) in lightweight communication and marshaling (LCM)

robotics tool, it inevitably lacks the specific features dedicated to simulation, such as time management and QoS policies. Moreover, due to reasons similar to ROS, it does not scale well.

Compared to the approaches in the last section, approaches from the robotics community include a ready-to-use library and provide well-documented instructions, which make them easier to use for skilled programmers. The main problem with these methods is the lack of a systemic way to deal with scalability issues. It is the users' responsibility to make sure that the connections among different simulators and time management for each simulator are set up correctly by adding corresponding code to the simulators. This mixture of code for connection and simulator functionality makes it hard to manage the simulators when their numbers greatly increase and thus limits these approaches to small or medium-scale problems.

3 Limitations

Based on the review in Sect. 2, in this section, three main limitations are recognized from the perspective of user experience when trying to create a complex coupling distributed simulation conveniently and efficiently from scratch in the fields related to natural hazards.

3.1 Lack of Easy-to-Use and Standard Solutions

Among the standards and standard-based solutions, both DIS and TENA have to use pre-defined sets of messages, which is not flexible for information exchange between different natural hazard simulators (as explained in the wind-building example in Sect. 2.1.1). Moreover, they can only build real-time simulations that run in wall-clock time. This makes the simulation of a recovery process, a typical simulator involved in a disaster-related simulation, very prolonged and inefficient.

Compared with DIS and TENA, HLA and DDS are more suitable for disaster simulations. As standards, they levy many requirements on the design of API, and some implementations have been designed by following such specifications. However, it is still difficult for a novice to rapidly build a functional simulation and, for experienced users, non-trivial to achieve desired simulation performance. On one side, with the aim of allowing an interoperability level of integration across areas in distributed simulations by defining common data types and specifying APIs, they have become formidably long standards that are quite hard to follow and adhere to. Therefore, it is common that some implementations just follow and support part of the API specifications and it is necessary for users to be

aware of the deviations from the standards in addition to a basic understanding of HLA or DDS concepts.

While HLA and DDS include the detailed requirement for API, they do not specify exactly what algorithms need to be used and how the API function should be implemented, which leaves the flexibility to API implementers. This flexibility for the implementers leads to diverse API implementations with different vendor-specific features and advantages, and it is important for users to be able to choose the appropriate implementations to achieve their custom simulation performance goals. In practice, achieving custom simulation performance goals requires the users to know about different implementations and the differences between them, since these differences are generally non-trivial and experience from one implementation cannot be directly applied to another.

Unlike tools built on standards, ROS (and LCM) can be thought of as a standalone tool providing much less, but necessary, APIs for data sharing, which is particularly well suited to users who need to quickly build up a small-scale application-specific simulation and distribute it over a network. While this approach provides a flexible and convenient way of constructing simulations, the issue for this category of tools is that different simulators have to agree on the structure of the shared message due to lack of standardization, even though it is not difficult to come up with simple specifications on the data structure for application-specific problems.

3.2 Lack of Scalability and Extensibility for Building Large-Scale Simulations

Generally, standards do not specify the scale of simulation that an API needs to and should support, and in theory, users can try to connect as many simulators as they want in one simulation. However, in practice, the scalability of the standard-based methods is greatly impacted by the detailed API implementations, and the practical performance can vary greatly as the size of the simulation changes. When the simulation scale is small, such as a simulation of interdependencies between natural hazards with several buildings, peer-to-peer communication is preferred since mediator-based communication would need one extra message copy for each subscriber of a message and thus need more bandwidth and result in more latency.

As the simulation scale increases to the middle scale, such as a city-scale simulation of natural hazard interdependency, mediator-based communication becomes preferable. The reason is that the overhead resulting from additional message copies becomes less important compared to the total message routing time, and mediator-based communication also provides other benefits such as

monitoring of individual simulators and more flexible central time management.

However, when the scale increases to a large scale (for example, state level) the performance bottleneck of the simulation is usually the power of the processor where the mediator runs since the mediator has to route a great number of different types of messages and conduct corresponding time management for a large number of simulators. Therefore, it generally needs additional algorithms to distribute the work of the mediator over multiple processors, which increases the complexity of the simulation. Since both the standard-based tools and standalone tools reviewed above use a fixed message delivery method, it is difficult for them to always obtain the best performance for different simulation scales.

For extensibility, DIS and TENA are seriously limited since they can only use fixed sets of messages. Other standard-based tools such as HLA and DDS support custom messages, which make them convenient for extending the information shared between different simulators. For standalone tools, new information to be shared has to be defined as new messages or added to the old message definitions, and the created or modified message definitions have to be recompiled to make sure they can be recognized by different simulators. This process almost always includes modification of the relevant simulators to make sure they can send and receive the pre-compiled messages. This process is not convenient and sometimes even difficult for experienced users.

3.3 Inability to Rapidly Build and Integrate Application-Specific Simulators

The most important goals of standards and standard-based tools are to improve reusability and interoperability and make simulators usable across fields. The benefits are significant when users have easy access to many choices of simulators that have been developed by people from different fields for different purposes. However, in practice, these benefits are limited for two reasons. First, it is still challenging to integrate simulators developed by others without any knowledge of them, even if they are compatible with the same standard. Such knowledge includes simulator time resolution, simulator mode (time-driven, event-driven, or hybrid), and time management option, which users may have to modify to make the simulators work correctly. Therefore, simulators' reusability and interoperability are mainly achieved in some relevant simulations that are developed by the same group of people who developed the simulators.

The second reason is that the complexity of utilizing the simulation tools to develop reusable and interoperable simulators limits the number of available simulators.

Skilled simulation engineers are good at achieving reusability and interoperability of simulators when they are given functional simulators from different domains. However, it is usually difficult for them to develop simulators from scratch without enough domain knowledge. Instead, it is the people with good domain knowledge that are more suitable to develop domain-specific simulators for specific applications. However, the complexity of standard-based simulation solutions creates a non-trivial gap between domain knowledge and a simulator compatible with the same simulation solutions.

It is also difficult to rapidly get started with building a functional simulation for domain expertise with limited simulation background. Users need to at least have some knowledge of the standard, the usage of the API implementation they have selected, and some programming skills to configure and compile the standard-based tool on their custom computers, which entails a steep learning curve. In this regard, standalone tools are also inappropriate. For these tools, time management has to be implemented additionally and it is difficult to separate message exchanging code and simulator code for scalability (domain users may care more about scalability than simulation efficiency). These are all challenging to achieve for users without much programming experience.

4 Recommendations

This section aims to improve the recognized limitations as listed in Sect. 3 with two improvement recommendations. In a simulation involving multiple analysis models, it is usually natural and straightforward to implement a simulator as a separate module that interacts with other modules and implement a sub-simulator as a separate component that interacts with other components within the same simulator. For example, in a simulation of interactions between sequential earthquakes and corresponding recovery processes, it is natural to define a seismic simulator separately to model the earthquake and its impact on the infrastructure in the environment, and a recovery simulator to model the recovery effort and how infrastructure functionalities are recovered. The seismic simulator can include several sub-simulators that work together to complete its tasks, such as a sub-simulator to model the effect of the earthquake on the ground surface and other sub-simulators to model how such effects further interact with and damage buildings, transportation systems, and other infrastructure. Similarly, the recovery simulator can include a group of sub-simulators to model how the recovery process evolves with the interaction among resources such as first responders, equipment and material, recovery strategy, and as-is recovery status. There can be any number of simulators and

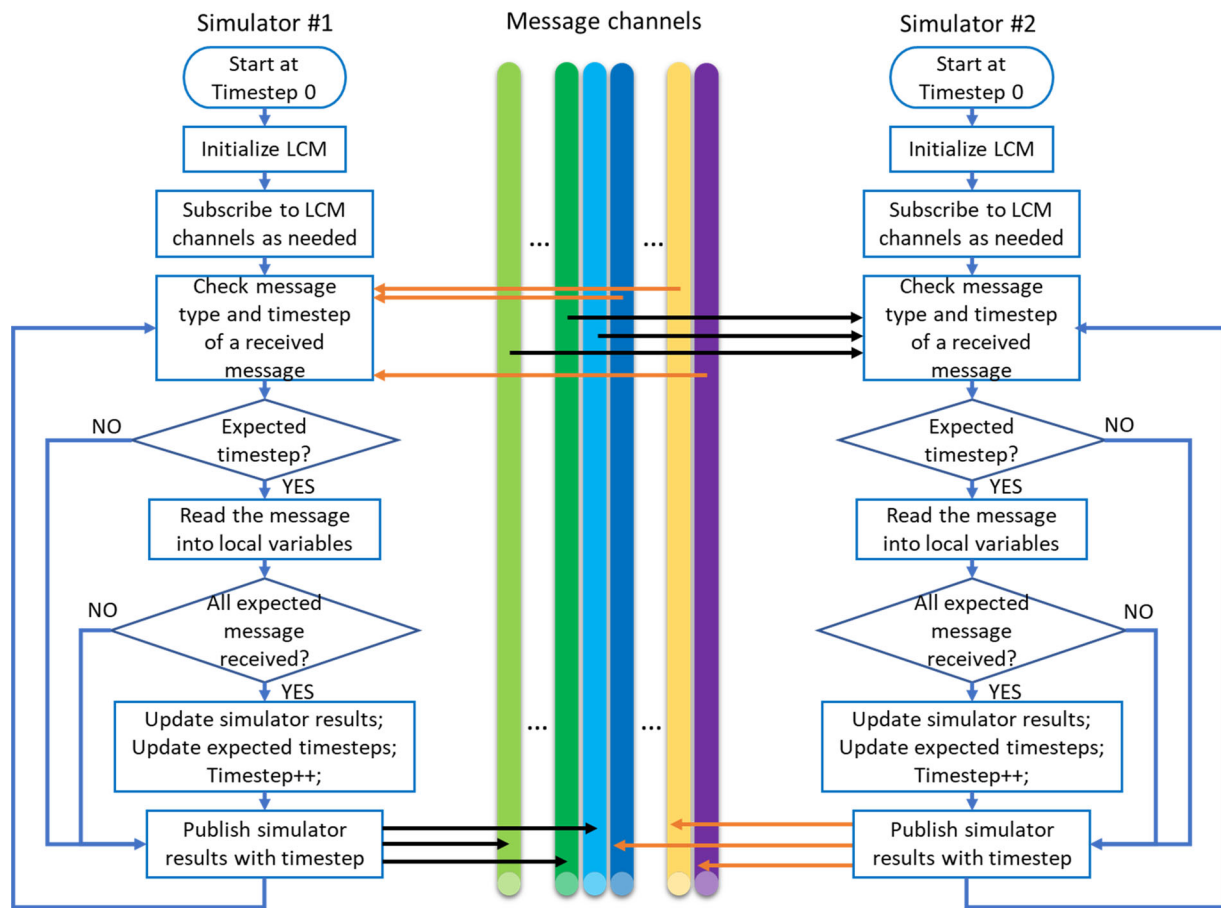


Fig. 5 A lightweight communication and marshalling (LCM)-based distributed coupling analysis framework

different simulators can be developed with different languages and run on different operating systems listed in Sect. 2.2.2. In this framework, different simulators can be developed separately and connected with LCM-based message passing. In each simulator, it first initializes LCM and subscribes to the message channels from which it can get the messages that the simulator depends on. Lightweight Communication and Marshalling can help receive the available messages from the subscribed channels, and the current simulator needs to decide if a received message is one that is currently expected.

There are two things to check—message type and expected timestep—for this type of message. After all the expected messages are received, this simulator will continue for one timestep, update simulator results, update expected timestep for each expected type of message, and update the current timestep in the simulator. After getting new simulator results, the simulator will immediately publish it with the current timestep value. It should be noted that a simulator still needs to keep publishing simulator results even in situations where failure is encountered in the process of checking if a message is an expected message or if all the expected messages are received. The

reason is that the messages are live data on the channels, and the same message needs to be sent repeatedly in order to make sure the message can be received by the simulators that need it to proceed. Different simulators can be developed separately by following the same steps and then they will automatically work together to make up a complete distributed analysis. Compared with standards and standard-based solutions, this framework is more flexible and more convenient to quickly create a small-scale analysis with domain knowledge.

In addition, as shown in our work in Lin et al. (2018) and Lin et al. (2019), even though simulators and sub-simulators were not differentiated from each other and all the separate components were implemented as separate simulators, LCM still worked efficiently to pass messages between different simulators benefiting from the fact that it uses UDP multicast as its transport and does not use a mediator to route the messages or broker connections between models. This LCM-based model communication scales well with the number of the involved models and is also extensible. However, the code dealing with receiving and sending messages was implemented together with the simulator functions, and thus it requires simulator

developers to know basic usage of LCM. Moreover, this coding work becomes more complex and error-prone as the number of simulators increases and the interaction between simulators becomes complex. This drawback further limits scalability and extensibility in practice and makes it only suitable for relatively small-scale analyses.

Ideally, simulator developers should not be required to have deep knowledge about the distributed analysis platform being used. Instead, they should be able to focus their attention on developing simulators in their domains and specifying how they want their simulators to communicate with each other. In order to achieve this benefit, a message wrapper design is proposed to work together with the simulator functions and receive and send messages from and to the channels for them. The term “channel” is inherited from LCM and is used to illustrate the new concept design of a message wrapper. As shown in Fig. 6, the proposed message wrapper acts as a bridge connecting message channels and a simulator function. It subscribes to the channels from which the simulator function gets input data, decodes messages when required messages are received, calls the simulator function to update the outputs of the simulator, encodes output messages, and publishes them to the specified channels.

Figure 7 shows the procedures of developing a simulator with a message wrapper. This general design can work with any standalone data passing tools such as LCM, ROS, or other custom data passing platforms. For convenience, LCM is used as an example here to show the detailed implementation of the files in Fig. 7. Simulator developers first need to prepare two files: a simulator configuration file and a message definition file. The simulator configuration

file includes all the settings about the simulator, including simulator name, the channels this simulator needs to subscribe to, the channels it needs to publish on, time step relationship between the current simulator and the messages it depends on, the simulator’s dependency on historical data, and whether the simulator needs to publish initial data for other simulators to start working.

The message definition file includes the name of the variables in each message and the corresponding data types. It should be noted that even though LCM can decode the message and its variable types automatically, these variables need to be stored as local variables in the wrapper and thus the variable types still have to be provided in the message definition file. In the case of LCM, it is straightforward to prepare these two files by drawing a communication network and referring to the LCM message definitions. When these two files are ready, a simulator function prototype generator (Fig. 7) is used to generate the function prototype of the simulator function.

This function prototype generator is implemented in a simple way that all the variables included in the input messages are listed as input arguments, all the variables in the output messages are listed as return values, and the simulator name is used as the function name. Therefore, only the information about simulator name, channels to subscribe to, and channels to publish on in the simulator configuration file are used to generate the function prototype. Then simulator developers need to complete the created simulator function with only domain knowledge. After completing the simulator function, the message wrapper can be run to handle message exchange when it works with the same simulator configuration file, the same

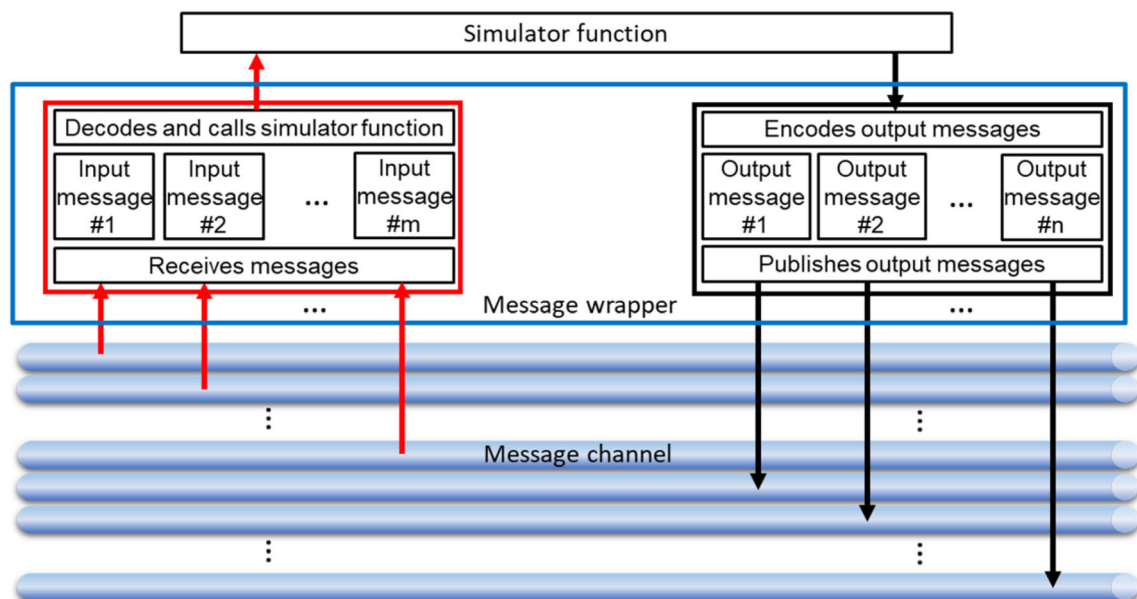
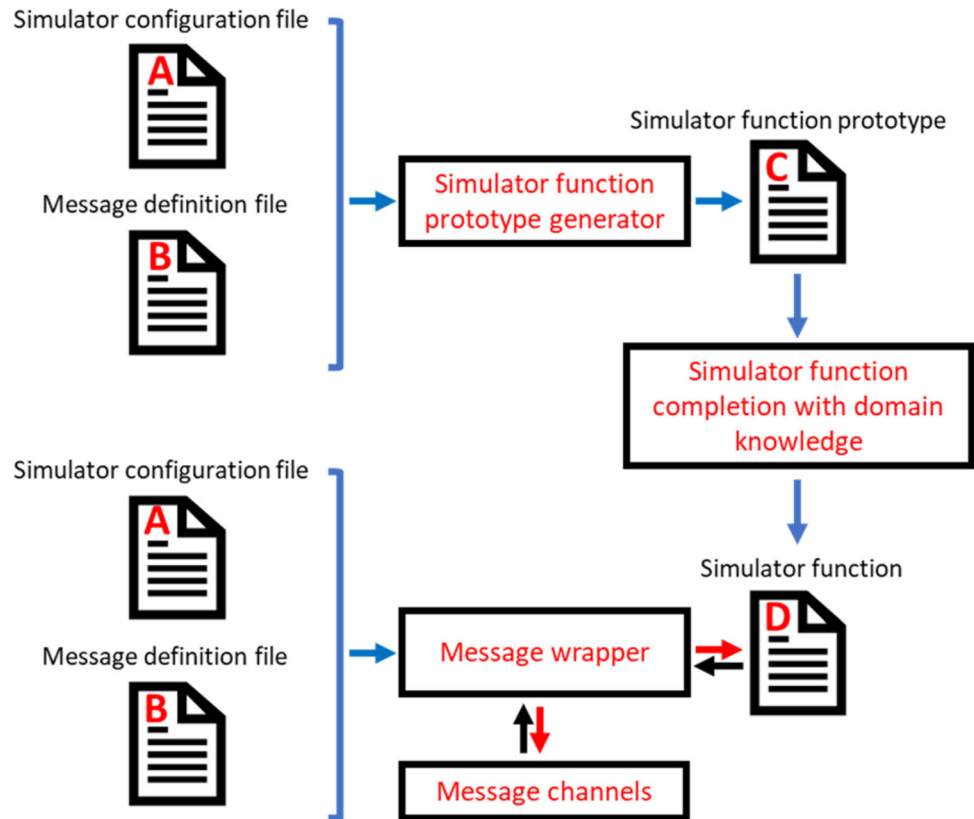


Fig. 6 The concept of a message wrapper

Fig. 7 The procedures for developing a simulator with a message wrapper



message definition file, and the completed simulator function.

Figure 8 shows how a message wrapper computes output messages based on the input messages and publishes the output messages on the specified channels. This process is very similar to those in the simulators depicted in Lin et al. (2019), the only difference is that the simulator function and the code for message exchange are now completely separated with the proposed message wrapper. Therefore, domain users just need a little effort to develop a simulator since all they need to know is the domain knowledge to complete the simulator function and the relationship

between different simulators. Besides, in the process of completing the simulator function, it is flexible for users to use any useful software and/or hardware to facilitate simulator development and/or accelerate the simulation.

Our previous work (Lin et al. 2018; Lin et al. 2019; Lin and El-Tawil 2020) showed the effectiveness of the proposed coupling analysis. Here, in order to further verify the design of the message wrapper, the wrapper-enhanced framework is used to replicate the active control algorithm introduced in Reinhorn et al. (1987) and restated in Fig. 9. The components in Fig. 9 are then formalized as three simulators in Fig. 10, where $P(t)$ represents the force

Fig. 8 An example implementation of the message wrapper for lightweight communication and marshalling (LCM)

1. Initialize LCM.
2. Configure the channels to subscribe to and the channels to publish on.
3. Configure the parameters to control receiving of messages and the timestep of the simulator.
4. Scan and decode the expected messages.
- If all the expected messages are received, go to Step 5, else go to Step 7.
5. Call the associated simulator function to update simulation results of the simulator.
6. Update control parameters and the current timestep.
7. Publish current simulation results with the current timestep.
- Go to Step 4 and repeat the above steps.

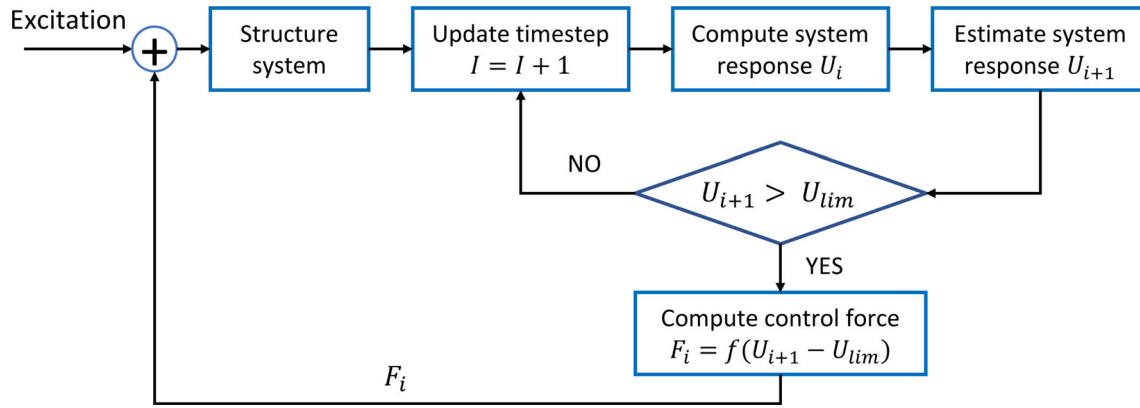


Fig. 9 The active control algorithm introduced in Reinhorn et al. (1987)

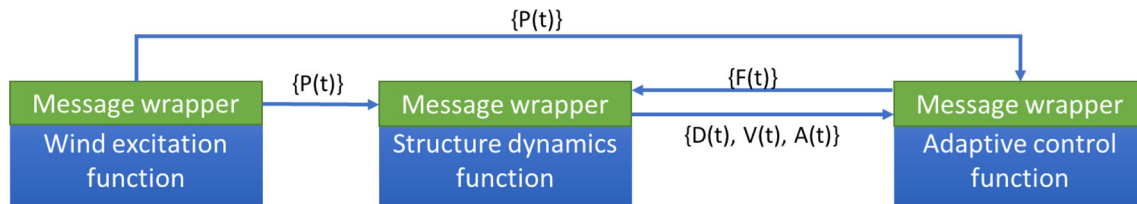


Fig. 10 Distributed analysis design of an active control algorithm

caused by a wind excitation. $D(t)$, $V(t)$, and $A(t)$ represent structure displacement, velocity, and acceleration at time t respectively and together compose response vector $U(t)$, and $F(t)$ is the active control force at time t . Wind excitation can be generated with different models, and here we adopted the model we used in Lin et al. (2019). For structure dynamics, one single building with stiffness K is assumed, and response U_{i+1} was found as follows from response U_i and time interval Δt for this single degree of freedom (SDOF) system.

$$U_{i+1} = \begin{bmatrix} D_{i+1} \\ V_{i+1} \\ A_{i+1} \end{bmatrix} = \begin{bmatrix} 1 \\ a_1 \\ a_0 \end{bmatrix} K^{-1} F_{i+1} + Q U_i$$

$$= \begin{bmatrix} \frac{1}{2} \\ \frac{\Delta t}{4} \\ \frac{\Delta t^2}{4} \end{bmatrix} K^{-1} F_{i+1} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & \frac{\Delta t}{4} & -1 \end{bmatrix} U_i$$

In this equation, the second column of Q can be written as:

$$Q = \begin{bmatrix} Q_{12} \\ Q_{22} \\ Q_{32} \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ \frac{\Delta t}{4} \end{bmatrix}$$

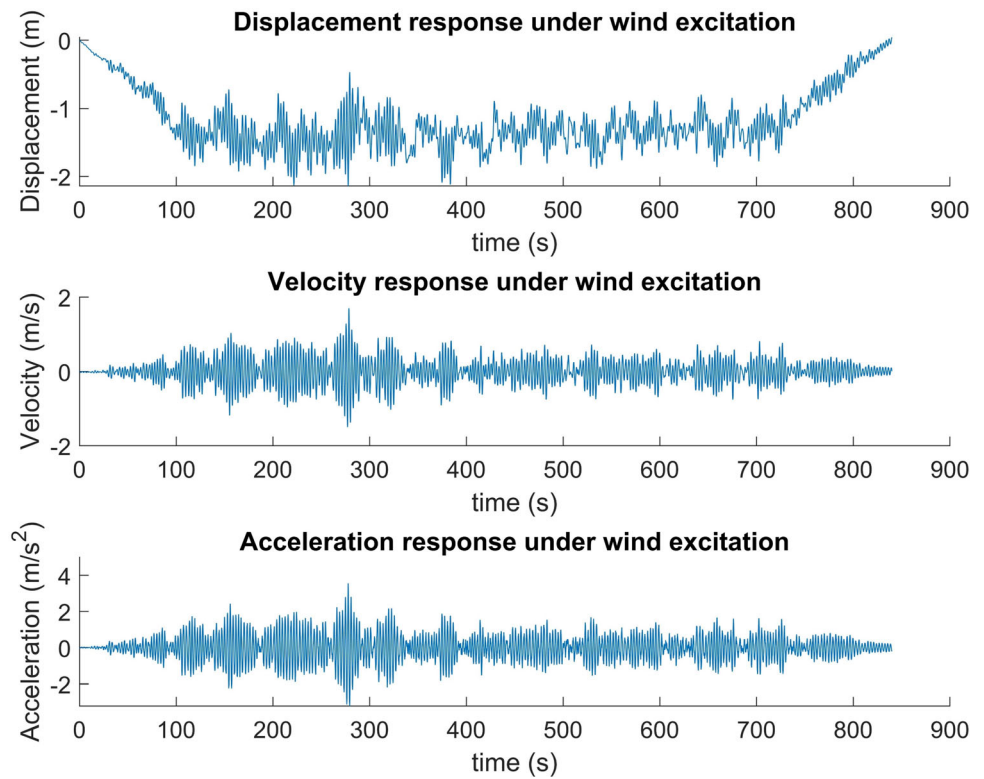
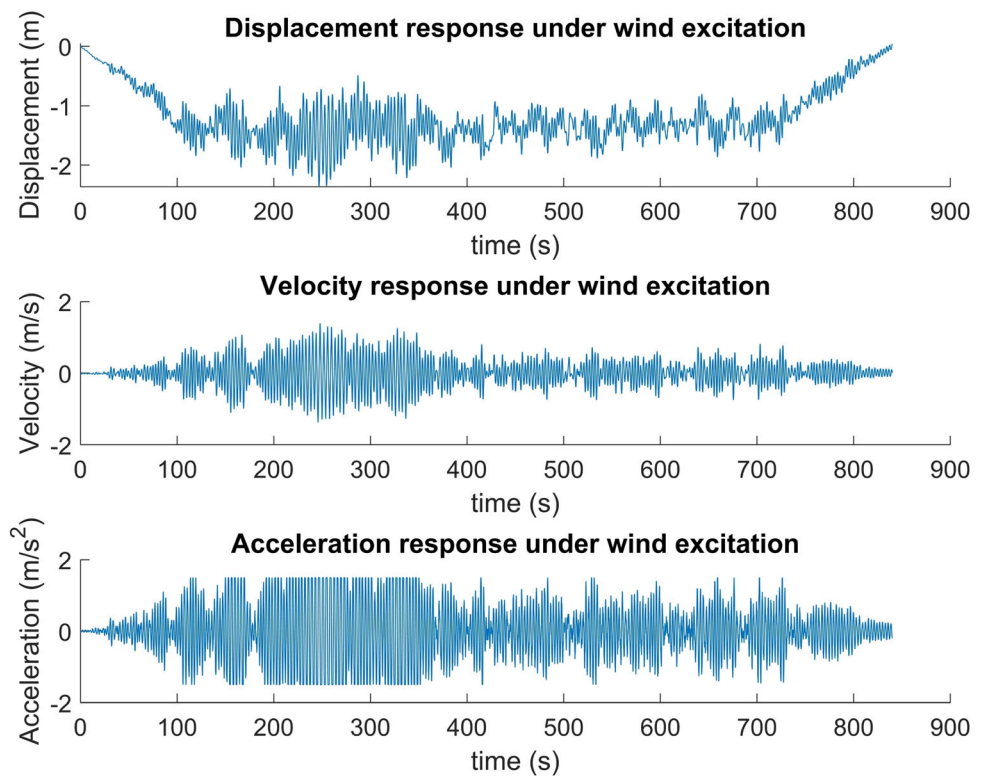
For the adaptive control function, acceleration was selected to be controlled with the following control strategy proposed in Reinhorn et al. (1987),

$$F_i = \frac{M |A_{lim} - A_{i+1}|}{\Delta t} Q_{32}$$

where M is the mass of the building, and A_{lim} is the acceleration magnitude to be limited. Two analysis results with and without active control are shown in Fig. 11 and Fig. 12.

It can be observed that with active control, acceleration was successfully limited to the range of $[-1.5, +1.5] \text{ m/s}^2$, and the displacement and the velocity responses were impacted correspondingly. The results demonstrated that the framework and the wrapper help discover the interdependency between the structure dynamics model and the adaptive control model. With domain knowledge from wind engineering and structural engineering, this distributed analysis model can be constructed conveniently by following the fixed steps introduced in this section without knowing how to use LCM to exchange messages.

However, it should be noted that even though the message wrapper can be used as an extension to any data passing platform and help improve the scalability and extensibility in terms of implementation, current widely used mediator-based data passing platforms generally suffer from scalability problems in message communication. For those that adapt well with the simulation scale, such as LCM, they still lack the necessary time management and error-recovery mechanism for robust simulation. This challenge leads to the second proposed improvement that

Fig. 11 Analysis results without active control**Fig. 12** Analysis results with active control of acceleration

jointly uses peer-to-peer and mediator-based communication.

4.2 Hybrid Data Passing Between Message Wrappers

As shown in the recommended design in Fig. 4, peer-to-peer communication is adopted to handle communication between different sub-simulators in each simulator and the simulator itself via message wrappers. This local communication generally needs more frequent and more extensive message exchange as compared to the simulator-simulator communication and is suggested to be implemented with UDP multicast that was also the transport utilized in LCM. The benefit to this approach is that there are no additional copies of messages that otherwise would increase linearly with the number of subscribers and result in a large overhead if the number of subscribers is large. Moreover, even though LCM was originally designed for real-time robotic applications, based on previous experience (Lin et al. 2018; Lin et al. 2019), it was shown to work efficiently in time-step based simulation. However, mediator-based communication between simulators is suggested to be implemented with TCP transport that provides reliable and ordered information delivery.

For the two types of communication, different marshaling methods can be chosen according to the tradeoff between transmission efficiency and marshaling cost. The message wrapper of a simulator can be designed to decode the marshaled messages from its sub-simulators and marshal them in a different way and communicate them via the mediator. Generally, in order to simplify the platform design, the same message marshaling format would be shared between the two ways of communication with LCM marshaling being a good example. With the benefit of UDP multicast in LCM, the lcm-spy tool can be used to inspect traffic without additional cost. A similar traffic inspection tool can be developed for communication between local sub-simulators. However, for the inspection of communication between simulators, the inspection tool needs to be implemented as a separate simulator that subscribes to all the channels and thus adds additional inspection cost.

For a platform only based on mediator-based communication, the load of the mediator increases with the number of messages that need to be delivered at any time and is usually the bottleneck of a large-scale simulation. With the proposed hybrid communication, this issue can be greatly improved since all the sub-simulators will be handled by peer-to-peer communication that does not need the mediator and can be implemented efficiently. In addition, the mediator in the proposed solution is no different from the mediator in an RTI for HLA and can easily take advantage of the time management methods in HLA. The

platform can be also integrated with some error recovery mechanism by using a certain number of historical messages from the simulators kept in the mediator. In this regard, the hybrid communication design jointly uses the ideas of LCM and HLA and capitalizes on their respective advantages.

5 Discussion and Conclusion

Distributed simulation platforms are essential to implement coupling simulations and identify deep interdependencies among different simulators. This article provides a systematic review of existing platforms for natural hazard simulation, identifies the limitations in the existing solutions for hazard simulations, and proposes some recommendations on improving the design of data passing tools for coupling simulations. This survey study offers a reference for researchers in hazards engineering when selecting distributed simulation platforms and data passing tools. Moreover, this article serves as a guiding document towards developing a general-purpose distributed simulation platform for natural hazard applications by identifying the current limitations and providing feasible recommendations for future studies.

Acknowledgments The authors gratefully acknowledge the financial support for this research received from the United States National Science Foundation (NSF) via Grants ACI #1638186 and CBET #1804321. Any opinions and findings presented in this article are those of the authors and do not necessarily represent those of the NSF.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- AIR Worldwide. 2020. *AIR hurricane model for the United States*. <https://www.air-worldwide.com/siteassets/Publications/Brochures/documents/AIR-Hurricane-Model-for-the-United-States-Brochure>. Accessed 20 Apr 2021.
- ARA (Applied Research Associates). 2021. *HurLoss hurricane catastrophe model*. <https://www.ara.com/hurloss/>. Accessed 20 Apr 2021.
- Arguello, L., and J. Miró. 2000. Distributed interactive simulation for space projects. *ESA Bulletin* 102: 125–130.

- Azar, E., and C.C. Menassa. 2010. A conceptual framework to energy estimation in buildings using agent based modeling. In *Proceedings of the Winter Simulation Conference 2019: Simulation for Risk Management*, 8–11 December 2019, Baltimore, MD, USA.
- Barrett, C., R. Beckman, K. Channakeshava, F. Huang, V.S.A. Kumar, A. Marathe, M.V. Marathe, and G. Pei. 2010. Cascading failures in multiple infrastructures: From transportation to communication network. Paper presented at the *2010 5th International Conference on Critical Infrastructure (CRIS)*, 20–22 September 2010, Beijing, China.
- Behner, H., and B. Lofstrand. 2017. The new HLA certification process in NATO. Paper presented at the *Symposium on M&S Technologies and Standards for Enabling Alliance Interoperability and Pervasive M&S Applications*, 19–20 October 2017, Lisbon, Portugal.
- Bunea, G., F. Leon, and G.M. Atanasiu. 2016. Postdisaster evacuation scenarios using multiagent system. *Journal of Computing in Civil Engineering* 30(6): Article 05016002.
- Calytrix Technologies. 2018. *Portico*. <http://porticoproject.org>. Accessed 23 Dec 2018.
- Chen, S.-C., M. Chen, N. Zhao, S. Hamid, K. Chatterjee, and M. Armella. 2009. Florida public hurricane loss model: Research in multi-disciplinary system integration assisting government policy making. *Government Information Quarterly* 26(2): 285–294.
- Cozby, R. 1998. Foundation initiative 2010. Army Test and Evaluation Command, Aberdeen Proving Ground, MD, USA.
- D.S. Committee. 2012. IEEE standard for distributed interactive simulation – Application protocols. *IEEE Standard 1278.1–2012*. Piscataway, NJ: IEEE Standards Association.
- Dahmann, J.S., F. Kuhl, and R. Weatherly. 1998. Standards for simulation: As simple as possible but not simpler the high level architecture for simulation. *Simulation* 71(6): 378–387.
- Dai, K., W. Zhao, H. Zhang, G. Shi, and Y. Zhang. 2011. TENA based implementation technology for virtual test. *Journal of System Simulation*. <https://doi.org/10.1016/j.cageo.2010.07.006>.
- DIS Steering Committee. 1998. IEEE standard for distributed interactive simulation – Application protocols. *IEEE Standard 1278.1a–1998*. Piscataway, NJ: IEEE Standards Association.
- Dong, S., and V.R. Kamat. 2010. Robust mobile computing framework for visualization of simulated processes in augmented reality. In *Proceedings of the winter simulation conference*, 5–8 Decemebr 2010, Baltimore, MD, USA.
- Eusgeld, I., and C. Nan. 2009. Creating a simulation environment for critical infrastructure interdependencies study. Paper presented at the *IEEE international conference on industrial engineering and engineering management (IEEM)*, 8–11 December 2009, Hong Kong, China.
- Eusgeld, I., C. Nan, and S. Dietz. 2011. “System-of-systems” approach for interdependent critical infrastructures. *Reliability Engineering & System Safety* 96(6): 679–686.
- Fiedrich, F. 2006. An HLA-based multiagent system for optimized resource allocation after strong earthquakes. In *Proceedings of the 38th conference on winter simulation*, December 2006, Monterey, CA, USA.
- Ginting, B.M., and R.-P. Mundani. 2019. Parallel flood simulations for wet-dry problems using dynamic load balancing concept. *Journal of Computing in Civil Engineering* 33(3): Article 04019013.
- Grogan, P.T., and O.L. De Weck. 2015. Infrastructure system simulation interoperability using the high-level architecture. *IEEE Systems Journal* 12(1): 103–114.
- HLA Working Group. 2000. IEEE standard for modeling and simulation (M&S) high level architecture (HLA) – Framework and rules. *IEEE Standard 1516–2000*. Piscataway, NJ: IEEE Standards Association.
- HLA Working Group. 2010. IEEE standard for modeling and simulation (M&S) high level architecture (HLA) – Framework and rules. *IEEE Standard 1516–2010 (Revision of IEEE Standard 1516–2000)*. Piscataway, NJ: IEEE Standards Association.
- Hollenbach, J.W. 2009. Inconsistency, neglect, and confusion: A historical review of DoD distributed simulation architecture policies. In *Proceedings of the spring simulation interoperability workshop*, 23–27 March 2009, San Diego, CA, USA.
- Hori, M. 2011. *Introduction to computational earthquake engineering*. Hackensack, NJ: World Scientific.
- Hori, M., and T. Ichimura. 2008. Current state of integrated earthquake simulation for earthquake hazard and disaster. *Journal of Seismology* 12(2): 307–321.
- Huang, A.S., E. Olson, and D.C. Moore. 2010. LCM: Lightweight communications and marshalling. Paper presented at the *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 18–22 October 2010 Taipei.
- Hwang, S., R. Starbuck, S. Lee, M. Choi, S. Lee, and M. Park. 2016. High level architecture (HLA) compliant distributed simulation platform for disaster preparedness and response in facility management. In *Proceedings of the winter simulation conference*, 11–14 December 2016, Washington, DC, USA.
- IoTivity. 2018. *IoTivity*. <https://www.iotivity.org>. Accessed 18 Dec 2018.
- Jain, S., and C. McLean. 2003. Simulation for emergency response: A framework for modeling and simulation for emergency response. In *Proceedings of the 35th conference on winter simulation: Driving innovation*, December 2003, New Orleans, LA, USA.
- Kamat, V.R., and J.C. Martinez. 2002. Comparison of simulation-driven construction operations visualization and 4D CAD. In *Proceedings of the Winter Simulation Conference*, 8–11 December 2002, San Diego, CA, USA.
- Karen Clark & Company. 2020. *RiskInsight® open loss modeling platform*. <https://www.karenclarkandco.com/riskinsight/>. Accessed 21 Apr 2021.
- Kerber, S., and J.A. Milke. 2007. Using FDS to simulate smoke layer interface height in a simple atrium. *Fire Technology* 43(1): 45–75.
- Koliou, M., J.W. van de Lindt, T.P. McAllister, B.R. Ellingwood, M. Dillard, and H. Cutler. 2017. State of the research in community resilience: Progress and challenges. *Sustainable and Resilient Infrastructure*. <https://doi.org/10.1080/23789689.2017.1418547>.
- Lazarov, B., G. Kirov, P. Zlateva, and D. Velez. 2015. Network-centric operations for crisis management due to natural disasters. *International Journal of Innovation, Management and Technology* 6(4): Article 252.
- Lin, S.-Y., and S. El-Tawil. 2020. Time-dependent resilience assessment of seismic damage and restoration of interdependent lifeline systems. *Journal of Infrastructure Systems* 26(1): Article 04019040.
- Lin, S.-Y., W.-C. Chuang, L. Xu, S. El-Tawil, S.M.J. Spence, V.R. Kamat, C.C. Menassa, and J. McCormick. 2019. Framework for modeling interdependent effects in natural disasters: Application to wind engineering. *Journal of Structural Engineering* 145(5): Article 04019025.
- Lin, S.-Y., L. Xu, W.-C. Chuang, S. El-Tawil, S.M.J. Spence, V.R. Kamat, C.C. Menassa, and J. McCormick. 2018. Modeling interactions in community resilience. Paper presented at the *Structures Conference*, 19–21 April 2018, Fort Worth, TX, USA.
- Liu, K., X. Shen, N.D. Georganas, A. El Saddik, and A. Boukerche. 2007. SimSITE: The HLA/RTI based emergency preparedness and response training simulation. In *Proceedings of the international symposium on distributed simulation and real-time applications*, 22–24 October 2007, Chania, Greece.

- MÄK Technologies. 2018. *MÄK high performance RTI*. <http://www.mak.com/products/link/mak-rti>. Accessed 23 Dec 2018.
- Mandiak, M., P. Shah, Y. Kim, and T. Kesavadas. 2005. Development of an integrated GUI framework for post-disaster data fusion visualization. Paper presented at the *international conference on information fusion*, 25–28 July 2005, Philadelphia, PA, USA.
- McGregor, D., D. Brutzman, and J. Grant. 2008. Open-DIS: An open source implementation of the DIS protocol for C++ and Java. Paper presented at the *simulation interoperability workshop (SIW) of simulation interoperability standards organization (SISO)*, 15–19 September 2008, Orlando, FL, USA, Paper No. 08F-SIW-051.
- Miles, S.B., and S.E. Chang. 2006. Modeling community recovery from earthquakes. *Earthquake Spectra* 22(2): 439–458.
- Miller, D.C., and J.A. Thorpe. 1995. SIMNET: The advent of simulator networking. *Proceedings of the IEEE* 83(8): 1114–1123.
- Mitsova, D. 2018. Integrative interdisciplinary approaches to critical infrastructure interdependency analysis. *Risk Analysis*. <https://doi.org/10.1111/risa.13129>.
- Nan, C., and I. Eusgeld. 2011. Adopting HLA standard for interdependency study. *Reliability Engineering & System Safety* 96(1): 149–159.
- NIST (National Institute of Standards and Technology). 2016a. Community resilience planning guide for buildings and infrastructure systems, Vol. I. National Institute of Standards and Technology, Gaithersburg, MD, USA.
- NIST (National Institute of Standards and Technology). 2016b. Community resilience planning guide for buildings and infrastructure systems, Vol. II. National Institute of Standards and Technology, Gaithersburg, MD, USA.
- NRC (National Research Council). 2011. *Grand challenges in earthquake engineering research: A community workshop report*. Washington, DC: National Academies Press.
- NRC (National Research Council). 2011. *National earthquake resilience: Research, implementation, and outreach*. Washington, DC: National Academies Press.
- OMG (Object Management Group). 2004. Data Distribution Service (DDS), Version 1.0. <https://www.omg.org/spec/DDS/1.0/About-DDS/>. Accessed 19 Dec 2018.
- OMG (Object Management Group). 2015. Data Distribution Service (DDS), Version 1.4. <https://www.omg.org/spec/DDS/1.4>. Accessed 19 Dec 2018.
- OMG (Object Management Group). 2018. *Open DDS*. <http://opendds.org/>. Accessed 19 Dec 2018.
- ONERA. 2018. *CERTI*. <http://savannah.nongnu.org/projects/certi/>. Accessed 23 Dec 2018.
- Pinelli, J.-P., M. Esteve, E.M. Rathje, D. Roueche, S.J. Brandenburg, G. Mosqueda, J. Padgett, and F. Haan. 2020. Disaster risk management through the DesignSafe cyberinfrastructure. *International Journal of Disaster Risk Science* 11(6): 719–734.
- Pitch Technologies. 2018. *Pitch pRTI*. <http://pitchtechnologies.com/products/prti/>. Accessed 23 Dec 2018.
- Powell, E.T., and J.R. Noseworthy. 2012. The test and training enabling architecture (TENA). In *Engineering principles of combat modeling and distributed simulation*, ed. A. Tolks, 449–479. Hoboken, NJ: John Wiley & Sons.
- Rathje, E.M., C. Dawson, J.E. Padgett, J.-P. Pinelli, D. Stanzione, A. Adair, P. Arduino, S.J. Brandenburg, et al. 2017. DesignSafe: New cyberinfrastructure for natural hazards engineering. *Natural Hazards Review* 18(3): Article 06017001.
- Rathje, E.M., C. Dawson, J.E. Padgett, J.-P. Pinelli, D. Stanzione, P. Arduino, S.J. Brandenburg, T. Cockerill, et al. 2020. Enhancing research in natural hazards engineering through the DesignSafe cyberinfrastructure. *Frontiers in Built Environment* 6: Article 213.
- Ray, P.P., M. Mukherjee, and L. Shu. 2017. Internet of things for disaster management: State-of-the-art and prospects. *IEEE Access* 5: 18818–18835.
- Real-time Innovations. 2018. *RTI connext DDS professional* 2014. <https://www.rti.com/products/connext-dds-professional>. Accessed 19 Dec 2018.
- Reinhorn, A.M., G.D. Manolis, and C.Y. Wen. 1987. Active control of inelastic structures. *Journal of Engineering Mechanics* 113(3): 315–333.
- Ren, A., C. Chen, J. Shi, and L. Zou. 2007. Application of virtual reality technology to evacuation simulation in fire disaster. In *Proceedings of the International conference on computer graphics & virtual reality (CGVR)*, 25–28 June 2007, Las Vegas, NV, USA.
- Risk Management Solutions. 2007. *RMSTM U.S. hurricane model*. https://www.sbafla.com/method/Portals/Methodology/ModelSubmissions/2006/2006_RMS06Standards_06_20_07.pdf. Accessed 20 Apr 2021.
- ROS.org. 2018. *ROS-introduction*. <http://wiki.ros.org/ROS/Introduction>. Accessed 9 Dec 2018.
- Sahin, A., R. Sisman, A. Askan, and M. Hori. 2016. Development of integrated earthquake simulation system for Istanbul. *Earth, Planets and Space* 68(1): Article 115.
- Team Oasis. 2021. *Oasis loss modelling framework*. <https://oasislmf.org/>. Accessed 21 Apr 2021.
- Thomas, A., C.C. Menassa, and V.R. Kamat. 2017. Lightweight and adaptive building simulation (LABS) framework for integrated building energy and thermal comfort analysis. *Building Simulation – An International Journal* 10: 1023–1044.
- VT MAK. 2018. *VR-Link*. <https://www.mak.com/products/link/vr-link#vr-link-supports-dis>. Accessed 18 Dec 2018.
- Wang, Y., C. Chen, J. Wang, and R. Baldick. 2015. Research on resilience of power systems under natural disasters – A review. *IEEE Transactions on Power Systems* 31(2): 1604–1613.
- Xie, H., N.N. Weerasekara, and R.R.A. Issa. 2016. Improved system for modeling and simulating stadium evacuation plans. *Journal of Computing in Civil Engineering* 31(3): Article 04016065.
- Xiong, C., X. Lu, H. Guan, and Z. Xu. 2016. A nonlinear computational model for regional seismic simulation of tall buildings. *Bulletin of Earthquake Engineering* 14(4): 1047–1069.
- Xu, L., C. Feng, V.R. Kamat, and C.C. Menassa. 2019. An occupancy grid mapping enhanced visual SLAM for real-time locating applications in indoor GPS-denied environments. *Automation in Construction* 104: 230–245.
- Xu, L., V.R. Kamat, and C.C. Menassa. 2018. Automatic extraction of 1D barcodes from video scans for drone-assisted inventory management in warehousing applications. *International Journal of Logistics Research and Applications* 21(3): 243–258.
- Yong, Y., and Y. Jin. 2000. Marine simulator and distributed interactive simulation technology. *Computer Simulation* 17(6): 66–68.
- Zobel, R.N., P. Tandayya, and H. Duerrast. 2006. Modelling and simulation of the impact of tsunami waves at beaches and coastlines for disaster reduction in Thailand. *International Journal of Simulation* 7(4–5): 40–50.