Simple Run-Time Infrastructure (SRTI): A Scalable, Modifiable and Extensible Platform for Simulating Natural Hazards Scenarios

Sherif El-Tawil, Department of Civil and Env. Engineering, University of Michigan, Ann Arbor, MI

Executive Summary: Simple Run-Time Infrastructure (SRTI) was developed with funding from the National Science Foundation's Division of Advanced Cyberinfrastructure as a powerful means by which to compose natural hazard simulation scenarios. The software employs a run-time infrastructure (RTI) to automate data communication between simulators, each of which models a distinct part of the overall problem. RTI is a middleware that facilitates data exchange between simulators through a publish-subscribe process. This key feature removes the direct dependency of a specific simulator on another, permitting a plug-and-play system design. Exercises with SRTI show that while its speed may be impacted by the communication time between simulators, its scalability, modifiability and extensibility provide a highly effective way to set up and run large-scale, multi-disciplinary simulations of natural hazard events.

Intended Users: SRTI is designed to cater to multidisciplinary teams of researchers who want to link their simulators to model overarching natural hazard scenarios. A key consideration in developing SRTI is ease of use by lay users. This is achieved by: 1) removing the need for compilation of the code prior to use, 2) providing users with the flexibility to add new data variables or change their format without code recompilation, 3) allowing users to conveniently add new simulators to build increasingly more complex natural hazard simulations, and 4) providing a graphical user interface designed to facilitate scenario composition.

History of SRTI: The concept generation and technical development of SRTI have been ongoing since 2016. The initial approach to the development of SRTI focused on a purely data-transmission system (SRTI v1.00.00). This version did not apply mandatory rules on the simulators for time management, which required users to embed time-related information within the transmitted messages. The new version of SRTI (v2.00.00) is designed to have substantially higher functionality by supporting time-dependent simulations, for which time synchronization between multiple simulators and phase changes are required. SRTI is opensource. The software, extensive documentation and examples can be found on the SRTI GitHub site (SRTI, 2019).

Platform Architecture: A high-level view of the architecture of the SRTI v2.00.00 is shown in Figure 1. SRTI is comprised of the following components:

RTI Management Server: is a server-client application that manages the messages published by and subscribed to by each simulator. This design avoids direct integration of simulation models, which usually depends on the compatibility of programming languages. Instead, it replaces that with a dependency on messages thereby allowing a plug and play approach. The RTI Management Server also handles time synchronization and controls the activation and deactivation of the simulators at different stages of the analysis.

RTI Wrapper: Using the RTI Lib API, the RTI Wrapper serves as a bridge between the RTI Management Server and the user's simulator code. It reads and writes data from the simulator during execution and shares it with the larger simulation. Currently, RTI Wrapper versions for Java, NetLogo, Matlab, and Python have been prepared for public access. In general, the simulator code and RTI Wrapper do not need to be edited or recompiled by the user.

RTI Lib API: are a set of standard libraries that handle the connection protocols between the RTI Management Server and the RTI Wrapper.

Config.json: The names of relevant variables and functions inside the user's simulator can be defined externally inside a configuration file and then read as input at the beginning of the RTI

Wrapper's execution. The configuration file is saved in JSON format and consists of parameters that can be defined by the user. Control over the time step and simulation stage can be automated through the RTI Management Server by defining relevant parameters inside a configuration file before the start of a system execution.

GUI: The SRTI Graphical User Interface (GUI), shown in Figure 2, has two primary functions: 1) it helps define Config.json files for each RTI Wrapper, and 2) allows execution of the simulation system from a single place (within the GUI itself). Unlike the RTI Management Server and RTI Wrapper, the GUI is optional. A simulation system's configuration files can be edited by hand, and individual RTI Wrapper instances (for each simulator) can be started manually. However, the GUI greatly improves the workflow. The GUI is treated as an additional simulator from the perspective of the RTI Management Server, one that subscribes to all messages. This approach is in line with the philosophy of the SRTI: to be open-source, easy-to-use and easy to extend, either through new apps or through support of new languages.

Simulators: are user-provided simulation models that need to be linked together via SRTI.

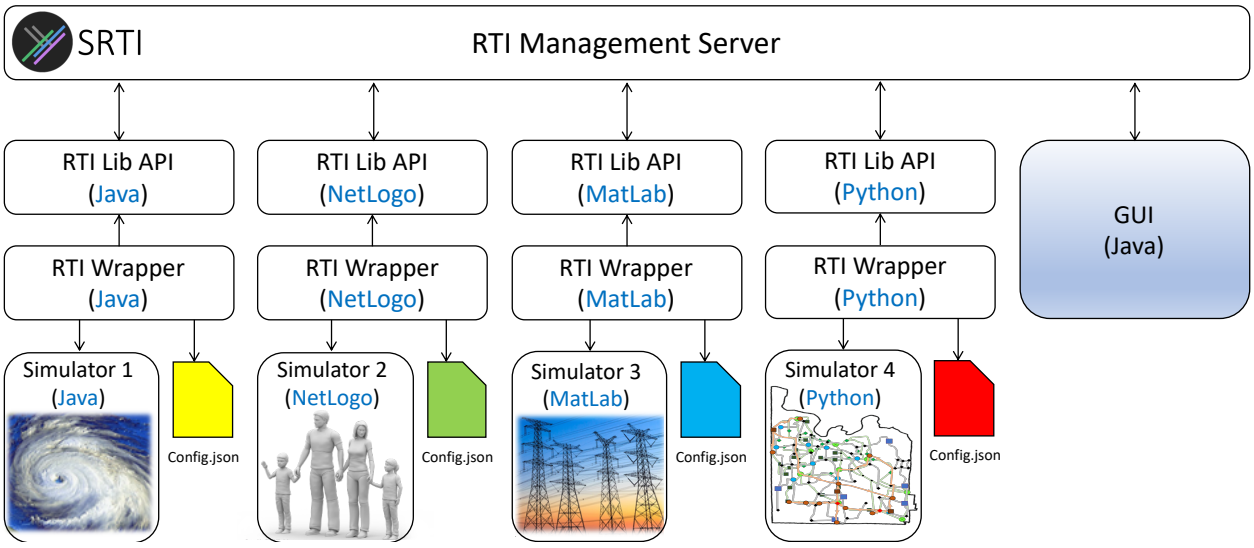

Figure 1: Architecture of SRTI

Overview of Data Structure and Requirements: As a distributed simulation tool, SRTI is primarily a data transmission software. It is therefore highly versatile in handling different data types. At present, the RTI Wrapper can handle translating and sending simulator variables in the following formats: Boolean, integer, double (decimal-precision numbers), strings (text), and multi-dimensional arrays of any of the previously listed formats.

User Experience and Software Execution: The pre-compiled SRTI files can be downloaded from the public GitHub Site (SRTI, 2019). The download includes the SRTI Server, RTI Wrapper, and SRTI GUI, all of which can execute without explicitly installation (uncompressing files and setting system paths, etc.). In addition to saving the SRTI Server and GUI in a file directory on their local machine, the user needs to prepare individual folders for each simulator that contain the executable file of the simulator and the corresponding Wrapper. After preparing the file system, the next step is to launch the SRTI GUI.exe whose layout is shown in Figure 2. Simulators/messages can be created and defined through the GUI. Then, users can click the objects on the Object List to add well-defined simulators or messages to the Canvas for a given

stage of the analysis. Action toggle buttons can be used to define the publish/subscribe relationships between simulators and messages. All of the defined simulators, messages, and publish/subscribe relationships can then be saved and imported and edited in the future.

Other features of the SRTI GUI include being able to display the content of messages and system timestep in the inspector panel (Figure 2) to better trace the system's execution outside the individual simulators. In addition, while the simulation is running, the color of the simulator objects on the Canvas will change to signify a change in execution status. Blue indicates simulators that are waiting for the messages that they subscribe to, and red indicates simulators running their internal calculations.

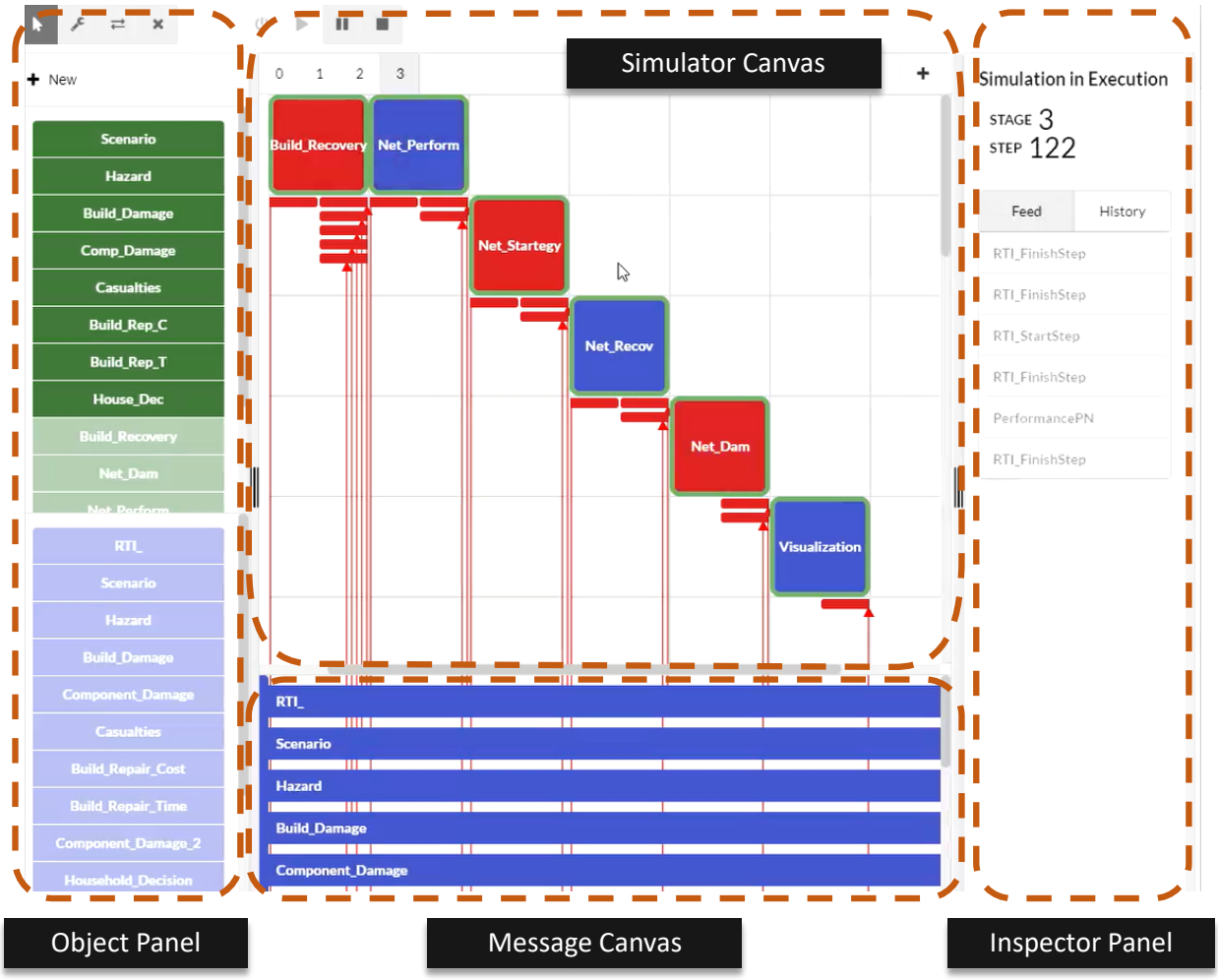

Figure 2: SRTI graphical user interface (GUI)

Illustrative Example: The capabilities of SRTI are demonstrated through an interdisciplinary (i.e., engineering and social science) multi-Language (i.e., simulators written in different programming languages) simulation example of Shelby County, Tennessee subjected to a M_w 7.7 earthquake. The example shown in Figure 3 consists of fourteen simulators each of which represents a different aspect of the scenario. The shown example combines the effect of building damage and power network damage to integrate the effect of household decisions in community resilience. Thirteen of the developed simulators are implemented in MATLAB while the other simulator (i.e.

Visualization Simulator) is implemented in NetLogo to show the ability of the SRTI to connect simulators across multiple languages.

As shown in Figure 3, the simulation framework is divided into four different stages. In stage 0, the *Scenario Simulator* broadcasts the attributes of the studied community. During stage 1, the *Hazard*, *Network Damage*, *Network Performance*, *Building Damage*, *Component Damage*, and *Casualties* simulators step through time (time step in sub-seconds) to simulate real-time seismic damage and losses associated with the earthquake. In stage 2, the *Repair Cost*, *Downtime*, and *Household Decision* simulators run for one time step to evaluate the final seismic losses resulting from the earthquake. During stage 3, the *Network Strategy*, *Network Recovery*, and *Building Recovery* simulators step through time (time step in days) to simulate the real-time recovery of the community from the earthquake. Figure 2 shows one instance of the GUI during the simulation (Stage 3, Step 122). Files for the shown example are documented and publicly available in the DesignSafe-CI repository (Sediek et al. 2020).

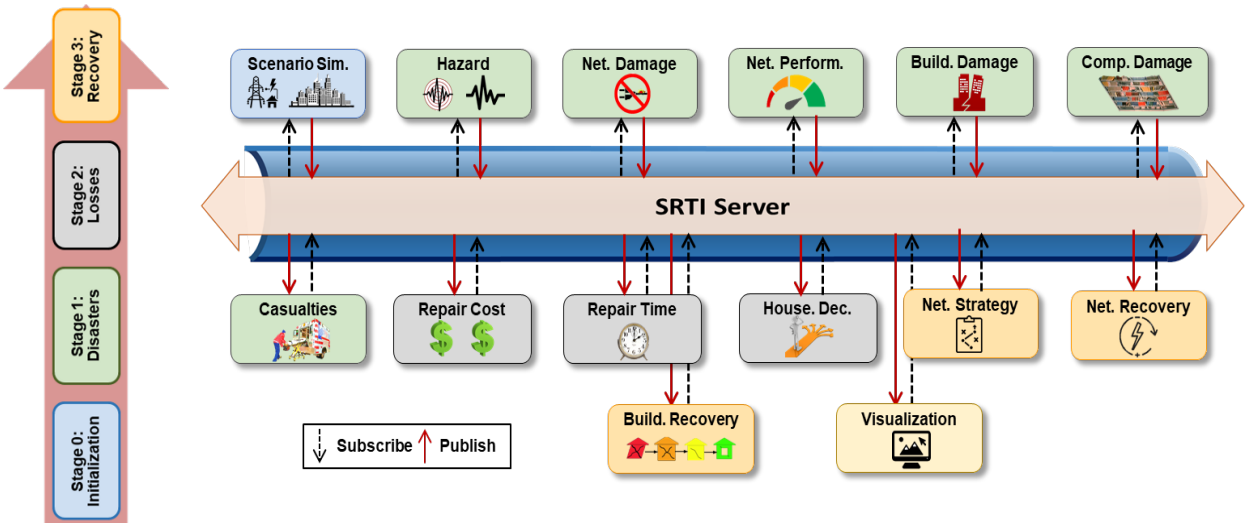

Figure 3: Simulation framework of cross-language example

Acknowledgment: A multidisciplinary team with members from civil engineering, computer science and social science developed SRTI. Members of this team include Sherif El-Tawil (project PI), Andrew W. Hlynka (lead developer), Szu-Yun Lin, Omar Sediek, Ahmed Abdelhady, Lichao Xu, Hao Lu, Seymour M.J. Spence, Jason McCormick, Vineet R. Kamat, Atul Prakash, Carol C. Menassa and Benigno Aguirre.

References and Links:

SRTI (2019). Simple Real Time Infrastructure (SRTI): Source code, Documentation and Examples, https://github.com/ICoR-code/SRTI

Sediek, O.A.; Lin, S.Y.; Hlynka, A.; El-Tawil, S. McCormick, J. (2020) “Interdisciplinary Multi-Language Community Resilience Simulation using Simple Run-Time Infrastructure (SRTI).” DesignSafe-CI. https://doi.org/10.17603/ds2-ycra-9v54