inventions

MDPI

Article

Gorilla: An Open Interface for Smart Agents and Real-Time Power Microgrid System Simulations

Carlos J. Vélez-Rivera, Fabio Andrade *, Emmanuel Arzuaga-Cruz and Agustín Irizarry-Rivera

Electrical and Computer Engineering Department, University of Puerto Rico at Mayagüez, Call Box 9000, Mayagüez 00680, Puerto Rico; carlos.velez99@upr.edu (C.J.V.-R.); emmanuel.arzuaga@upr.edu (E.A.-C.); agustin@ece.uprm.edu (A.I.-R.)

* Correspondence: fabio.andrade@upr.edu; Tel.: +1-787-966-5427

Received: 1 July 2018; Accepted: 20 August 2018; Published: 27 August 2018

Abstract: A recurring issue when studying agent-based algorithms and strategies for Power Microgrid Systems is having to construct an interface between the agent domain and the electrical model domain being simulated. Many different tools exist for such simulations, each with its own special external interface. Although many interfacing efforts have been published before, many of them support only special cases, while others are too complex and require a long learning curve to be used for even simple scenarios. This work presents a simple programming application interface (API) that aims to provide programming access to the electrical system model for any real-time simulation tool, from any agent-based platform, or programming language. The simplicity of the interface stems from the assumption that the simulation happens in real-time and the agent domain is not being simulated. We propose four basic operations for the API: read, write, call, and subscribe/call-back. We tested these by supporting two examples. In one of the examples, we present a creative way to have the model access libraries that are not available in the simulated environment.

Keywords: agent-based algorithms; IEEE 13-bus distribution; programming application interface

1. Introduction

The fusion between the power system, computers, intelligent sensors, and communication networks within the concept of the Smart Grid has created a great research niche to meet new technological challenges, such as Power Big Data, resilience, transitive markets, Smart Microgrids, active networks, etc. The need to monitor and act on each node of the power system in real time to always keep the power system healthy, as well as being attentive to natural or human threats (e.g., cyber-attacks, hurricanes, etc.) requires a growing computational infrastructure. The Smart Grid has been conceived as a sophisticated, complex, dynamic network of intelligent power infrastructure elements that work together to deliver affordable high-quality power to consumers. Individual elements are designed to tolerate failing elements to some extent [1], but there are practical limits [2] to the amount of failures that can be absorbed by the system before operations, security, and/or power quality are impacted. Therefore, the introduction of new elements to the system will have to be preceded by rigorous testing processes if a healthy rate of innovation is to be maintained. That is why the development of new technological devices for management and control of the grid needs to account for a complex and dynamic range of scenarios, with a plethora of other heterogeneous devices and technologies from multiple vendors, in an affordable way. Using only real equipment to support these efforts in most cases is not feasible for important reasons, namely the following:

- buying, installing, and configuring all equipment might be prohibitively expensive;
- some equipment might still be under development and might need to be accounted for with mathematical models;

- consistent replication of each possible scenario with real equipment, in particular, those involving failing or misbehaving devices, becomes unpractical because it is not designed to support such scenarios; and,
- continuous development and manufacturing processes become timewise inefficient if run in real-time.

Simulating real equipment with software tools works to overcome these limitations to allow for the provision of smart services and the development of many solutions (in the form of software components) that must communicate and coordinate among them. These components can be installed centrally or distributed among different operator servers.

Sophisticated simulation environments for the power systems domain exist today (e.g., DigSilent, PSCAD/EMTDC, OpenDSS, PowerWorld). However, a key difference between the smart grid and the conventional power grid is the addition of a robust communication network, supported by a range of actors implementing specialized protocols for control and management, which is commonly referred to as information communication technology (ICT). Although specialized simulation tools for the ICT domain were also widely available (e.g., Network Simulator, OMNeT++, OPNET Modeler®) before the advent of the Smart Grid concept, concerted efforts that follow a more holistic approach of either simulating both aspects of the grid with the same tool or integrating separate simulations for each aspect into a single environment are more recent (e.g., [3–8]). In [9], a comprehensive survey of these simulation and co-simulation tools is presented. A survey of current simulation techniques can be found in [10].

While simulations offer the ultimate level of control over all aspects of the problem being studied, there might be significant costs that are associated with it. For instance, a model for each piece of equipment and infrastructure to be simulated must be constructed first. The more precise these models are, the more expensive it usually becomes to build them. Since imprecise models can limit the gains attained by the simulation, their reusability, interoperability, and expandability becomes particularly important aspects of any successful simulation environment due to the need to amortize the initial cost of development. An approach called federated co-simulation is based on using standard interfaces for distributed simulations to exchange synchronization and state information among themselves. Each federated simulation models a particular piece or aspect of the system, while it uses a standard interface to stay integrated to the rest. This allows for different parts of the overall simulation to be seamlessly plugged in as needed, allowing for the reuse of complex models in a variety of scenarios and mitigating the initial cost of developing such models. IEEE Standard 1516 [11–13] defines the High-Level Architecture (HLA) used by many tools following this approach. HLA provides a very complete and sophisticated framework for managing a vast array of aspects of simulation interfacing. Implementing an HLA conforming simulation, therefore, adds some cost on top of the base effort to build the model itself. Although it is hoped that reusing big parts of the simulation through federation will account for the additional effort and offer significant extra savings, a significant learning curve and initial implementation overhead might make this approach too costly for small research projects.

The need for simulating a system in real-time often arises when real equipment needs to be integrated into the simulation, because a reasonable reusable model is not available and a compromise between cost and precision cannot be achieved when building a new one. Techniques called hardware-in-the-loop (HIL) and software-in-the-loop (SIL) are used to have real hardware and software actors interact with real-time simulations when needed. Such actors often need read/write access to simulation model variables, as they play roles, such as sensors or control sub-systems. Researchers need to routinely develop ad-hoc interfaces for this purpose, as in [14–17]. Since the interface itself is often a secondary topic, researchers usually tailor it specifically for the experiments, simulation tools, computing platforms, or programming languages at hand, as in [18]. The authors in [19] make a case for the need of a reusable tool for this purpose and offer details of the design and even source code, but they do not publish a specific programming interface for others to reuse and adapt to the various

simulation tools that are available. As in the case of federated co-simulation, the definition of an API is critical for reusability.

This paper presents an open application programming interface (API) that aims to bridge HIL or SIL actors and real-time simulations for the development of smart grid technologies with an overhead and cost that is significantly lower than that afforded by federated co-simulation. The approach should be particularly useful for small research teams or short lead-time projects. The work is based on an open environment developed by the authors that builds on existing open software components to facilitate the development of agent-based algorithms and strategies for distributed smart grid control and presented in [20]. The environment uses established open standards for data representation, communication, algebraic operations, and scalable data distribution for intelligent agents. The proposed API represents an important expansion that aims to facilitate interaction between actors in the ICT domain and real-time power system simulations for the development of smart grid technology.

This work focuses on the following:

1. defining the core operations that should support most types of interaction;
2. proposing a set of principles for managing various aspects of the process;
3. an initial API binding for the Java™ programming language; and,
4. initial test cases exercising the API.

Our emphasis has been on supporting this use-case with maximum simplicity and reusability. We present results for two experiments using the environment as a way to exercise it.

We use Matlab, SimuLink and a dSpace real-time simulator for the power system simulation and our environment based on Java™ Agent Development Environment (JADE), the JScience library, the Java Universal Network/Graph Framework (JUNG), and the Java™ programming language.

2. Proposed Application Programming Interface (API)

This section presents the API where it fits in the overall development environment and its operations in detail. It also shows programming examples and discusses the advantages of using it.

2.1. Goals and High-Level Architecture

The goals of the proposed API can be summarized as follows:

1. provide cross-domain, bidirectional access to data;
2. support cross-domain, synchronous and asynchronous interactions;
3. extensible to support all language bindings, computing and simulation platforms and networking technologies; and,
4. support flexible data typing

We chose frugality and simplicity over sophistication when designing the API in an effort to make it as easy to use as possible. This is critical in order to motivate others to incorporate it in their projects, since it shortens its adoption learning curve. For that reason, the supported operations are limited to the following:

- read/Write—Cross-domain reading/writing of variables;
- call—Cross-domain synchronous transfer of control; and,
- subscribe/Call-back—Cross-domain asynchronous event notification.

These operations cover the first two objectives above. The third goal is covered by the architecture of the tool, as shown in Figure 1. The left half of the figure depicts how the agent domain is organized, while the right half does the same for the real-time simulation domain. Software modules that were produced by the user of the platform are coloured in violet, namely the Distributed Smart Grid Control

Application (CA) module and the Simulated Model (SM) module. Platform software modules are coloured in blue, namely the Platform Interface (PI) module and the Simulation-specific Integration Code (SIC) module. The PI module implements both ends of a client-server model, assuming either role (client or server), depending on the domain of the caller. For example, agents can use a local instance of a PI module in its client role in order to contact a remote instance of a PI module in its server role to request a specific interaction with the model being simulated on that host. Activity in the opposite direction happens in a different way, due to how external interfaces for simulation tools usually work. The SIC module registers with the Proprietary Simulation External Interface (EI) module to be called back in specific scenarios or conditions that might arise within the model during the simulation. The model can be modified to set the parameters for the external interfaces to pick up when the condition is met. Call-back mechanisms are activated and the integration code requests interaction with remote or local agents using the local PI instance in its client role.

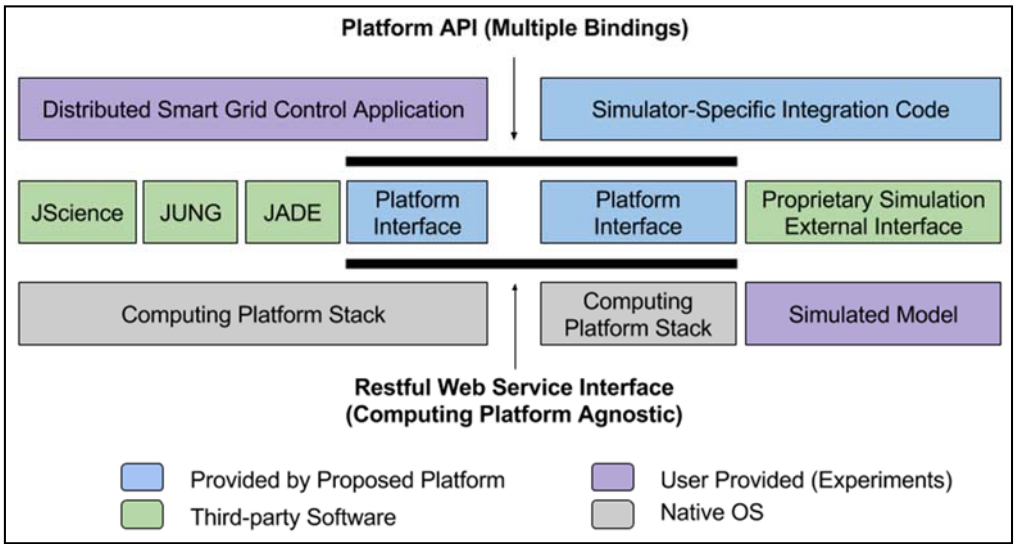

Figure 1. Platform Software Architecture.

Modules that were provided by third parties are coloured in green, namely JScience, JUNG, JADE, and EI. The functionality of this type of modules, in general, will depend on the choice of factors, such as agent platform, simulation tool, and data representation. Only the EI module has a direct impact on the platform, however, since the SIC module must interact directly with it to enable access to the model being simulated on behalf of the PI module. Finally, computing platform software modules are represented by boxes coloured in grey, labelled Computing Platform Stack. Black bars above and below the PI modules represent the Platform API and the platform's Restful Web Services Interface, respectively. The Platform API will eventually support multiple programming language bindings, while the web service interface allows for platform-agnostic inter-PI communication.

2.2. Operation Specification

This section provides the signature for the Java™ language binding of the API and discuses some basic concepts that are related to its design. Figure 2 shows the declaration of the related interface classes and a stub for a call-back class.

The GModelInterface interface class provides the core operations. Note that variable names and values are typed as UTF-8 [21] strings. This allows for arbitrary variable naming and data serialization schemes to be used on top of it. The Simulation-specific Integration Code (SIC) module is responsible for passing the variable names and values to the simulation via external interfaces. Syntax and semantics for variable names and values can be optionally adapted by the SIC; in our test cases,

Matlab/SimuLink hierarchical variable naming convention for electrical circuit models and the Matlab literal syntax rules for values.

```
public interface GModelInterface {

    String read(String variableName);

    boolean write(String variableName, String valueToWrite);

    boolean subscribe(String eventName, GCallback callback);

    String call(String functionName, String inputArguments);
}

public interface GCallbackInterface {

    int getPort();

    java.net.InetAddress getAddress();
}

public class GCallback implements Runnable, GCallbackInterface {…}
```

Figure 2. Application programming interface (API) Operations Signature.

2.3. Read/Write Operations

The read operation takes as argument a string with the name of the variable to be accessed from the model and returns a string with the value of the variable. A null string is returned in the case of an error. The write operation takes as arguments the name of the variable to access and the value to write to it. It returns a Boolean that is true if the operation was successful and false otherwise.

2.4. Call Operation

The call operation works as a generic Remote Procedure Call (RPC) mechanism. It receives as arguments two strings: one with the name of the function to call at the remote end, and the other with the input parameters separated by commas or newlines. The operation returns the output parameters that are returned by the remote function call, separated by commas or newlines.

2.5. Subscribe Operation

The subscribe operation receives an event name and a GCallback object. When the corresponding event happens in the model, a socket connection is established to the Internet Protocol (IP) address and port combination that is provided by the GCallback object through the GCallbackInterface interface and a string consisting of the event name and input parameters is streamed through it, separated by commas or newlines. Implementing the Runnable interface allows for the object to decide what to do when the call-back is received. This object must create a server socket, bind to the appropriate port and IP address and listen for connections that are associated with call-backs. The event name is used in the simulation to trigger the event and send input parameters, if any.

Existing operations are extensible through polymorphism and new operations can be added to the API when needed.

3. Applications Test Cases

This section presents two sample applications where the operations that are provided by the platform are exercised.

3.1. Case 1: Agent-Based Voltage Control in a IEEE 13-Bus Distribution Test Feeder

In this case, intelligent agents will be called upon to enable stand-by capacitive load on a bus of a power distribution network whose voltage has fallen below acceptable levels. The choice is made by agents, which first read the bus voltage and load current for each node to estimate the actual power load in each case. Agents read these values from the model by invoking the read operation of the PI. A power flow solver is then used to compute theoretical steady-state voltages for all buses, for each possible level of selectable capacitive load. The first capacitive load setting that will theoretically bring all bus voltage levels within an acceptable range (5% of rated values) is selected as the solution to the problem and the corresponding switches in the model are actuated by writing to appropriate the model variables.

The power distribution system, an IEEE 13-bus test feeder [22], is studied. The feeder is modelled in Matlab/Simulink and run on a dSpace real-time simulator. Figure 3 shows the complete environment that is built for this experiment. It consists of three major components, namely a real-time simulator, a controller personal computer, and a server for hosting intelligent agents. Although, in some cases, it is possible to avoid using an intermediate host, we decided to use it in these initial tests to avoid overrun situations on the simulator, since everything on it must run in lock-step with the simulation. Supporting some of the I/O operations that are required for the PI might need additional consideration. The two SIC handlers (Embedded and Host) are python scripts that use the platform-specific event handling framework of the simulation tool to connect it to the external world. Host call events are fired by the embedded handler at frequent intervals, when certain conditions in the model are met or when an explicit request is made by the model. The host handler should get back to the embedded handler with results as fairly quickly, so it is limited to two quick, simple operations:

- queue new or processed read, write, subscribe or call-back transactions to the SIC transaction out queue; and,
- de-queue new or returning read, write, subscribe, or call-back transactions from the SIC transaction in queue.

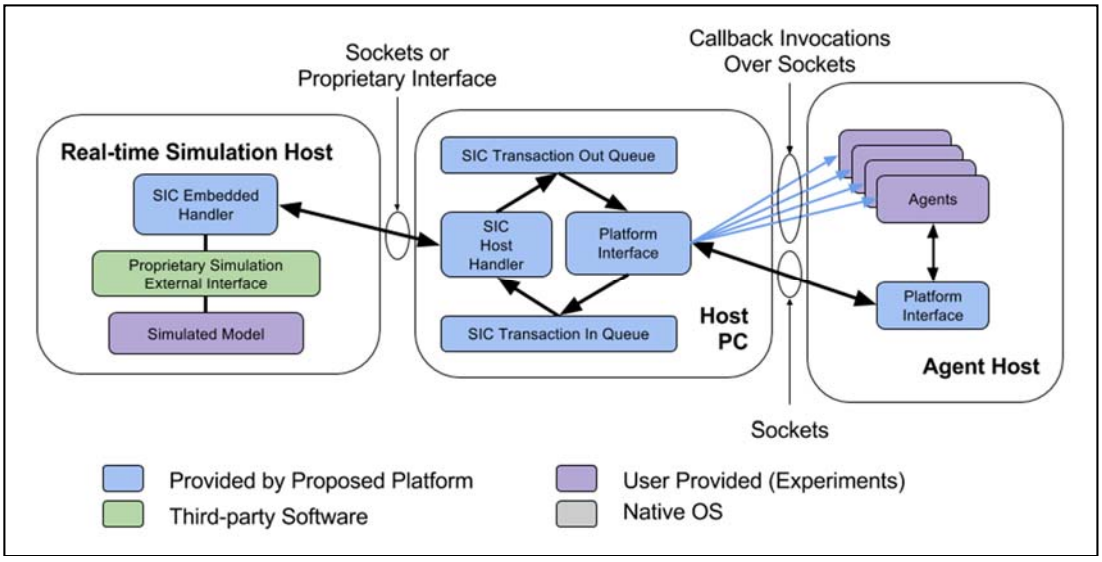

Figure 3. Case 1 Test Environment.

All the exchanges between the two handlers are initiated by the embedded handler. Frequent host calls without associated events allow for the host handler to send data to the embedded handler. A separate thread running the PI module on the host PC receives API requests or responses to

previously sent requests from other (possibly remote) PI module instances. It converts each response into a transaction and queues it in the SIC transaction in queue for the host handler to pick up. Another thread constantly checks for transactions placed on the SIC transaction out queue and converts them into either requests for other PI module instances or replies to previously received requests. PI module instances connect amongst themselves while using TCP sockets. In this initial implementation, a web service client-server duo was not used. Finally, call-back transactions result in direct socket connections to the subscribed party, as illustrated.

Note that all the components shown except for the EI module, the Simulated Model, and the Agents are provided by the proposed platform. Users of the platform are shielded from the effort that is required to implement these components by the API and do not need to understand how they work.

3.2. Case 2: Fuzzy Control for Home Microgrids

Our second sample application is a fuzzy control for a battery management in a home microgrid in a grid-connected mode [23]. The HIL-simulated microgrid has a photovoltaic generator, a bank of batteries, and several critical and non-critical loads. The microgrid control uses fuzzy logic to decide when to switch on and off non-critical loads to maintain a constant power demand from the utility grid, depending on the state of charge, the current consumption and solar irradiance. We assume that the fuzzy logic tool that is used for this solution is not supported by the real-time simulator. We then show how the proposed platform can be used as an RPC mechanism through the use of the call operation of the PI to access the unsupported functionality from a connected Matlab instance.

Figure 4 shows the environment used for this application. In this case, one of the PIs is implemented in the Matlab code. The PI itself interprets the input parameters, calls the fuzzy control function, and sends the formatted output parameters to the calling PI at the host PC. Only the call operation of the platform is exercised by this application.

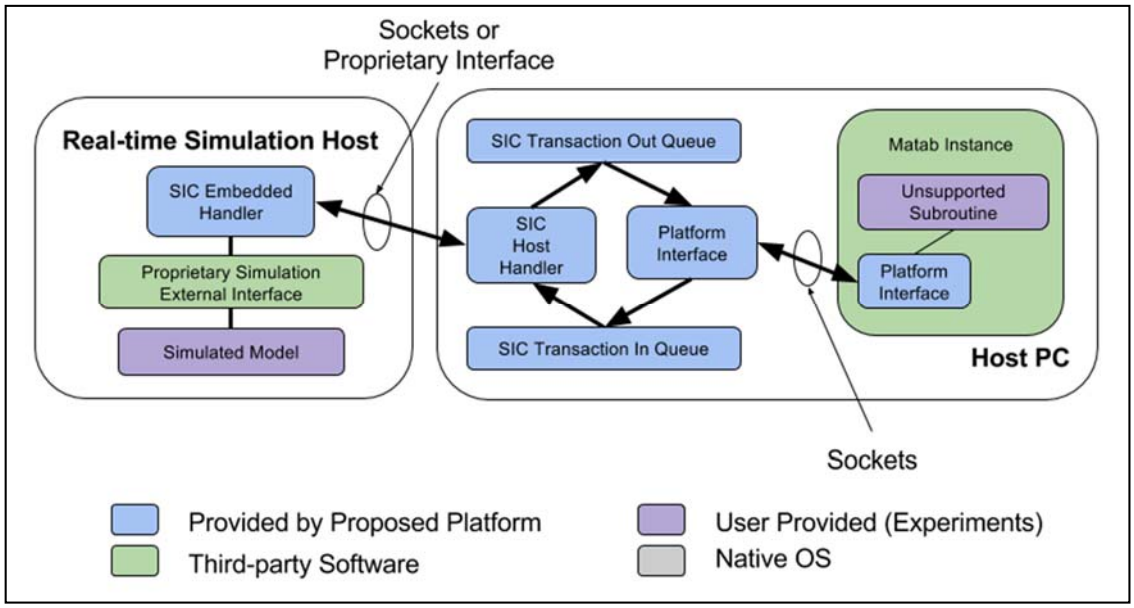

Figure 4. Case 2 Test Environment.

4. Results and Discussion

4.1. Case 1: Agent-Based Voltage Control

The IEEE 13 Node Test Feeder is very simple and is quite frequently used to test common features of distribution systems operating at 4.16 kV. It is characterized by being short; relatively highly loaded; a single voltage regulator at the substation; overhead and underground lines; two shunt

capacitors; an inline transformer; and, a total of nine unbalanced loads. The power flow solver uses the backward–forward iterative technique as explained in [14,24]. The generalized matrices for each element of the test feeder were computed and placed on a file following an extended JUNG file format.

Figure 5 shows the interaction between the agents and the real-time simulation over time and the effects of their control decisions on the voltage of phases A, B, and C of node 675 of the feeder. Voltage levels at this node are below the recommended values. Agents subscribe for the "low voltage" event (see subscribe subplot) at around ten seconds. A monitor on the model polls the voltages and triggers an event at around forty seconds. The SIC embedded module notifies the SIC host handler of the platform and the latter posts a transaction on the SIC transaction out queue. The PI on the host PC reads the transaction from the queue and calls back the registered agent while using the subscription information for the event.

The registered agent receives the call-back and starts reading voltage and current values to compute actual load conditions. This period lasts just under fifty seconds. At around second ninety-seven, the agent completes the computation of a capacitor setting, thus raising the voltage to an acceptable level. Shortly after that, it writes to the capacitor to switch variables of the model; it uses the write operation to add capacitive load to the system, raising the voltages at node 675, as can be seen in the top three subplots in Figure 5.

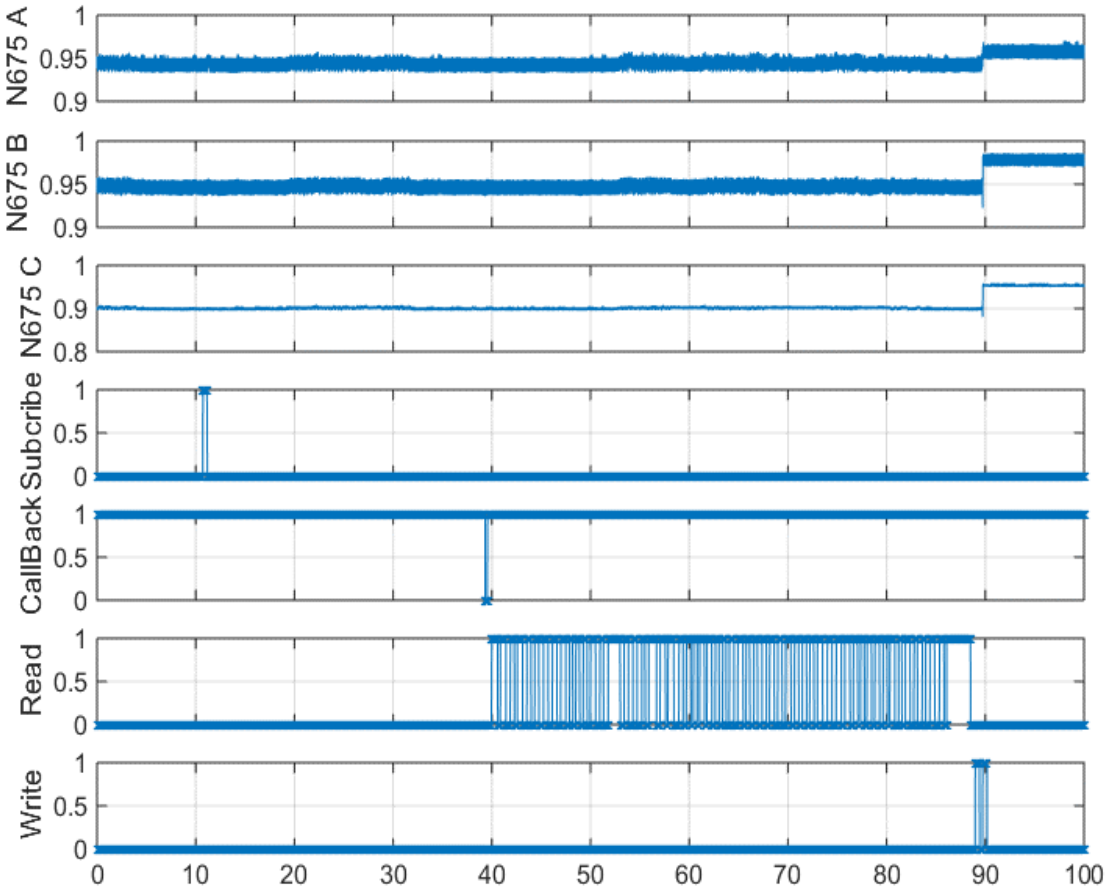

Figure 5. Interaction between agents and the real-time simulation and the effects of control actions on the voltage at node 675.

4.2. Case 2: Fuzzy Control for Home Microgrids

The Fuzzy control that is proposed in this case tries to maintain a flat consumption of the mains and to compensate for the intermittent nature of the PV generator using the storage system and the demand response function. The control must consider the SOC of the battery to prevent significant

damages due to excessive charge or discharge. The demand response activates and disconnects the controllable loads to drop the demand from the houses and maintain the power flow balance. Once the SOC starts to increase, the loads are slowly reconnected. Thus, the proposed platform performs the call operation periodically to compute incremental control changes. This operation invokes the remote fuzzy controller function on the remote Matlab instance that implements it. It receives the state of charge (SOC) and remaining battery power as input parameters. It returns the demand response actions DR1, DR2, and DR3.

Figure 6 shows the interaction of the simulation with the remote Matlab instance, as aided by our implementation of the platform. It shows the values of the input and output parameters along with the platform operation activity over time. Consider the state of the input parameters at the 30-s mark. DR values before the call operation are (1,1,1). The call operation returns values (1,0,1), and the corresponding values are set by the SIC embedded handler. This disconnects non-critical loads to balance the generation and consumption of power inside the microgrid. Battery power, however, continues to drop, driven by other factors. At the 40-s mark, another call operation is invoked, and soon after that, the DR variables are set to (1,0,0) by the SIC embedded handler, which disconnects more non-critical loads. Similar control events happen at seconds 60, 90, 100, and 110.

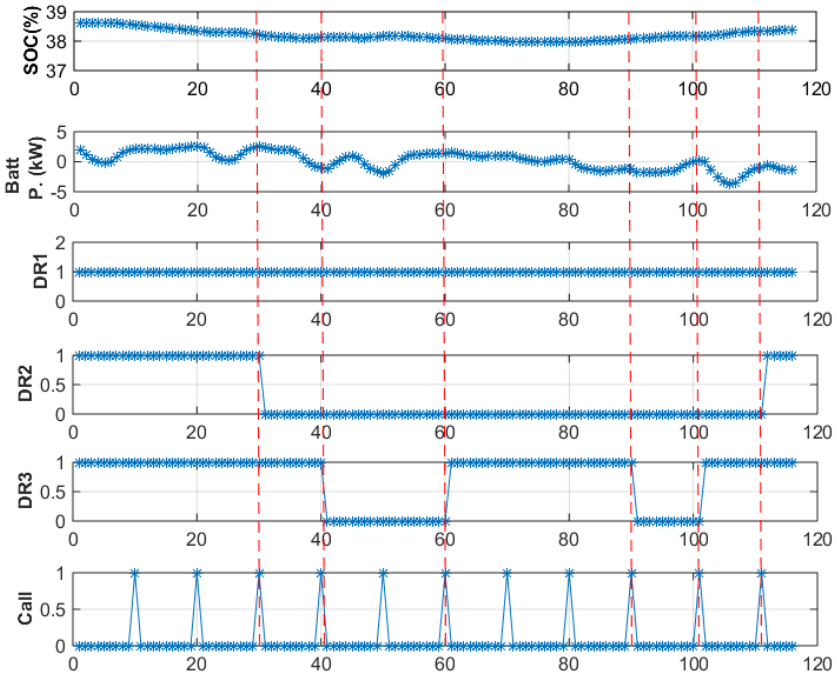

Figure 6. Input and output parameters of the fuzzy control function along with invocations of the call platform operation.

4.3. Performance Analysis of the API

This initial implementation of the client role for the PI has around 230 lines of Java™ code. The server role and the SIC for the first application have about 316 lines of python code, while the same modules for the second application have 222 lines of python code and 15 lines of Matlab code. The simplicity of the design requires little effort from its users or developers.

Each read/write access takes one second to complete the round trip from the agents to the model and back. Most of the time is spent on the SIC-embedded handler, since the polling mechanism has a sampling period of one second. Although it is possible to increase the sampling frequency, this increases the total sampling overhead, as well and can have an effect on the capacity of the system to support more complex models. This is where the external interface of the simulation tool can help

by providing a way to read model data from the host PC synchronously. Elapsed time outside of the sampling period is in the order of milliseconds.

5. Conclusions

This paper has presented a programming application interface (API) between the agent domain and the electrical domain real-time simulation model that facilitates the development and study of agent-based algorithms and strategies for the Smart Grid. The use of the API with two sample power system control applications is demonstrated. The first case is an agent-based control with the IEEE 13-nodes feeder, which uses the read and write platform to compensate a low voltage level in one of the inner nodes. The second case is a microgrid system that uses the call platform operation to manage the available energy inside the Microgrid by using a fuzzy control function. The tool easily lends itself to supporting real-time interactions with the simulated model, although some extensions are warranted. In particular, the bulk transfer of data, through the support of either or both compound or vector types, needs to be addressed. We suggest that simulation tools need to be sufficiently integrated with the modelling tool to support parameterized external event generation explicitly from the model and external asynchronous access.

Author Contributions: Conceptualization, C.J.V.-R.; Project administration, E.A.-C.; Supervision, F.A. and A.I.-R.

Funding: This material is based upon work supported by the National Science Foundation under Grant No. ACI-1541106 of the CRISP Program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Conflicts of Interest: The authors declare no conflicts of interest.

Nomenclature

Actor	Real or simulated device or entity that interacts concurrently with others in a system
Agen	A software component that exhibits authority and independence to make decisions
Application Programming Interface (API)	A set of well-defined rules for communication between applications that depend on it and the software that provides its services
Client-server model	A distributed application arrangement composed of requesters (clients) and providers (servers) of a service
Computing Platform (Software) Stack	software components providing access to resources of the host computing platform
Java™ Agent Development Environment (JADE)	An environment for the development of agent-based systems with the Java™ programming language
JScience	An open software library for math operations using Java™
Java™ Universal Network/Graph Framework (JUNG)	A framework for representing and exchanging network/graph information
Remote Procedure Call (RPC)	A mechanism to make an API provided by a remote host accessible to local applications by means of procedure invocations

References

1. McArthur, S.D.J.; Davidson, E.M.; Catterson, V.M. Multi-Agent Systems for Power Engineering Applications—Part I: Concepts, Approaches, and Technical Challenges. *IEEE Trans. Power Syst.* **2007**, *22*, 1743–1752. [CrossRef]
2. Siqueira, R.; Mohagheghi, S. Impact of Communication System on Smart Grid Reliability, Security and Operation. In Proceedings of the 2016 North American Power Symposium (NAPS), Denver, CO, USA, 18–20 September 2016.
3. Lin, H.; Veda, S.S.; Shukla, S.S.; Mili, L.; Thorp, J. GECO: Global Event-Driven Co-Simulation Framework for Interconnected Power System and Communication Network. *IEEE Trans. Smart Grid* **2012**, *3*, 1444–1456. [CrossRef]

4. Hopkinson, K.; Wang, X.; Giovanini, R.; Thorp, J.; Birman, K.; Coury, D. EPOCHS: A platform for agent-based electric power and communication simulation built from commercial off-the-shelf components. *IEEE Trans. Power Syst.* **2006**, *21*, 548–558. [CrossRef]
5. Rohjans, S.; Lehnhoff, S.; Schütte, S.; Scherfke, S.; Hussain, S. Mosaik—A modular platform for the evaluation of agent-based Smart Grid control. In Proceedings of the IEEE PES ISGT Europe 2013, Lyngby, Denmark, 6–9 October 2013; pp. 1–5.
6. Taylor, S.J.E.; Sudra, R.; Janahan, T.; Tan, G.; Ladbrook, J. Towards COTS distributed simulation using GRIDS. In Proceedings of the 33nd Conference on Winter simulation 2001, Arlington, VA, USA, 9–12 December 2001; Volume 2, pp. 1372–1379.
7. Nutaro, J. Designing power system simulators for the smart grid: Combining controls communications and electro-mechanical dynamics. In Proceedings of the IEEE Power and Energy Society General Meeting 2011 (PES '11), Detroit, MI, USA, 24–28 July 2011; pp. 1–5.
8. Nutaro, J.; Kuruganti, P.; Miller, L.; Mullen, S.; Shankar, M. Integrated hybrid-simulation of electric power and communications systems. In Proceedings of the IEEE Power Engineering Society General Meeting 2007 (PES '07), Tampa, FL, USA, 24–28 June 2007; pp. 1–8.
9. Mets, K.; Ojea, J.A.; Develder, C. Combining Power and Communication Network Simulation for Cost-Effective Smart Grid Analysis. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 1771–1796. [CrossRef]
10. Tsampasis, E.; Sarakis, L.; Leligou, H.C.; Zahariadis, T.; Garofalakis, J. Novel Simulation Approaches for Smart Grids. *J. Sens. Actuator Netw.* **2016**, *5*, 11. [CrossRef]
11. IEEE Standard for Modeling and Simulation (M&S). *High Level Architecture (HLA)—Framework and Rules*; IEEE Std 1516–2010 (Revision of IEEE Std 1516–2000); IEEE: Piscataway, NJ, USA, 2010; pp. 1–38.
12. IEEE Standard for Modeling and Simulation (M&S). *High Level Architecture (HLA)—Federate Interface Specification*; IEEE Std 1516.1-2010 (Revision of IEEE Std 1516.1-2000); IEEE: Piscataway, NJ, USA, 2010; pp. 1–378.
13. IEEE Standard for Modeling and Simulation (M&S). *High Level Architecture (HLA)—Object Model Template (OMT) Specification*; IEEE Std 1516.2-2010 (Revision of IEEE Std 1516.2-2000); IEEE: Piscataway, NJ, USA, 2010; pp. 1–110.
14. Nguyen, C.P.; Flueck, A.J. A novel agent-based distributed power flow solver for smart grids. *IEEE Trans. Smart Grid* **2014**, *6*, 1261–1270. [CrossRef]
15. Gomez-gualdron, J.G.; Velez-Reyes, M. Simulating a Multi-Agent based Self-Reconfigurable Electric Power Distribution System. In Proceedings of the 2006 IEEE Workshops on Computers in Power Electronics, Troy, NY, USA, 16–19 September 2006; pp. 1–7.
16. Pipattanasomporn, M.; Feroze, H.; Rahman, S. Multi-agent systems in a distributed smart grid: Design and implementation. In Proceedings of the PSCE 09 IEEE/PES, Power Systems Conference and Exposition, Seattle, WA, USA, 15–18 March 2009; pp. 1–8.
17. Kleinberg, M.; Karen, M.; Nwankpa, C. Distributed multi-phase distribution power flow: Modeling solution algorithm and simulation results. *Trans. Soc. Model. Simul. Int.* **2008**, *84*, 403–412. [CrossRef]
18. Andren, F.; Stifter, M.; Strasser, T.; de Castro, D.B. Frame-work for co-ordinated simulation of power networks and components in smart grids using common communication protocols. In Proceedings of the 37th Annual Conference on IEEE Industrial Electronics Society (IECON '11), Melbourne, Australia, 7–10 November 2011; pp. 2700–2705.
19. Roche, R.; Natarajan, S.; Bhattacharyya, A.; Suryanarayanan, S. A Framework for Co-simulation of AI Tools with Power Systems Analysis Software. In Proceedings of the 2012 23rd International Workshop on Database and Expert Systems Applications, Vienna, Austria, 3–6 September 2012; pp. 350–354.
20. Vélez-Rivera, C.J.; Arzuaga-Cruz, E.; Irizarry-Rivera, A.A.; Andrade, F. Global Data Prefetching Using BitTorrent for Distributed Smart Grid Control. In Proceedings of the 2016 North American Power Symposium (NAPS), Denver, CO, USA, 18–20 September 2016.
21. Yergeau, F. UTF-8, a Transformation Format of ISO 10646. STD 63, RFC3629. November 2003. Available online: https://datatracker.ietf.org/doc/rfc3629/ (accessed on 27 August 2018).
22. Kersting, W. Radial distribution test feeders. In Proceedings of the IEEE Power Engineering Society Winter Meeting, Columbus, OH, USA, 28 January–1 February 2001; pp. 908–912.

23. Lebron, C.; Andrade, F.; O'Neill, E.; Irizarry, A. An Intelligent Battery Management System for Home Microgrids. In Proceedings of the 7th Conference on Innovative Smart Grid Technologies, Minneapolis, MN, USA, 6–9 September 2016.
24. Kersting, W. *Distribution System Modeling and Analysis*, 3rd ed.; CRC Press: Boca Raton, FL, USA, 2012.