IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 23, 2018, accepted March 1, 2018, date of publication April 9, 2018, date of current version May 2, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2824341

Co-Simulation of Distributed Smart Grid Software Using Direct-Execution Simulation

CHONG SHUM[1], (Student Member, IEEE), WING-HONG LAU[1], (Senior Member, IEEE), TIAN MAO[2], (Student Member, IEEE), HENRY SHU-HUNG CHUNG[1], (Fellow, IEEE), KIM-FUNG TSANG[1], (Senior Member, IEEE), NORMAN CHUNG-FAI TSE[1], (Member, IEEE), AND LOI LEI LAI[3], (Fellow, IEEE)

[1]Centre for Smart Energy Conversion and Utilization Research, City University of Hong Kong, Hong Kong
[2]Electric Power Research Institute, China Southern Power Grid, Guangzhou 510623, China
[3]School of Automation, Guangdong University of Technology, Guangzhou 510006, China

Corresponding author: Wing-Hong Lau (itwhlau@cityu.edu.hk)

This work was supported by Hong Kong RGC under Grant CityU 116013.

ABSTRACT The use of distributed computation and control is pervasive for a wide range of smart grid research topics. However, recent developments of smart grid co-simulation platforms have not been able to provide effective support for the modeling and simulation of distributed software systems. In particular, co-simulation literatures only focused on the integration of electrical and communication network simulators, and the responsibility for modeling distributed software is often delegated to one of these two simulators. Since these domain specific simulators are not designed for this purpose, such delegation incurs many limitations that prevent convenient, effective, and accurate modeling of software behaviors. To mitigate the problem, this paper presents, to our knowledge, the first co-simulation integration of direct-execution simulators to provide dedicated support for distributed smart grid software. We first present the development of the novel DecompositionJ framework (DEterministic, COncurrent Multi-PrOcessing SImulaTION for Java programs), which is a compiler-based code analyzer and transformer to automatically convert multi-thread Java programs into direct-execution simulators, eliminating the need for manual code or model development. Next, we apply DecompositionJ framework to generate simulators for a popular multi-agent platform JADE. The JADE simulators are then integrated with electromagnetic transient simulator (PSCAD) and packet-level network simulator (OPNET) using standardized co-simulation runtime infrastructure. At last, we conduct a case study on agent-based smart grid restoration using this novel co-simulation platform. Through the analysis of simulation results, it is shown that the proposed direct-execution simulation framework is able to facilitate the understanding, evaluation, and debugging of distributed smart grid software.

INDEX TERMS Smart grid co-simulation, direct-execution simulation, high-level architecture, power system, communication, distributed systems, multi-agent system, power system restoration.

I. INTRODUCTION

The rapid decentralization of power system is driving revolutionary changes in its underlying ICT infrastructures. Various smart grid applications require distributed computation and pervasive data communication for coordinating millions of automated devices. The study of these smart grid applications often requires performance evaluation, either via experimentations on real-world testbed or via simulation. Unfortunately, real-world platforms are rarely available for the purpose of experiments due to the costs and the critical nature of power stability. Even if a testbed is available, experiments are usually bounded to a small scale with a limited set of scenarios. Furthermore, experimental results are often difficult to repeat due to varying operational conditions. In contrast, simulation enables fully configurable experiments with arbitrary hypothetical scenarios at a low cost and the results are easily repeatable. It is therefore of no surprise that most published results are obtained via software-based simulations.

For a comprehensive simulation of smart grid, all three aspects of electrical, communication and distributed computation, as well as their inter-dependencies should be modeled.

Given the complexity and heterogeneity of the smart grid, creating a comprehensive simulator from scratch would be costly and time consuming. Therefore, the co-simulation designs are often implemented by combining existing well-developed and validated domain-specific simulators, e.g., power system and communication network simulators, with each of them responsible for modeling and simulating some aspects of the smart grid.

This smart grid co-simulation paradigm was pioneered by the work of Electrical Power and Communication Synchronizing Simulator [3] (EPOCHS), and later works [4]–[16] contributed to this field mainly in three directions:

1) Exploring various combinations of power system and communication simulators for integrating and expanding the tool sets to support different smart grid applications.

2) Improving time synchronization mechanisms. This is to ensure that simulation events and messages across all simulators are processed in correct timestamp order, such that simulation outcomes are causally correct and repeatable.

3) Improving the interoperability of co-simulation platforms by conforming to standardized co-simulation frameworks such as Distributed Interactive Simulation (DIS, IEEE 1278 Std.) and High Level Architecture (HLA, IEEE 1516 Std.).

Despite previous advancements, discussions are lacked on the modeling and simulation of distributed computation. Most previous works simply delegated such responsibility to either electrical or communication simulators, leading to several problems:

- Electrical/communication simulators allow creation of custom models via ‘‘user-codes". However, user code development is not as flexible as general programming, typically subject to many limitations such as reduction of usable language features, libraries and interfaces. In particular, multi-threading and blocking calls are generally forbidden to prevent non-deterministic execution and deadlocks.
- Due to user code limitations, existing software cannot be simulated in its deployable form. Researchers need to rewrite the software with respect to user-code constraints. Depending on the complexity of target software, development process can be time consuming and labor intensive.
- Due to user-code constraints and development costs, researchers may need to discard non-essential parts of the software which may result in loss of functional fidelity.
- Electrical/communication simulators do not provide mechanisms to model computation delays which cause loss of timing fidelity.
- Separation of deployment and simulation codes complicates version control and code maintenance.

To mitigate these problems, we delegate the simulation of distributed software to *direct-execution simulators*, in conjunction with the typical integration of electrical and communication simulators. In a direct-execution simulation, the original code of the target program will be executed to emulate its own functional behavior. Additional simulation control codes are inserted to execute alongside the target codes, which are responsible for: i) redirect program's interactions with real systems, i.e., I/O, system clock, timers, to their simulated counterparts; ii) determine computation delays and track the logical timestamp of program's actions; and iii) control the order of execution over actions performed by different threads within the simulator, and the order of execution over events across co-simulators.

In this paper, we discuss the challenges and operating principles of a direct-execution simulation, then provide an overview on the design of DecompositionJ simulation framework (DEterministic, COncurrent Multi-PrOcesssing SImulaTION for Java programs). This framework performs compiler-based source analysis on a target Java program, then automatically transforms the original program into a direct-execution simulator by instrumenting simulation control codes. The proposed framework eliminates the need for manual code modification, hence significantly reduces the cost for developing and maintaining a model for the target software, yet at the same time fully retains its functional fidelity. Furthermore, this framework does not require new hardware support and programmer annotation, therefore is highly compatible with existing Java execution environments and development tools. This allows researchers to use mainstream IDEs and debuggers to investigate and debug the target software during simulation. Further details on the framework, such as definitions of the simulation model, exploitation of parallelism, scalability and performance evaluations, can be found in our paper [27].

To demonstrate the usefulness of our framework, we apply the direct-execution simulation techniques on a popular multi-agent platform JADE (Java Agent Development Framework) [26]. The DecompositionJ framework is used to analyze and transform JADE source code to produce direct-execution simulators, which are then integrated with power system simulator PSCAD and network simulator OPNET via a runtime infrastructure (RTI). The entire co-simulation platform adheres to the HLA standard. Using this co-simulation platform, a case study on agent based fault location, isolation and service restoration (FLISR) has been conducted. Through the analysis of simulation results, it is shown that the proposed direct-execution simulation framework is able to facilitate the understanding, evaluation, and debugging of distributed smart grid software. The rest of this paper is organized as follows: Section II discusses and compares related works in smart grid co-simulation with our proposed framework. Section III presents the operating principle, design, and implementations of DecompositionJ framework. Section IV discusses an HLA-based co-simulation platform that integrates electrical simulator - PSCAD, communication simulator - OPNET, and DecompositionJ simulators. Section V presents the FLISR case study, and results are shown in Section VI. Conclusion is given in Section VII.

II. RELATED WORKS

The software-based co-simulation paradigm in smart grid research was pioneered by the work of Electrical Power and

TABLE 1. Characteristics of related works in smart grid co-simulation.

Related Works	Year	Example Use Case	Power	Commun.	Computation	Synchronization	Interface
EPOCHS [3]	2006	Agent based WAMPAC	PSCAD, PSLF (Transient)	NS-2	External Process	Time-stepped	Adhoc
Nutaro *et al.* [4]	2007	WAMPAC	Adevs (Steady state)	NS-2	NS-2	Master-slave	Adhoc
Nutaro [5]	2011	WAMPAC	Adevs-THYMS (Transient)	OMNET++	Adevs, OMNET++	Master-slave	Adhoc
PowerNet [6]	2011	Networked Control	Modelica (Transient)	NS-2	NS-2	Time-stepped	Adhoc
Mets *et al.* [7]	2011	DSM, DR	Simulink (steady state)	OMNET++	OMNET++	Master-slave	Adhoc
MAPNET [8]	2011	WAMPAC	MATLAB (Transient)	OPNET	MATLAB	Master-slave	Adhoc
VPNET [9]	2011	Distributed Control	VTB (Transient)	OPNET	VTB	Time-stepped	Adhoc
GECO [10]	2012	WAMPAC	PSLF (Transient)	NS-2	NS-2	Event driven	Adhoc
Levesque *et al.* [11]	2012	DSM, DR	OpenDSS (steady state)	OMNET++	OMNET++	Master-slave	Adhoc
INSPIRE [12]	2014	WAMPAC, IEC 61850	DIgSILENT (Transient)	OPNET	External Process	Event driven (Parallel)	HLA
Celli *et al.* [13]	2014	DSM, DR	OpenDSS (Steady state)	NS-2	MATLAB	Event driven	Adhoc
Perkonigg *et al.* [14]	2015	Agent based protection and control	-	OPNET	Extended JADE	Event driven (Parallel)	HLA
Li *et al.* [15]	2017	DSM, DR, AMI	GridLabD (Steady state)	CORE	GridLabD	Real-time	Adhoc
Garau *et al.* [16]	2017	Distribution protection and control	OpenDSS (Steady state)	NS-3	MATLAB	Event driven	Adhoc
This paper	2018	Agent based protection and control	PSCAD (Transient)	OPNET	DecompositionJ	Event driven (Parallel)	HLA

Communication Synchronizing Simulator [3] (EPOCHS), which combined power system simulators PSCAD (for electromagnetic transients) and PSLF (for electro-mechanical dynamics) with communication simulator NS2 (for packet-level simulation of wired/wireless communications). Simulators are interfaced to a Runtime-Infrastructure (RTI) which facilitates the exchange of data and the synchronization of event execution across simulators such that simulation events are executed in correct order according to their timestamps. The capability of EPOCHS was demonstrated with an example case study in agent based wide area monitoring, protection and control (WAMPC).

Many other co-simulation frameworks and platforms have since been proposed. Table 1 summarizes their characteristics with respect to several aspects: i) target use cases; ii) selection of simulation tools for power systems, communication networks, and distributed software; iii) time synchronization schemes; and iv) co-simulation interfaces. We discuss each of these aspects in the following.

A. SIMULATION OF POWER SYSTEMS

The selection of power system simulator depends primarily on the target use case. In general, power system simulation models can be classified into two types:

1) Steady state models of which the stable state of the power network is solved using power flow analysis. The model is typically used in power network planning, demand side management (DSM), energy markets, and optimization studies. Example simulation tools include Adevs, OpenDSS, GridLab-D, PowerWorld, PSSE, DigSilent and MATLAB.

2) Transient dynamic models of which the power network is characterized at circuit level by differential equations. The trapezoidal rule is then applied to discretize the equations such that sampling and switching events can be modeled. Simulator then solves the system equations repeatedly for each time step to obtain numerical time-domain solutions. These models are typically used to study power system control and protection, where transitions between stable states occur due to the changing of operation point. Example tools include PSCAD, PSLF, Adevs-THYMS, DIgSILENT, and MATLAB.

Since our proposed co-simulation framework targets on delay-sensitive applications that operate during state transitions (e.g., power system restoration), we select the PSCAD simulator to model the fast transients in the electrical system.

B. SIMULATION OF COMMUNICATION NETWORK

Packet-level network simulators such as OPNET, NS-2, NS-3 and OMNET++ are commonly used for simulating smart grid communications. The principles for these tools are similar: the processing and transmission of messages are modeled as a sequence of discrete events along the logical time-line. Selection of suitable simulator depends on model availability, development flexibility and licensing types (e.g., open source or proprietary).

For this paper, we selected OPNET for its rich set of wired and wireless communication models, and its built-in HLA co-simulation interface.

C. SIMULATION OF DISTRIBUTED SOFTWARE

Distributed software programs are usually modeled within the network simulator [4]–[7], [10], [11], or within power system simulator [5], [8], [9], [15], and their drawbacks have been discussed in the Introduction section. Other approaches [3], [12], [13], [16] allow the target software to be modeled as external processes and interfaced to the simulators as slaves. We do not consider these slave processes to be co-simulators because they are not time-regulating; rather, they simply perform computation and return results upon receiving triggering signals from their master simulators. Modeling using slave processes suffers from the following limitations: i) blocking operations are forbidden within slave codes since master simulators need to wait for slave results, otherwise deadlocks will occur; and ii) multi-threading within slave process should be avoided to prevent non-deterministic results. These limitations significantly hindered the modeling of distributed software as multi-threading and blocking operations are very common.

Here, we discuss the work of Perkonigg *et al.* [14] in particular, which extended the JADE platform to support the simulation of unmodified agent codes. This is achieved by using

a wrapper agent class to override the standard JADE agent class. The wrapper agent class provides the same APIs as the standard agent class, and in addition, implements simulation control codes to i) redirect agent communications events to OPNET simulator; ii) track the computation delay and logical timestamp of the agent; and iii) control agent code executions to achieve synchronization. This design philosophy is similar to that of a direct-execution simulation, but the implementation is limited to JADE. In this paper, we propose a generalized design at the language level to support simulation of Java programs.

D. TIME SYNCHRONIZATION

Note that during a co-simulation session, simulators execute local events and advance local logical clocks in parallel. Synchronization mechanisms must be installed to ensure events and messages across all simulators are executed in correct timestamp order. The methods used by previous works can be classified into three types: master-slave, time-stepped, and event-driven.

1) In a master-slave synchronization scheme, one of the simulators is chosen as the master that triggers slave simulators to execute events and exchange information. The exact logical time for interaction is only known by the master. This synchronization method can often be found in co-simulations with a steady state power system simulator (slave), which calculates new system states when being triggered by the communication simulator (master). The master-slave approach should only be used if the slave simulators do not actively generate events that affect the master.

2) In a time-stepped synchronization scheme, all co-simulators execute independently until a specific logical time is reached. When all co-simulators have reached the time step, messages accumulated during this period can be exchanged between simulators. The simulation is then resumed until the next step. Since messages accumulated within a time-step are delayed until the next synchronization point, system error may occur and accumulate througout the simulation. This error can be reduced by decreasing step-size statically or allowing changes of step-size adaptively, but cannot be completely eliminated.

3) In an event-driven synchronization scheme, a simulator may send and receive messages at arbitrary time. During simulation, a time bound is assigned to each simulator. By limiting the execution to events within this bound, it can be guaranteed that local events and external messages are processed in a correct order. This eliminates system error that may occur in a time-stepped approach. Depending on the implementation, the synchronization protocol can be sequential or parallel.

Since our proposed co-simulation platform adheres to the HLA standard, its time synchronization mechanism is inherently parallel and event-driven which i) fully eliminates system error due to synchronization, and ii) allows the exploitation of parallelism in the host computer.

E. CO-SIMULATION INTERFACE

Since most proprietary simulation tools do not provide direct interfaces with standardized co-simulation frameworks, the implementation of ad-hoc interfaces to integrate communication and power system simulators is prominent in practice, especially in earlier works. Recent developments [12] [14] and this paper, however, devote efforts to the conformity of standardized IEEE 1516 High-Level Architecture (HLA) [25] interface to achieve better interoperability, re-usability and scalability in co-simulator design. The HLA standard specifies i) Object Model Templates to define co-simulation message structures understandable by all federates, and ii) the Runtime Infrastructure (RTI) to coordinate the message exchange, time synchronization, and management of a co-simulation session. Details on HLA-based co-simulation architecture is presented in Section IV.

III. DIRECT EXECUTION SIMULATION

The use of distributed software is pervasive for a wide range of smart grid applications. Their designs typically utilizes multi-threading, with dedicated threads for handling network events. In addition, synchronization operations are often performed by threads as a means to achieve inter-thread coordination. However, these kinds of design patterns are not supported by smart grid co-simulators reported in the literature because the unsupervised execution of multi-threading and blocking codes within a electrical/communication simulator may result in problems such as non-deterministic outcomes and deadlocks. In this section, we explain how direct-execution simulators are designed to eliminate this kind of problems. The following discussions are focused on the Java language, due to its popularity in distributed software development.

The major challenges in simulating a multi-thread Java execution is to ensure deterministic results. To achieve this, we first explore the reasons for non-deterministic behaviors, and then describe the corresponding countermeasures.

According to Java memory model (JMM) [30], a single-thread Java program performs a sequence of actions. Given a particular control flow path, the order for action execution is uniquely defined according to the thread-local semantics. Therefore, the execution of a single thread program is deterministic and repeatable as long as it does not contain unspecified actions such as reading object references and external inputs.

However, for a multi-threaded program, actions issued by different threads are not necessarily in order, hence their execution order may vary in different runs and produce different timing and functional behaviors. These differences accumulate as the simulation proceeds and lead to diverged results. The causes of non-deterministic/un-repeatable behaviors are summarized below.

1) Actions' execution order is not deterministically defined and enforced as explained above.

2) Thread scheduling is not defined nor controlled by the program, which contributes to the non-determinism of action's execution order.
3) The outcomes of Java synchronization operations (e.g., lock/unlock, wait/notify) may be non-repeatable even if their order of execution are exactly repeated. For example, when multiple threads contend for a lock, the order of lock acquisition is not defined in JMM even if the order of contention is the same.
4) Interactions between a program and external systems may not be exactly repeatable.

Corresponding countermeasures are employed by the DecompositionJ framework to eliminate these four sources of non-determinism.

1) In the simulation, actions are executed in an order determined by their timestamps. To enforce this timestamp ordering of action execution, a *logical time barrier* is inserted ahead of each thread action. This barrier postpones an action execution until all actions with lesser timestamps are completed.
2) Thread scheduling is modeled by introducing logical processors, logical threads, logical scheduler, and scheduling actions in the simulation. Logical threads participate in the scheduling by executing *processor-contend*, *acquire* and *release* actions. The outcomes of these actions are determined by the logical scheduler based on the first-come-first-served principle. A logical thread must acquire a logical processor to continue action execution. This is enforced by inserting a *logical processor barrier* after a *processor release*, which postpones action execution until the thread has re-acquired a logical processor.
3) Java synchronization operations in the program are replaced with their deterministic versions, which interacts with the logical scheduler.
4) External systems are replaced with their simulated counterparts, i.e., co-simulators. External interactions with other co-simulators are performed by exchanging timestamped messages. External messages and thread actions are processed according to their timestamp order. This is enforced by inserting *logical time barrier* before the processing of an external message.

A. OPERATION OF A DIRECT-EXECUTION SIMULATION

The design of a direct-execution simulator is in itself a very complicated matter that deserves dedicated paper [27] for discussion. Reference [27] provides detailed discussions on i) concurrent execution model; ii) definitions of actions, inter-action relationships, and constraints to ensure well-formedness of simulation; iii) exploitation of parallelism; and iv) performance and scalability evaluation. In the following subsections, we provide an overview on the runtime operation of direct-execution simulation by first describing the data structures used in the simulation, then followed by the discussion on the tracking of timestamps and the enforcing of timestamp order.

1) SIMULATION METADATA

The simulation *metadata* is a global data structure that tracks the states of a simulated execution. Metadata provides the necessary information for evaluating action timestamps and calculating the outcomes of scheduling and synchronization operations, and it consists of:

- Processor States: For each processor, metadata maintains its ID, logical clock, the latest dispatched thread, operating frequency, context switching delay, and an idle flag.
- Logical Threads: For each logical thread, metadata maintains its ID, the latest acquired processor and time-slice, the lock it contends, and the wait set it resides in.
- Logical Scheduler: This includes a set of active threads and a processor contention queue.
- External Event Queue: This includes messages received from external processes, which are sorted according to their timestamps.
- Locks: For each lock, metadata keeps track of its locked/unlocked state, latest acquiring thread, and a contention queue.
- Wait sets: For each wait set, metadata contains its wait queue and the associated lock.

Since the metadata is a shared data structure, a metadata lock is used to prevent concurrent access that leads to inconsistent states. A thread must hold the metadata lock when accessing metadata.

2) ENFORCING SIMULATED THREAD SCHEDULING

To simulate the effect of thread scheduling, actions can only be executed when the logical thread has acquired a logical processor. Therefore, processor barrier codes are inserted at the beginning of threads and at the end of every processor releasing operation. When a thread reaches a processor barrier, it will wait until a logical processor is assigned to it by the logical scheduler. Note that the metadata lock must be i) acquired before entering processor barrier, and ii) released while waiting inside or leaving the barrier.

3) TRACKING ACTION TIMESTAMPS

The timestamps of thread actions are tracked by using the processors' logical clock in the metadata. When a thread is being released from processor barriers, the clock of the newly acquired processor is updated. After leaving the processor barrier, the processor's clock will be incremented by the control codes each time a new action is reached. Essentially, processor clock tracks the timestamp of the next action to be performed by the running thread.

4) ENFORCING TIMESTAMP ORDER

The timestamp order of event execution is enforced by inserting logical time barrier ahead of synchronization actions or external messages. Note that in a co-simulation environment, it is also necessary to ensure that no external messages with lesser timestamps will arrive after leaving a logical time barrier. Therefore, the barrier contains a loop,

in which a thread invokes RTI's *NextMessageRequest* service to obtain the next external message with a timestamp less than that of the next action, and waits until a *TimeAdvanceGrant* is issued by the RTI. This request-grant cycle repeats until no further messages with lesser timestamp is received.

5) HANDLING EXTERNAL EVENTS

Messages received from co-simulators will be stored in the external event queue in the metadata. Events in the queue are sorted according to their timestamps and await to be processed by an external event thread (EET). The main body of EET is a loop. In each iteration, EET first waits at a logical time barrier until the next external message has the least timestamp. It then executes the handler associated with that message. If the event queue is empty, EET will wait at an *external event barrier* until new events arrive.

The operation principle of an external event thread is similar to that of a logical thread but with the following differences: (i) since external actions are not performed in the context of a logical thread, handlers must not contain blocking operations such that EET will not be blocked as a result of executing external operations, and (ii) external actions do not require logical processors for execution and therefore EET will not be blocked by processor barriers.

B. DECOMPOSITIONJ FRAMEWORK

The DecompositionJ framework is in essence a runtime library and a compiler-based code analyzer and transformer.

Data structure and various mechanisms used in a direct-execution simulation are implemented in a purposely designed runtime library, such as:

1) The simulation metadata.
2) Execution order enforcing mechanisms. These include logical scheduler, external event thread, barriers and time tracking mechanisms that are inserted into the target source code.
3) Deterministic versions of Java synchronization operations and time related operations. These include: *Object.wait*/*notify*/*notifyAll*, *Thread.start*/ *sleep*/ *yield*/ *join*/ *interrupt*/ *interrupted*/ *isInterrupted*/ *isAlive*/ *getState*, monitor *enter*/ *exit*, *System.currentTimeMillis*/ *nanoTime*.
4) Simulated versions of external interactions. For example, a simulated version of *java.net* package has been implemented to facilitate the simulation of TCP communications between Java programs. The simulation package translates various network events into co-simulation messages which are exchanged with the OPNET network simulator.

With the aid of the runtime library, the compiler-based analyzer and transformer can then convert a target Java program into a direct-execution simulator of itself. The analysis and conversion procedures are as follows:

1) The parser generates abstract syntax trees (AST) for all the compilation units, i.e., *.java* source files of the target program.
2) AST nodes representing Java synchronization operations are instrumented with barriers or replaced by their deterministic counterparts in the runtime library. Specifically, (i) for a volatile variable access, metadata lock and logical time barrier are inserted before the access, and metadata unlock is inserted after the access; (ii) for Synchronization-Blocks and Synchronization-Methods, they are replaced by a try-finally block that begins with a monitor enter and ends with a monitor exit operation; and (iii) for commonly used synchronization operations, they are replaced by their counterparts in the library.
3) AST nodes representing external interactions are replaced by their simulated versions, e.g., the *java.net* package.
4) Time tracking codes are then inserted between actions. Note that it is not necessary to inject a tracking code between every action since multiple intra-thread actions can be lumped into one if there is no branching in their control flow, i.e., they belong to the same basic block. Therefore, basic blocks are identified by using precise exceptional intra-procedural control flow analysis [33] and tracking codes are only inserted before each basic block.
5) The transformed ASTs are then rewritten into Java source which can then be compiled using JDKs to produce simulator executables.

The JastaddJ/ExtendJ compiler framework is used to automatically perform the above analysis and source-to-source transformation. It is important to note that manual modification of the original program source code is not required. The simulator produced by DecompositionJ is compatible with any JDK, JVM and their associated development tools such as debuggers, profilers, and IDEs. To begin the simulation, users are only required to write a startup code to specify parameters for the simulation environment, e.g., number of logical processors and their operating frequencies, and then redirect the control flow to the simulator produced by DecompositionJ. Therefore, the entire process is convenient and almost fully automatic.

IV. CO-SIMULATION PLATFORM FOR AGENT-BASED SMART GRID APPLICATIONS

As noted in recent literatures, the multi-agent software platform JADE [26] has become a popular choice for implementing and studying distributed smart grid applications. JADE features full compliance with the FIPA [28] (Foundation for Intelligent Physical Agents) specification designed by IEEE Computer Society for promoting agent interoperability. The platform provides Agent Management System, Directory Facilitator, Agent Interaction Protocols (AIPs) and Message Transport Services (MTS). The entire software is composed of more than 400K lines of codes and continue to evolve under active developments.

The multi-threading nature and complexity of JADE make it a good candidate for demonstrating the capability and

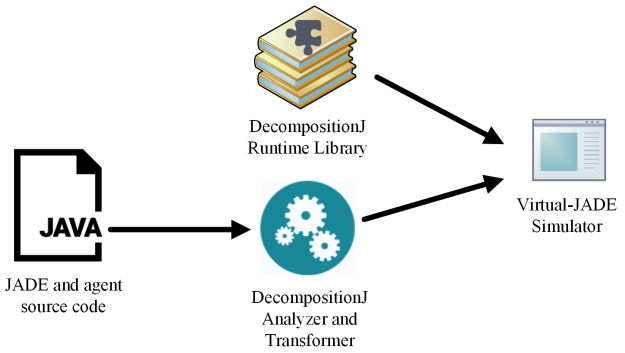

FIGURE 1. Converting JADE and agent code into direct-execution simulators using DecompositionJ.

usefulness of the proposed framework. As shown in Fig.1, the source code of JADE and agents are automatically analyzed and transformed using the DecompositionJ framework to produce Virtual JADE (V-JADE) simulators. V-JADE simulators then participate in a co-simulation with eletrical system simulator PSCAD and network simulator OPNET.

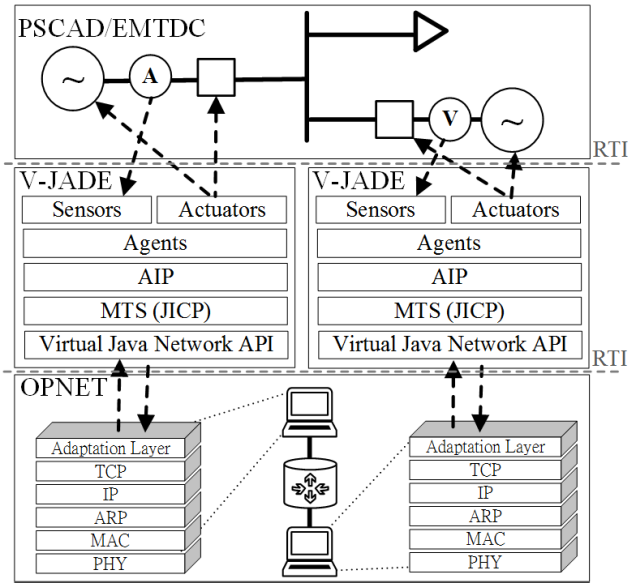

FIGURE 2. Co-simulation architecture.

As illustrated in Fig. 2, the co-simulation is achieved by interfacing each simulator with a common HLA-compliant Runtime Infrastructure. For V-JADE, the RTI interface has been developed as part of the external event handling mechanism inside the DecompositionJ runtime library. For PSCAD, the RTI interface has been developed as an external library linked to user-defined modules. For OPNET, the RTI interface is already built into the software. Note that in the JADE architecture, the multiagent platform is formed by multiple containers distributed over networked computers. Each container may encapsulate more than one agent. Therefore, multiple V-JADE simulators are used in the co-simulation.

A. CO-SIMULATION MESSAGE EXCHANGE

To develop an HLA co-simulation, it is crucial that the format of exchanged messages is made known to all simulators. Therefore, a Federate Object Model (FOM) must be designed to specify the structures and attributes of all objects and interactions being exchanged. The FOM specification file is then shared between simulators.

In our co-simulation design, V-JADEs and PSCAD exchange information via *sensor* and *actuator* objects. At the initialization phase of a co-simulation, *sensor* and *actuator* objects are created and registered on the RTI by PSCAD. During simulation, sensor readings from the electrical system are periodically published to the RTI by PSCAD. The RTI then delivers the messages to V-JADEs according to time-stamp order. For *actuator* objects, their values are published by V-JADE and subscribed by PSCAD.

The message exchange between V-JADEs and OPNET is considerably more complicated. Firstly, each V-JADE simulator is associated with a *workstation* object which is in turn associated with multiple *network interface* objects. Each *network interface* contains simulated network attributes exposed to V-JADE, which include *workstation ID* / *network interface ID* / *MAC address* / *hostname* / *IP version* / *IP address* / *subnet mask* and *MTU*. During the co-simulation initialization phase, OPNET creates, registers and publishes *workstation* and *network interface* objects to V-JADE simulators.

Secondly, when TCP socket operations are performed by V-JADE during simulation, RTI Interaction messages are sent to OPNET with reference to particular *workstation* and *interface* objects. Different types of interactions are used for different types of the operation, which include socket *create* / *open* / *listen* / *send* / *close* and *abort*. Conversely, when TCP socket events occur in OPNET, interactions will be sent to V-JADE with reference to particular *workstation* and *interface*. The interaction types include *open confirmation* / *close indication* / *close confirmation* / *data reception* / *error Indication* / *Abort indication* and *FIN reception*. The above types cover most functionalities of the *java.net* package, which is sufficient for simulating the communications between JADE containers.

B. CO-SIMULATION TIME SYNCHRONIZATION

Proper ordering of events across co-simulators is necessary to ensure causality and repeatability of the results. Time Management Service provided by HLA compliant RTIs handles this ordering using conservative time synchronization algorithms.

During the co-simulation, a simulator refrains from processing the next local event until it is guaranteed by the RTI that no messages with timestamps less than the next event will be received. This stop-and-wait procedure is achieved by first issuing a *NextMessageRequest* with t_{next_event} as parameter to the RTI, and then wait until a *TimeAdvanceGrant* is received from RTI. By issuing the *NextMessageRequest*,

the simulator also guarantees that it will not generate a message with timestamp less than $min(t_{next\_msg}, t_{next\_event}) + LA$, where t_{next_msg} refers to the timestamp of next message that will be delivered to the simulator, and LA refers to simulator's lookahead. The lookahead is a non-negative value that allows other simulators to execute beyond this simulator's local time. Using large lookahead values improve parallelism and hence enhance the performance of the co-simulation. Deriving a suitable lookahead value depends on the physical limitations of the simulated system, i.e., how quickly a simulated system can react to events received from another simulated system. For OPNET and Virtual JADE simulators, the overhead is assumed to be $1\mu s$ with the consideration of the processing delays between JVM and underlaying protocol stacks. For PSCAD, the lookahead is the same as the time-step size since PSCAD will not send out messages between simulation steps.

V. AGENT BASED FLISR CASE STUDY

Agent-based Fault Location, Isolation, and Service Restoration is a delay sensitive application that relies on fast decision response to reduce service interruption time. With this simulation case study, we demonstrate that the timeliness of service restoration can be influenced by the configurations of the JADE software and background traffics. In addition, we investigate a scenario in which both electrical and communication system failures occur simultaneously.

A. THE RESTORATION PROBLEM

When fault occurs in a radially configured power distribution network, the protection system isolates the faulted section and consequently blocks the power flow towards downstream sections. Switches are reconfigured to restore service for the affected sections with a network topology to minimize de-energized loads. Optimal network reconfiguration problem is traditionally solved off-line and the solutions are statically programmed to react upon events. This approach does not adapt well to the dynamic operation of smart distribution grids. Agent-based solutions were proposed in [19]–[24] to calculate optimal configuration based on dynamic information in a distributed manner.

In a reconfiguration problem, the power distribution network is considered as a graph $\mathbb{G} = \{V, E\}$, where each edge corresponds to a switch, and each node corresponds to a set of feeder buses and lines bounded by a common set of switches. For a node v, its power generation, generation capacity and load consumption are given by $G(v)$, $G^{max}(v)$ and $L(v)$ respectively; and for an edge $\{v_i, v_j\}$, its power flow (from node v_i to v_j) and line capacity are given by $P(v_i, v_j)$ and $P^{max}(v_i, v_j)$. Note that power flow is directional, i.e. $P(v_i, v_j) = -P(v_j, v_i)$. A modified IEEE 34-bus system is shown in Fig. 3a and its graph model is shown in Fig. 3b which illustrates the graph representation of distribution network. This system is also used in this FLISR case study.

When a fault occurs, the nodes are classified into three disjoint subsets: supplier nodes V^s (nodes with positive net power output), consumer nodes V^c, and faulted nodes V^f. Consider a graph $\mathbb{G}$ with N supplier nodes, the reconfiguration algorithm aims to find a set of N trees $H' = \{T_1, T_2, , T_N\}$ for restoration, such that the total restored load is maximized:

$$H' = \underset{H=\{T_1,,T_N\}}{\text{argmax}} \sum_{T\in H} \sum_{v\in T} L(v). \tag{1}$$

The optimization subjects to the following constraints:

1) Each tree must contain a supplier node as the root and exclude all the faulted nodes.

$$\forall T_n = \{V_n, E_n\} \in H. \quad |V_n \cap V^s| = 1 \tag{2}$$

$$\forall T_n = \{V_n, E_n\} \in H. \quad V_n \cap V^f = \emptyset \tag{3}$$

2) The trees are disjoint.

$$\forall T_n, T_m \in H. \quad T_n \neq T_m \implies T_n \cap T_m = \emptyset \tag{4}$$

3) Power balance must be satisfied, i.e., the input power of a node is equal to the total output power for its successor nodes plus local load. For consumer node, the net input power equals load consumption.

$$\forall v \in V^c \cap H. \quad \sum_{\{v',v\}\in E} P(v', v) = L(v) \tag{5}$$

For supplier node, the sum of generation and net input power equals consumption.

$$\forall v \in V^s. \quad G(v) + \sum_{\{v',v\}\in E} P(v', v) = L(v) \tag{6}$$

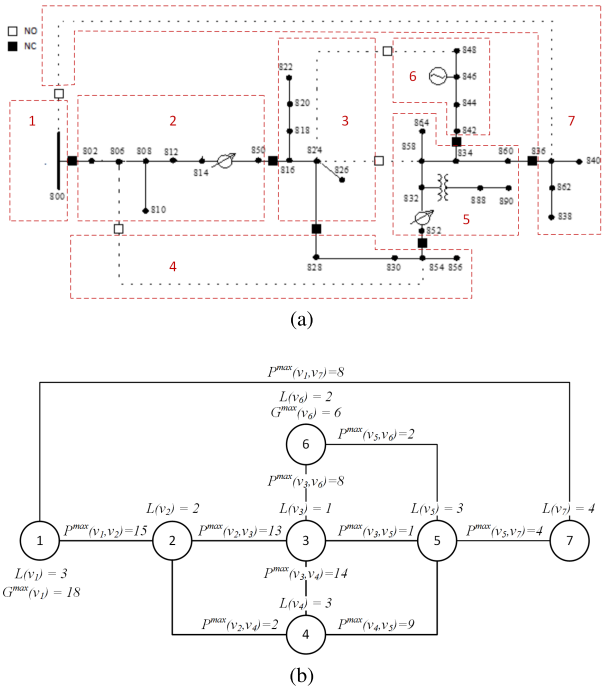

FIGURE 3. (a) Shows a modified IEEE 34 bus distribution network, and feeder buses bounded by a common set of switches are grouped in dashed boxes. (b) Shows the graph model for the distribution network.

4) Power generation of each DER is bounded by its maximum capacity.

$$\forall v \in V^s. \quad 0 \leq G(v) \leq G^{max}(v) \tag{7}$$

5) Power flow of each line is bounded by maximum line capacity.

$$\forall \{v_i, v_j\} \in H. \quad |P(v_i, v_j)| \leq P^{max}(v_i, v_j) \tag{8}$$

B. RECONFIGURATION ALGORITHM

Due to the distributed nature of multi-agent systems and the dynamic nature of smart grid, each node on the network only possesses local information. Therefore, after the fault, the agents at each node explore the graph and obtain information from agents in other nodes in order to determine the restoration trees. Since each restoration tree must contain a supplier node, the exploration and tree growth start from the supplier node. The procedures of the reconfiguration algorithm are as follows:

1) Initially, a restoration tree T_n is created for each supplier node $v_n^s \in V^s$, which contains v_n^s as its root.

2) For each restoration tree, frontier edges F_n is defined as the edges between nodes inside T_n and nodes outside T_n. F_n will be updated when a new node is recruited to the tree.

3) For each tree node v other than the root supplier node, $in(v)$ represents the parent tree node that supplies power to v, and $out(v)$ represents the set of child tree nodes which draws power from v.

4) For each tree node v, $R(v)$ represents the surplus power that can be drawn from the node if new nodes are added to the tree through v. By constraining $R(v) \geq 0$ for all nodes, (5)-(8) can be enforced.

$$\forall v \in V^c. \quad R(v) = min\big(R(pr(v)), P^{max}(pr(v), v)\big) - L(v) - \sum_{v' \in out(v)} P(v, v') \tag{9}$$

$$\forall v \in V^s. \quad R(v) = G^{max}(v) - L(v) - \sum_{v' \in out(v)} P(v, v') \tag{10}$$

5) A weight w, representing the load of the neighboring node reachable through the edge, is assigned to each frontier edge.

$$\forall \{v_i, v_j\} \in F_n. \quad w(v_i, v_j) = w(v_j, v_i) = L(v_j) \tag{11}$$

6) For as long as F_n is not empty, the algorithm greedily explores the frontier edge with the largest weight and attempts to recruit the neighboring node through that edge.

7) To ensure a neighboring node is eligible for recruitment, the supplier node's agent enquires its state through communications. The neighboring node will be recruited unless it i) is faulted, or ii) is already included in another restoration tree, or iii) has a load exceeding the R value of its parent node.

8) New frontier edges F_{new} are added to F_n when a new node is recruited. Supplier node communicates with the new node in order to acquire F_{new} and their associated weights. Additionally, R value of all tree nodes will be updated to account for the new load.

9) The algorithm ends when all the frontier edges are explored. The distribution network will be restored according to the topology of the restoration trees.

C. RESTORATION AGENTS

The FLISR mechanisms including the reconfiguration algorithm are implemented using a two-tier multiagent system (MAS) with four agent types. As shown in Fig. 4, the upper tier consists of Node Agents (NA) and Switch Agents (SA), while the lower tier consists of Load Agents (LA) and DER Agents (DA). Lower tier agents can only communicate with their supervising NAs, while upper tier agents can communicate with one another. The operation of each type of agent will be presented in the following.

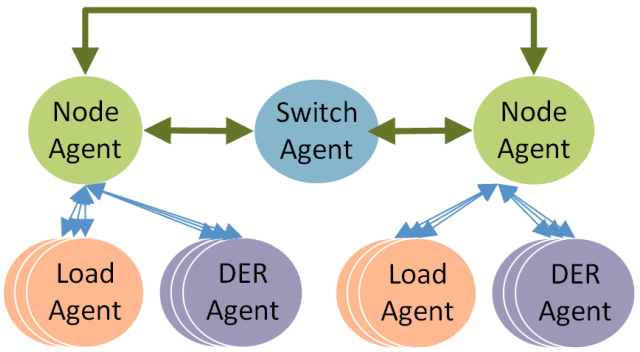

FIGURE 4. Two-tiered MAS hierarchy.

SAs, LAs and DAs are connected to their corresponding physical switches, loads and DERs through sensors and actuators. These agents control and monitor their corresponding physical devices and respond to their supervising NAs' requests, such as adjusting operating points and reporting device status. In addition, SAs are also responsible for over current protection and fault detection.

NAs have no direct interaction with the electrical system, but they are responsible for coordinating the FLISR mechanism. Fig. 5 depicts its operating states. Initially, NA stays in idle state until an anomaly is reported by a neighboring SA. Upon which the NA will enquire current and voltage readings from all neighboring SAs. If a fault is detected within its segment, NA will isolate the fault by requesting all neighboring SA to open their switches and it will then remain isolated and not be restored. On the contrary, NA will proceed to the inquiry phase to identify its local generations

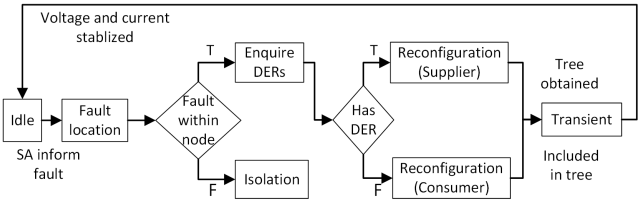

FIGURE 5. Operation flow of a Node Agent.

and consumptions through subordinate DAs and LAs. After that, NAs with local generations (i.e. supplier node) will initiate the reconfiguration algorithm, while consumer NAs will wait and respond to inquiries. Supplier NAs will enter transient state and request both DERs and switches to adjust to new network configuration upon the completion of the reconfiguration algorithm. Finally, NAs return to idle state when stabilized bus voltages is observed.

D. COMMUNICATION NETWORK CONFIGURATIONS

For communication infrastructure, we consider gateway routers being installed at all buses, which are connected by optical carriers running alongside with distribution lines. The distances between routers are stipulated in the IEEE 34 Bus system dataset. Each agent runs in a separate JADE container which is mapped to a workstation that connects to a router. Further details are listed in Table 2.

TABLE 2. Characteristics of simulated network.

Characteristics	Configuration
Transport layer	Background traffic: UDP. JADE traffic: TCP
TCP Configurations	TCP Reno, Karn's Algo., initial retransmission timeout: 0.5s
Network layer	IPv4
Routing	EIGRP
Link layer	Ethernet. MTU:1500 Bytes. Speed grade: OC-1 (51.8Mbps)
Router Processing	Switching rate: 200,000 pkt/s. Forwarding: 20,000 pkt/s
Background traffic	From all buses to bus 800 (substation)

VI. SIMULATION RESULTS

The agent based FLISR case has been simulated using the proposed co-simulation platform. In this section, we first explain the MAS operation by presenting a trace of significant simulation events in the FILSR process. Then the performance of MAS is evaluated with different network and software configurations: i) background traffics, ii) communication link failure, iii) communication link failure time, and iv) location of JADE main container responsible for maintaining all agents' network address.

A. AGENT ACTIONS AND EVENTS

The event trace in Table 3 is produced without simulating background traffic or link failure. This table only shows relevant information from supplier Node agents, i.e., NA1 and NA6.

- Phase 1 - Fault Detection

At t=0, a three-phase ground fault occurs between bus 806 and 808 (inside v_2), which causes fault current to flow through Bus 800 towards 808 and voltage sag in the distribution network. The fault current is first detected by SA12 and reported to NA1 at t=0.00931 (#1), using the *Switch Anomaly Inform: Inform* message. This triggers NA1 to change its state from Idle to Fault Location (#2). Similarly, NA6 enters Fault Location state after receiving the message from SA56 (#15, 16). Note that only the first

TABLE 3. Simulation event trace for NA1 and NA6.

#	Time(s)	Agt	Type	Event	Remarks
1	.00931	NA1	Recv	Switch Anomaly: Inform	From SA12
2	.00931	NA1	State	Idle → Fault Location	
3	.00935	NA1	Send	Switch Status Query: Query-ref	To SA12, SA17
4	.01303	NA1	Recv	Switch Status Query: Inform	SA12 (over cur.) , SA17 (under vlt.)
5	.01303	NA1	Algo	Team Status Confirmed	Not Faulted
6	.01304	NA1	State	Fault Location → Enquire	
7	.01307	NA1	Send	DA/LA Status Query: Query-ref	To DA1, LA1
8	.01571	NA1	Recv	DA/LA Status Query: Inform	From DA1, LA1
9	.01572	NA1	State	Enquire → Reconfig. Supplier	
10	.01573	NA1	Algo	Initialize frontier edge set	$\{v_1, v_2\}\{v_1, v_7\}$
11	.01573	NA1	Algo	Initialize $R(v_1)$	N/A → 15.0
12	.01573	NA1	Algo	Attempt to restore frontier edge	$\{v_1, v_7\}$
13	.01576	NA1	Send	Restoration Query: Query-If	To NA7
14	.02143	NA1	Recv	Restoration Query: Agree	From NA7
15	.02872	NA6	Recv	Switch Anomaly: Inform	From SA56
16	.02874	NA6	State	Idle → Fault Location	
17	.02878	NA6	Send	Switch Status Query: Query-ref	To SA56, SA36
18	.03239	NA1	Recv	Restoration Query : Inform	From NA7
19	.03240	NA1	Algo	New frontier edge discovered	$\{v_5, v_7\}$
20	.03240	NA1	Algo	Update $R(v_7)$	N/A → 4.0
21	.03240	NA1	Algo	Update $R(v_1)$	15.0 → 11.0
22	.03241	NA1	Algo	Attempt to restore frontier edge	$\{v_7, v_5\}$
23	.03243	NA1	Send	Restoration Query: Query-If	To NA5
24	.04143	NA6	Recv	Switch Status Query: Inform	SA56 (under vlt.) , SA36 (under vlt.)
25	.04143	NA6	Algo	Team Status Confirmed	Not Faulted
26	.04144	NA6	State	Fault Location → Enquire	
27	.04147	NA6	Send	DA/LA Status Query: Query-ref	To DA6, LA6
28	.04599	NA1	Recv	Restoration Query: Agree	From NA5
29	.04916	NA6	Recv	DA/LA Status Query: Inform	From DA6, LA6
30	.04917	NA6	State	Enquire → Reconfig. Supplier	
31	.04918	NA6	Algo	Initialize frontier edge set	$\{v_6, v_3\}\{v_6, v_5\}$
32	.04918	NA6	Algo	Initialize $R(v_6)$	N/A → 4.0
33	.04918	NA6	Algo	Attempt to restore frontier edge	$\{v_6, v_5\}$
34	.04918	NA6	Algo	Violate constraint	$w(v_6, v_5) \leq P^{max}_{(v_6,v_5)}$
35	.04918	NA6	Algo	Attempt to restore frontier edge	$\{v_6, v_3\}$
36	.04921	NA6	Send	Restoration Query: Query-If	To NA3
37	.05302	NA1	Recv	Restoration Query : Inform	From NA5
38	.05303	NA1	Algo	New frontier edge discovered	$\{v_5, v_3\}\{v_5, v_4\}$ $\{v_5, v_6\}$
39	.05303	NA1	Algo	Update $R(v_5)$	N/A → 1.0
40	.05303	NA1	Algo	Update $R(v_7)$	4.0 → 1.0
41	.05303	NA1	Algo	Update $R(v_1)$	11.0 → 8.0
42	.05304	NA1	Algo	Attempt to restore frontier edge	$\{v_5, v_4\}$
43	.05305	NA1	Algo	Violate constraint	$w(v_5, v_4) \leq R(v_5)$
44	.05305	NA1	Algo	Attempt to restore frontier edge	$\{v_5, v_6\}$
45	.05305	NA1	Algo	Violate constraint	$w(v_5, v_6) \leq R(v_5)$
46	.05306	NA1	Algo	Attempt to restore frontier edge	$\{v_1, v_2\}$
47	.05307	NA1	Send	Restoration Query: Query-If	To NA2
48	.05679	NA6	Recv	Restoration Query: Agree	From NA3
49	.05689	NA1	Recv	Restoration Query: Refuse	From NA2, faulted
50	.05695	NA1	Algo	Attempt to restore frontier edge	$\{v_5, v_3\}$
51	.05697	NA1	Send	Restoration Query: Query-If	To NA3
52	.06244	NA6	Recv	Restoration Query : Inform	From NA3
53	.06245	NA6	Algo	New frontier edge discovered	$\{v_3, v_2\}\{v_3, v_4\}$ $\{v_3, v_5\}$
54	.06245	NA6	Algo	Update $R(v_6)$	4.0 → 3.0
55	.06246	NA6	Algo	Attempt to restore frontier edge	$\{v_3, v_5\}$
56	.06246	NA6	Algo	Violate constraint	$w(v_3, v_5) \leq P^{max}_{(v_3,v_5)}$
57	.06246	NA6	Algo	Attempt to restore frontier edge	$\{v_3, v_4\}$
58	.06248	NA6	Send	Restoration Query: Query-If	To NA4
59	.06557	NA1	Recv	Restoration Query: Refuse	From NA3, restored
60	.06733	NA1	Algo	No restorable frontier edge left	
61	.06733	NA1	State	Reconfig. Supplier → Transient	
62	.06735	NA1	Send	DA Operation Request: Request	To DA1 (P=10)
63	.06788	NA1	Recv	DA Operation Request: Agree	From DA1
64	.07050	NA6	Recv	Restoration Query: Agree	From NA4
65	.07650	NA6	Recv	Restoration Query : Inform	From NA4
66	.07651	NA6	Algo	New frontier edge discovered	$\{v_4, v_5\}$,$\{v_4, v_2\}$
67	.07651	NA6	Algo	Update $R(v_6)$	3.0 → 0.0
68	.07651	NA6	Algo	Update $R(v_3)$	3.0 → 0.0
69	.07652	NA6	Algo	No restorable frontier edge left	$R(v_6) = 0$
70	.07652	NA6	State	Reconfig. Supplier → Transient	
71	.07654	NA6	Send	DA Operation Request: Request	To DA6 (P=6)
72	.07708	NA6	Recv	DA Operation Request: Agree	From DA6

Switch Anomaly Inform messages trigger the state change, later messages are neglected and not shown in the trace.

- Phase 2 - Fault Location

During Fault Location phase, NA1 sends *Switch Status Query: Query-ref* messages to all its neighboring SAs (#3),

and awaits their responses. Responses (*Switch Status Query: Inform*) containing recent current readings are received at t = 0.01303 (#4). The readings indicate that the fault is outside v_1 (#5). Consequently, NA1 proceeds to Enquire DERs and Loads state (#6). Similarly operations are also performed by NA6 (#17, 24-26).

- Phase 3 - Enquire DERs

To identify the total power generation capacity and load consumption of the node. NA1 sends a query message (*DER/Load Status Query: Query-ref*) to DA1 and LA1 (#7). The corresponding response messages are received at #8, indicating that v_1 has a generation capacity of 18.0 and load of 3.0, and hence 15.0 units are available for restoring other nodes. NA1 then enters Restoration Supplier state (#9). Similarly, NA6 follows the same routine (#27, 29-30) and has a surplus power of 4 units. For nodes without DER, their NAs will enter Restoration Consumer state after receiving the response messages from their subordinate LAs.

- Phase 4 - Reconfiguration

After entering the reconfiguration phase, each supplier node becomes the root of a restoration tree and its node agent executes reconfiguration algorithm to expand the tree. Consumer nodes also participate in the process by reacting to the messages from supplier nodes. Initially, T_1 only consists of the root node v_1 with two frontier edges $\{v_1, v_2\}$ and $\{v_1, v_7\}$ (#10). The edge with the largest weight (i.e., $\{v_1, v_7\}$) is selected for restoration. NA1 sends a *Restoration Query: Query-If* message to NA7 (#13) to verify that the constraints mentioned in step 7) of the reconfiguration algorithm are satisfied. Note that v_7 is restartable because it is not faulted, nor has it been included in any restoration tree. Therefore the recruitment of v_7 is confirmed with a *Restoration Query: Agree* message replied to NA1 (#14). Upon recruitment, NA7 issues *Switch Operation Request: Request* to SA17 to establish connection to T_1 by closing the switch. Following that, NA7 proceeds to enquire its neighboring SAs and NAs for a list of new frontier edges. These edges are then identified and reported back to NA1 through an *Restoration Query: Inform* message (#18,19). Finally, NA1 updates the surplus power of v_1 and v_7 (#20,21). The above process represents one iteration of the reconfiguration algorithm. Each restoration supplier will repeat this process to explore its neighborhood and attempt to expand the tree until all eligible frontier edges are exhausted. For NA1 and NA6, their reconfiguration phases end at t=0.06733(#61) and t = 0.07652 (#70), respectively. Fig.6 shows the final network configuration.

- Phase 5 - Transient

Once the algorithm ends, supplier node enters transient state and issues an *Operation Request: Request* to its DA (#62 for NA1, #71 for NA6). The DA then replies with an *Operation Request: Agree* message and starts power generation. The communication and computational delays of MAS operations are observable from the electrical system, which is shown in Fig. 7. The fault current is detected at #1 by SA12 which controls the switch to open at the next zero crossing point. Voltages of bus 836 and bus 824 are restored shortly after the DERs agree to power generation (#63, 72).

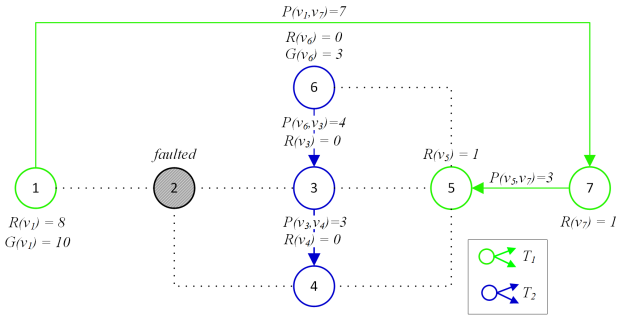

FIGURE 6. New network configuration, with two restoration trees rooted at node-1 and node-6.

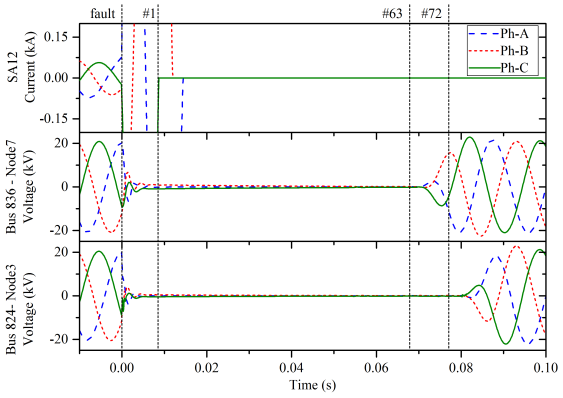

FIGURE 7. Simulation of electrical transients during fault and restoration process.

B. EFFECTS OF BACKGROUND TRAFFICS AND LINK FAILURE

Background traffic is sent from all feeder buses to a data aggregator at bus 800 using IP-based protocols. Note that two OC-1 links are connected to bus 800, i.e., from bus 836 to 802, allowing a maximum throughput of 100.224Mbps (payload) which is equivalent to 3.037Mbps per bus. Fig. 8 shows the solution time, i.e., time for supplier NA to enter transient state, of NA1 and NA6 with different levels of traffics. As the configuration of background traffic rate is increased towards the maximum throughput, the average packet queuing delay approaches infinity, which explains the exponentially increasing solution time.

Since communication links run along with power distribution lines, it is likely for an electrical fault to correlate with communication link failures. To study the impact, a link failure is scheduled to occur simultaneously with the electrical fault between buses 806 and 808. As shown in Fig. 8, the reconfiguration time increases due to more background traffics being distributed over the remaining links. Table 4 shows the percentage increment of reconfiguration time caused by simultaneous link failure and the results indicate a more significant impact under higher traffics.

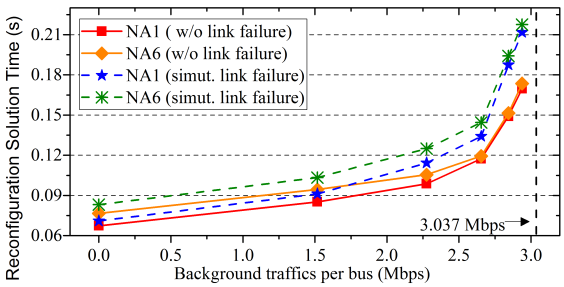

FIGURE 8. Reconfiguration solution time under different levels of background traffic.

TABLE 4. Percentage change of reconfiguration time.

Data rate per bus / 3.037Mbps	0%	50%	75%	87.5%	93.75%
NA1 solution time % change	+5.33%	6.98%	15.78%	14.21%	25.70%
NA6 solution time % change	+8.61%	9.43%	18.51%	21.00%	28.13%

Percentage change of reconfiguration time caused by simultaneous link failure under different amount of background traffics

C. EFFECTS OF LINK FAILURE TIME

Since JADE message transport service benefits from the retransmission mechanisms provided by the underlaying TCP protocol, application programmers may often assume reliable and timely agent communications without concerning packet losses in the network so long as the reachability between JADE containers is maintained. However, the co-simulation reveals that when both electrical fault and communication link failures occur, their temporal proximity greatly affects the reconfiguration solution time.

In Fig. 9, an electrical fault occurs at $t = 0$. The X-axis represents the link failure time and Y-axis represents the reconfiguration solution time of node agents.

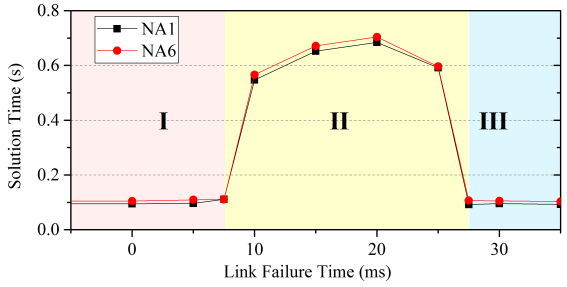

FIGURE 9. If comm. link failure occurs shortly after electrical fault, agent communications may experience large delay due to TCP initial retransmission delay. Reconfiguration time varies significantly depending on whether router's forwarding tables are updated before agent communications.

If link failure occurs in zone I, i.e., it occurs less than 7.5ms after the electrical fault, the effect of link failure on solution time is insignificant. This is because agent communications only begin at 9.31ms (#1 in Table 3) after the fault, hence routers adjacent to the failed link have sufficient time to detect it and update their forwarding tables before the agent communications. As a result, no MAS packet is lost.

However, a dramatic increment of solution time is observed if link failure occurs in zone II, i.e., between 7.5ms and 27.5ms after the fault. This is because agents have begun establishing TCP connections with each other via three-way handshakes before alternative routes are updated in router's forwarding tables. The loss of handshake packets is not timely retransmitted by TCP because the session's round-trip time (RTT) has yet to be measured and a rather long initial retransmission timeout of 0.5s is used, leading to a significant impact on the solution time.

If link failure occurs in zone III, i.e., 27.5ms after the fault, it will not have a significant impact on the solution time. Because TCP handshake between agents has been completed before the link failure and RTT measurements can be used to perform timely retransmission with a negligible timeout delay.

A possible solution to achieve short reconfiguration time is to modify JADE or agent codes which include i) TCP connections are pre-established between every pair of agents, and ii) heartbeat signals are sent between agents to periodically update the RTT. However this is outside the scope of this paper.

The above analysis of simulation results demonstrates the capability of the proposed framework to capture the long solution time when communication failure and electrical fault occurred in quick succession.

D. EFFECTS OF MAIN-CONTAINER LOCATION CONFIGURATION

Under the JADE architecture, a *Global Agent Descriptor Table* storing the network addresses of all agents is maintained by the main container (MC). The network location of MC has an impact on the solution time since agents must first enquire MC to resolve each other's network address before they attempt to communicate for the first time.

As shown in Fig.10, the solution time for different node agents varies depending on the location of the MC. When NA1's container is selected as MC, its solution time is benefited because all agent interactions originated from NA1 can be resolved locally. Similar benefit for NA6 can be obtained if its container is assigned as MC. The timing information is useful for finding the optimial MC location.

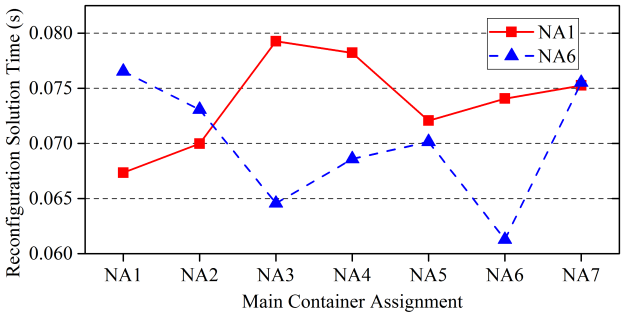

FIGURE 10. Reconfiguration time affected by the network location of Main Container in the JADE platform.

E. SUMMARY ON SIMULATION RESULTS

Through this agent-based FLISR case study, the benefits for incorporating direct-execution simulators in a smart grid co-simulation are demonstrated:

- The detailed and authentic behavior of smart grid software systems can be generated to facilitate the understanding of system operations, as shown by the agent event trace.
- Inter-dependencies between communication systems and software systems can be simulated, as shown by the effects of background traffics on reconfiguration solution time.
- Inherent design concerns in software and communication systems can be revealed by co-simulation, as demonstrated in the case where electrical fault and link failure occurred in quick succession.
- Software designers may fine-tune the JADE main container location to improve solution time.

VII. CONCLUSION

This paper presents a novel integration of direct-execution simulators to a Smart Grid co-simulation platform that adheres to the HLA standard. The DecompositionJ framework automatically performs static analysis and source-to-source transformation to convert a target program into its own simulator. This removes the burden of manual modeling and code development, and simplifies version control and maintenance. The transformed simulator is compatible with existing Java environments and tools, which facilitates researchers to debug and study a target program during simulation. A complex multiagent platform JADE is simulated and used in an FLISR case study. Results illustrate that low-level details in software and network configuration can affect the time required for restoration, which would otherwise be overlooked. This convenient and effective direct-execution simulation framework with high-fidelity modeling capability is believed to be important for enabling simulation studies on a wide range of smart grid applications.

REFERENCES

[1] K. Mets, J. A. Ojea, and C. Develder, "Combining power and communication network simulation for cost-effective smart grid analysis," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1771–1796, 3rd Quarter., 2014.

[2] W. Li, M. Ferdowsi, M. Stevic, A. Monti, and F. Ponci, "Cosimulation for smart grid communications," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2374–2384, Apr. 2014.

[3] K. Hopkinson, X. Wang, R. Giovanini, J. Thorp, K. Birman, and D. Coury, "EPOCHS: A platform for agent-based electric power and communication simulation built from commercial off-the-shelf components," *IEEE Trans. Power Syst.*, vol. 21, no. 2, pp. 548–558, May 2006.

[4] J. Nutaro, P. T. Kuruganti, L. Miller, S. Mullen, and M. Shankar, "Integrated hybrid-simulation of electric power and communications systems," in *Proc. IEEE Power Eng. Soc. General Meeting*, Tampa, FL, USA, Jun. 2007, pp. 1–8.

[5] J. Nutaro, "Designing power system simulators for the smart grid: Combining controls, communications, and electro-mechanical dynamics," in *Proc. IEEE Power Eng. Soc. General Meeting*, San Diego, CA, USA, Jul. 2011, pp. 1–5.

[6] V. Liberatore and A. Al-Hammouri, "Smart grid communication and co-simulation," in *Proc. IEEE Energytech*, Cleveland, OH, USA, May 2011, pp. 1–5.

[7] K. Mets, T. Verschueren, C. Develder, T. L. Vandoorn, and L. Vandevelde, "Integrated simulation of power and communication networks for smart grid applications," in *Proc. IEEE 16th Int. Workshop Comput. Aided Modeling Design Commun. Links Netw. (CAMAD)*, Cleveland, Kyoto, Jun. 2011, pp. 61–65.

[8] W. Li, H. Li, and A. Monti, "Using co-simulation method to analyze the communication delay impact in agent-based wide area power system stabilizing control," in *Proc. Grand Challenges Modeling Simulation Conf.*, Vista, CA, USA, Jun. 2011, pp. 356–361.

[9] W. Li, A. Monti, M. Luo, and R. A. Dougal, "VPNET: A co-simulation framework for analyzing communication channel effects on power systems," in *Proc. IEEE Electr. Ship Technol. Symp.*, Alexandria, VA, USA, Apr. 2011, pp. 143–149.

[10] H. Lin, S. S. Veda, S. S. Shukla, L. Mili, and J. Thorp, "GECO: Global event-driven co-simulation framework for interconnected power system and communication network," *IEEE Trans. Smart Grid*, vol. 3, no. 3, pp. 1444–1456, Sep. 2012.

[11] M. Lévesque, D. Q. Xu, Gé. Joós, and M. Maier, "Communications and Power Distribution Network Co-simulation for Multidisciplinary Smart Grid Experimentations," in *Proc. 45th Annu. Simulation Symp.*, San Diego, CA, USA, Mar. 2012, pp. 2:1–2:7.

[12] H. Georg, S. C. Müller, C. Rehtanz, and C. Wietfeld, "Analyzing cyber-physical energy systems:The INSPIRE cosimulation of power and ICT systems using HLA," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2364–2373, Nov. 2014.

[13] G. Celli, P. A. Pegoraro, F. Pilo, G. Pisano, and S. Sulis, "DMS cyber-physical simulation for assessing the impact of state estimation and communication media in smart grid operation," *IEEE Trans. Power Syst.*, vol. 29, no. 5, pp. 2436–2446, Sep. 2014.

[14] F. Perkonigg, D. Brujic, and M. Ristic, "Platform for development and validation agent-based smart grid applications incorporating accurate communications modelling," *IEEE Trans. Ind. Informat.*, vol. 11, no. 3, pp. 728–736, Apr. 2015.

[15] X. Li, Q. Huang, and D. Wu, "Distributed large-scale co-simulation for IoT-aided smart grid control," *IEEE Access*, vol. 5, pp. 19951–19960, 2017.

[16] M. Garau, G. Celli, E. Ghiani, F. Pilo, and S. Corti, "Evaluation of smart grid communication technologies with a co-simulation platform," *IEEE Wireless Commun.*, vol. 24, no. 2, pp. 42–49, Apr. 2017.

[17] V. Salehi, A. Mohamed, A. Mazloomzadeh, and O. A. Mohammed, "Laboratory-based smart power system, part I: Design and system development," *IEEE Trans. Smart Grid*, vol. 3, no. 3, pp. 1394–1404, Sep. 2012.

[18] X. Wang *et al.*, "Interfacing issues in multiagent simulation for smart grid applications," *IEEE Trans. Power Del.*, vol. 28, no. 3, pp. 1918–1927, Jul. 2013.

[19] C. P. Nguyen and A. J. Flueck, "Agent based restoration with distributed energy storage support in smart grids," *IEEE Trans. Smart Grid*, vol. 3, no. 2, pp. 1029–1038, Jun. 2012.

[20] M. Eriksson, M. Armendariz, O. O. Vasilenko, A. Saleem, and L. Nordström, "Multiagent-based distribution automation solution for self-healing grids," *IEEE Trans. Ind. Electron.*, vol. 62, no. 4, pp. 2620–2628, Apr. 2015.

[21] J. M. Solanki, S. Khushalani, and N. N. Schulz, "A multi-agent solution to distribution systems restoration," *IEEE Trans. Power Syst.*, vol. 22, no. 3, pp. 1026–1034, Aug. 2007.

[22] F. Ren, M. Zhang, D. Soetanto, and X. Su, "Conceptual design of a multi-agent system for interconnected power systems restoration," *IEEE Trans. Power Syst.*, vol. 27, no. 2, pp. 732–740, May 2012.

[23] D. Ye, M. Zhang, and D. Sutanto, "A hybrid multiagent framework with Q-learning for power grid systems restoration," *IEEE Trans. Power Syst.*, vol. 26, no. 4, pp. 2434–2441, Nov. 2011.

[24] A. Sharma, D. Srinivasan, and A. Trivedi, "A decentralized multiagent system approach for service restoration using DG islanding," *IEEE Trans. Smart Grid*, vol. 6, no. 6, pp. 2784–2793, Nov. 2015.

[25] *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), Mar. 2013, pp. 1–38.

[26] F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi, "JADE—A Java agent development framework," in *Multi-Agent Programming*. NewYork, NY, USA: Springer, 2005, ch. 5, pp. 125–147.

[27] C. Shum *et al.*, "DecompositionJ: Parallel and deterministic simulation of concurrent Java executions in cyber-physical systems," *IEEE Access*, to be published, doi: doic 10.1109/ACCESS.2018.2825254.

[28] FIPA. (2014). *Foundation for Intelligent Physical Agents. A Standards Organization of the IEEE Computer Society*. [Online]. Available: http://www.fipa.org/
[29] H. W. Dommel, *EMTP Theory Book*. Portland, OR, USA: Bonneville Power Administration, 1984.
[30] J. Manson, W. Pugh, and S. V. Adve, "The Java memory model," in *Proc. 32nd ACM SIGPLAN-SIGACT Symp. Principles Programming Lang.*, Long Beach, CA, USA, 2005, pp. 378–391.
[31] T. Ekman and G. Hedin, "The jastadd extensible Java compiler," *ACM SIGPLAN Notices*, vol. 42, no. 10, pp. 1–18, Oct. 2007.
[32] J. Öqvist and G. Hedin, "Extending the JastAdd extensible Java compiler to Java 7," in *Proc. Int. Conf. Principles Practices Programming Java Platform, Virtual Mach., Lang., Tools*, Stuttgart, Germany, 2013, pp. 147–152.
[33] E. Söderberg, T. Ekman, G. Hedin, and E. Magnusson, "Extensible intraprocedual flow analysis at the abstract syntax tree level," *Sci. Comput. Programming*, vol. 78, no. 10, pp. 1809–1827, 2013.

CHONG SHUM (S'14) received the B. Eng. degree in computer engineering from the City University of Hong Kong in 2012, where he is currently pursuing the Ph.D. degree in electronic engineering. His research interests include the modeling and simulation of cyber-physical systems and smart grids.

WING-HONG LAU (M'88–SM'06) received the B.Sc. and Ph.D. degrees in electrical and electronic engineering from the University of Portsmouth, Portsmouth, U.K., in 1985 and 1989, respectively. He joined the Department of Electronic Engineering, City University of Hong Kong, as an Associate Professor in 1990.

His current research interests include digital signal processing, digital audio engineering, pulse width modulation spectrum analysis, embedded system design, and smart-grid development.

Dr. Lau received the IEEE Third Millennium Medal. He was the Chairman of the IEEE Hong Kong Section in 2005.

TIAN MAO (S'14) received the B.S. degree in electrical engineering and its automation and the M.S. degree in electrical engineering from Hunan University in 2010 and 2013, respectively, and the Ph.D. degree from the City University of Hong Kong in 2017. He is currently with the Electric Power Research Institute, China Southern Power Grid. His research interests include power system operation, electric vehicle charging scheduling, and smart-grid energy management and optimization.

HENRY SHU-HUNG CHUNG (M'95–SM'03–F'16) received the B.Eng. and Ph.D. degrees in electrical engineering from The Hong Kong Polytechnic University in 1991 and 1994, respectively.

Since 1995, he has been with the City University of Hong Kong. He is currently a Professor with the Department of Electronic Engineering and the Director of the Centre for Smart Energy Conversion and Utilization Research. His research interests include renewable energy conversion technologies, lighting technologies, smart-grid technologies, and computational intelligence for power electronic systems. He has edited one book, and he authored eight research book chapters and over 355 technical papers, including 178 refereed journal papers in his research areas, and holds 42 patents.

Dr. Chung has received numerous industrial awards for his invented energy saving technologies. He was the Chair of the Technical Committee of the High-Performance and Emerging Technologies, IEEE Power Electronics Society, from 2010 to 2014. He is currently the Editor-in-Chief of the IEEE Power Electronics Letters and an Associate Editor of the IEEE Transactions on Power Electronics and the IEEE Journal of Emerging and Selected Topics in Power Electronics.

KIM-FUNG TSANG (M'95–SM'14) received the Associate degree in electrical engineering from The Hong Kong Polytechnic University in 1983 and the M.Eng. (by research) and Ph.D. degrees in electrical engineering from the University of Wales College of Cardiff (formerly known as the University of Wales Institute of Science and Technology), Cardiff, U.K., in 1987 and 1995, respectively.

He joined the City University of Hong Kong in 1988, where he is currently an Associate Professor with the Department of Electronic Engineering. He has published about 200 technical papers and four books/chapters.

Dr. Tsang is a fellow of HKIE, a Chartered Engineer and a member of IET, an Associate Editor and a Guest Editor of the IEEE Transactions of Industrial Informatics, an Associate Editor of *IEEE Industrial Electronics Magazine* and the IEEE ITeN, and an Editor of the *KSII Transactions on Internet and Information Systems*.

NORMAN CHUNG-FAI TSE (M'09) received the Graduate degree from The Hong Kong Polytechnic University in 1985, the M.Sc. degree from the University of Warwick, Coventry, U.K., in 1994, and the Ph.D. degree from City University of London, London, U.K., in 2007.

He is currently with the Centre for Smart Energy Conversion and Utilization Research, City University of Hong Kong. His current research interests include power quality measurement and analysis, Web-based power quality monitoring, harmonics mitigation, and building energy efficiency study.

LOI LEI LAI (M'09–SM'92–F'07) received the B.Sc., Ph.D., and D.Sc. degrees from the University of Aston and the City University of London, respectively. He was the Director of the Research and Development Centre, a Pao Yue Kong Chair Professor, the Vice President, a Professor, and the Chair in electrical engineering and a Fellow Committee Evaluator of the State Grid Energy Research Institute, China, Zhejiang University, China, the IEEE Systems, Man and Cybernetics Society (IEEE/SMCS), the City University of London, and the IEEE Industrial Electronics Society, respectively. He is currently a University Distinguished Professor with the Guangdong University of Technology, China. He is a fellow of IET, and a National Distinguished Expert in China and a Distinguished Expert in the State Grid Corporation of China. He received the IEEE Third Millennium Medal, the IEEE Power and Energy Society (IEEE/PES) UKRI Power Chapter Outstanding Engineer Award in 2000, the IEEE/PES Energy Development and Power Generation Committee Prize Paper in 2006 and 2009, the IEEE/SMCS Outstanding Contribution Award in 2013 and 2014, and the Most Active Technical Committee Award in 2016.