Using Soft Real-Time Simulation in a Hybrid Environment for Cyber-Physical Security Experiments

Béla Genge and Christos Siaterlis
Joint Research Centre, IPSC, European Commission
Via E. Fermi, 2749, Ispra (VA), 21027, Italy,
Email: {bela.genge, christos.siaterlis}@jrc.ec.europa.eu

***Abstract*—The study of complex systems, either physical or cyber, could be carried out by experimenting with real systems, software simulators or emulators. This paper presents an innovative framework for an experimentation environment that incorporates both physical and cyber systems. The proposed approach uses soft real-time simulation for physical processes (based on Simulink) and an emulation testbed (based on Emulab) for ICT components. The paper also presents two coupling algorithms that couple the simulation time to the system time, in order to achieve soft real-time simulation on a multitasking OS. The main difference between the two algorithms is the size of the critical section that enables concurrent access to a shared resource between emulated and simulated environments. The experimental results show that resource sharing between the two environments is a major concern and future algorithms should also consider a proper analysis of critical sections.**

***Keywords*-Soft real-time, Simulation, Emulab, cyber-physical, security.**

I. Introduction

Modern Critical Infrastructures (CI), e.g. power plants, water plants and transport systems, rely on Information and Communication Technologies (ICT) for their operation since ICT can lead to cost reduction as well as greater efficiency, flexibility and interoperability between components. In the past CIs were isolated environments and used proprietary hardware and protocols, limiting thus the threats that could affect them. Nowadays CIs and more specifically Networked Industrial Control Systems (NICS) are exposed to significant cyber-threats; a fact that has been highlighted by many studies on the security of Supervisory Control And Data Acquisition (SCADA) systems [1].

The study of complex systems, either physical or cyber, could be carried out by experimenting with real systems, software simulators or emulators. Experimentation with production systems suffers from the inability to control the experiment environment in order to reproduce results. Furthermore if the study intends to test the resilience or security of a system, there are obvious concerns about the potential side effects (faults and disruptions) to mission critical services. On the other hand the development of a dedicated experimentation infrastructure with real components is often economically prohibitive and disruptive experiments on top of it could be a risk to safety. Software based simulation has always been considered an efficient approach to study physical systems, mainly because it can offer low-cost, fast and accurate analysis. Software simulators can effectively model normal operations, however, they fail to capture the way computer systems fail.

Based on these facts, this paper follows a hybrid approach in between the two extremes of pure simulation and experimentation with only real components. It proposes the use of simulation for the physical components and an emulation testbed based on Emulab [2] in order to recreate the cyber part of NICS, e.g., SCADA servers, corporate network, etc. The models of the physical systems are developed in Matlab Simulink from which the corresponding 'C' code is generated using Matlab Real Time Workshop (Matlab RTW). The generated code is then executed in *soft real-time* and is able to interact with the real components of the emulation testbed.

Soft real-time systems are allowed to miss deadlines from time to time without any catastrophic consequences [3]. In the context of process model simulation the *deadlines* represent model executions. As the code generated with Matlab RTW must be executed in fixed *time steps*, i.e. the time between two model executions, missing deadlines leads to the accumulation of model execution delays. As a consequence, for the previous statement to hold in the context of soft real-time simulations there must be a mechanism that recovers the missed model executions. We implement such a mechanism within a *coupling algorithm* that couples the model execution to the system time. Based on several experimental results we show that the interaction with the other components of the emulation testbed affects the coupling mechanism through the introduction of a *deviation* time, i.e. the difference between simulation and system time. However, the deviation can be significantly reduced by maximizing the size of *critical sections*, i.e. code sequence that accesses a shared resource and must not be concurrently run by more than one thread. This approach does not follow general critical section design principles, which tend to minimize the size of critical sections in order to ensure access to as many concurrent threads as possible. Therefore our main contribution is that we show that applying general

critical section design principles in the context of soft real-time simulations leads to an accumulated deviation that makes the entire system unusable.

The paper is structured as follows. After a short overview of related work in Section II, the proposed framework is presented in Section III followed by Section IV with the presentation of two coupling algorithms that enable soft real-time simulation of process models. The performance analysis of the two algorithms is presented in Section V and the paper concludes in Section VI.

II. Related Work

An approach that uses real components for the physical parts of NICS and partly simulated ones for the cyber parts has been proposed by Chunlei, *et al.* [4]. This approach uses a real OPC (Object Linking and Embedding (OLE) for Process Control) server, the NS-2 network simulator and real PLCs and field devices to analyze NICS. NS-2 is used to simulate the enterprise network of the SCADA system. Calls from NS-2 are dispatched through software agents to the OPC server that sends Modbus packages to the physical PLCs. Although from one point of view such a testbed would provide reliable experimental data, since almost everything is real, it would be hardly able to support tests on large infrastructures such as the process system of a chemical installation, because that would require a complete industrial system to experiment with. Another approach that uses real physical devices has been proposed by Queiroz, *et al.* [5]. In this case only the sensors and actuators are real physical devices, while the remaining components, e.g., PLCs, and the communication protocols between them are implemented as OMNeT++ modules.

Other researchers have focused on simulating both SCADA and field devices. For example, Chabukswar, *et al.* [6] used the Command and Control WindTunnel (C2WindTunnel) [7] multi-model simulation environment, based on the High-Level Architecture (HLA) IEEE standard 1.3 [8], to enable the interaction between various simulation engines. The authors used OMNeT++ to simulate the network and Matlab Simulink to build and run the physical plant model. C2WindTunnel provides the global clock for both OMNeT++ and Matlab Simulink. With this approach, analyzing the cyber-physical effects of malware is not a trivial task, as it requires a detailed description of all ICT components and more importantly a detailed knowledge on the dynamics of malware, that is not always available.

Outside the context of NICS security we find several approaches for achieving soft real-time execution. The work of Brandt, *et al.* [9] introduces priority levels. When an application is detected to miss a deadline, its level is automatically decreased so that it receives more computing power. However, this approach can only be applied if there is a complete control over the task scheduling mechanism, that is not trivial on modern multitasking OS. Another approach to ensure a soft real-time execution is the optimization of algorithms and application structure. In this direction we find the work of Tromp, *et al.* [10] that reduced the overall execution time of their simulation algorithm by optimizing matrix operations and by reducing the processor cache misses. Although this is rather an application-specific approach, it shows that soft real-time can also be achieved from the user space on a multitasking OS. Our approach also discusses the importance of algorithm design in the user space, however, it goes further into the details of resource sharing between different simulators and emulators, a topic that is not covered by previous methods.

III. Framework Architecture

This section presents a new experimentation framework for studying the physical impact of cyber-threats against Networked Industrial Control Systems (NICS).

A. Process Control Architecture Overview

In modern NICS architectures, one can identify two different control layers: (i) the **physical layer** composed of all the actuators, sensors, and generally speaking hardware devices that physically perform the actions on the system (e.g. open a valve, measure the voltage in a cable); (ii) the **cyber layer** composed of all the ICT devices and software which acquire the data, elaborate low level process strategies and deliver the commands to the physical layer. The cyber layer typically uses SCADA protocols to control and manage the physical devices within the cyber layer. The "distributed control system" of the cyber layer is typically split among two networks: the *control network* and the *process network*. The process network usually hosts all the SCADA (also known as SCADA Masters) and HMI (Human Machine Interface) servers. The control network hosts all the devices which, on the one side control the actuators and sensors of the physical layer and on the other side provide the "control interface" to the process network. A typical control network is composed of a mesh of *PLC*s (Programmable Logic Controllers). From an operational point of view, PLCs receive data from the physical layer, elaborate a "local actuation strategy", and send back commands to the actuators. PLCs execute also the commands received from the SCADA servers (Masters) and additionally provide, whenever requested, detailed physical layer data.

B. Framework Overview

The proposed experimentation framework follows a hybrid approach, where the Emulab-based testbed recreates the control and process network of NICS, including PLCs and SCADA servers, and a software simulation reproduces the physical processes. The argument for using emulation for the cyber components is that the study of the security and resilience of computer networks would require the simulation of all the failure related functions, behaviors and

states, most of which are unknown in principle. On the other hand software simulation is a very reasonable approach for the physical layer due to small costs, the existence of accurate models and the ability to conduct experiments in a safe environment.

The architecture, as shown in Figure 1, clearly distinguishes 3 layers: the cyber layer, the physical layer and a link layer in between. The cyber layer includes regular ICT components used in SCADA systems, while the physical layer provides the simulation of physical devices. The link layer provides the glue between the two layers through the use of a shared memory region.

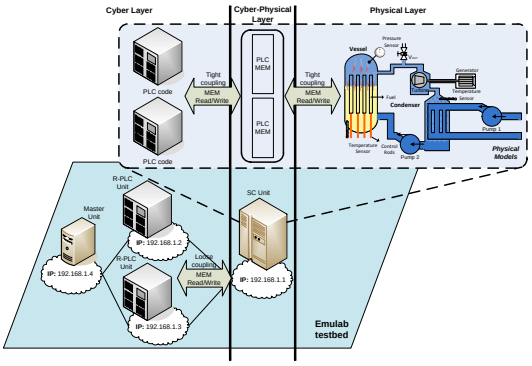

Figure 1: Architectural overview

The physical layer is recreated through a soft real-time simulator that runs within the *SC* (Simulation Core) unit and executes a model of the physical system. The simulator's execution time is strongly coupled to the timing service of the underlying operating system (OS) and is run in soft real-time. Throughout the paper the term *time step* is used to denote the time between two successive executions of the physical model in the simulator.

The cyber layer is recreated by an emulation testbed that uses the Emulab architecture and software [2] to automatically and dynamically map physical components (e.g. servers, switches) to a virtual topology. In other words the Emulab software configures the physical topology in a way that it emulates the virtual topology as accurately as possible.

Besides the process network, the cyber layer also includes the control logic code, that in the real world is run by PLCs. In the proposed approach the control code can be run sequentially or in parallel to the physical model. In the sequential case, a *tightly coupled* code (TCC) is used, i.e. code that is running in the same memory space with the model, within the SC unit. In the parallel case a *loosely coupled* code (LCC) is used, i.e. code that is running in another address space, possibly on another host, within the *R-PLC* unit (Remote PLC). The main advantage of TCCs is that these do not miss values generated by the model between executions. On the other hand, LCCs allow running PLC code remotely, to inject (malicious) code without stopping the execution of the model, and to run more complex PLC emulators. As control code is implemented within TCCs and LCCs, throughout the paper these terms are used to denote PLCs.

The cyber-physical layer incorporates the PLC memory, seen as a set of registers typical of PLCs, and the communication interfaces that glue together the other two layers. Memory registers provide the link to the inputs (e.g. valve position) and outputs (e.g. sensor values) of the physical model.

C. Implementation Details

Prototypes of SC, R-PLC and Master units have been developed in C# (Windows) and have been ported and tested on Unix-based systems (FreeBSD, Fedora and Ubuntu) with the use of the *Mono* platform. The implementation allows TCCs to be provided either as C# source files or as binary DLLs, both dynamically loaded at run-time. C# source files are dynamically loaded, compiled and executed at run-time using .NET's support for dynamic code execution. C# source files provide the ability to implement PLC code without the need of a development environment at the cost of longer execution time. At this time, LCCs are written in C# and must be compiled together with the SC-unit. Matlab Simulink was used as the physical process simulator (physical layer) since it is a general simulation environment for dynamic and embedded systems. The communication between SC and R-PLC units is handled by .NET's binary implementation of RPC (called *remoting*) over TCP. Currently, for the communication between the R-PLC and the Master units, the Modbus over TCP protocol is implemented.

IV. Coupling Algorithms

The main role of the SC unit is to provide a soft real-time execution of the process model. This is achieved through a *coupling algorithm* (CA) that couples the execution time of the model to the system time. The main concern with the design of such an algorithm proves to be the PLC memory that is a shared resource between the SC unit and R-PLC units. This means that the PLC memory needs to be protected against simultaneous access, which introduces the problem of *critical sections* known from the field of concurrent programming. Intuitively, a CA would need to run the process model only once for each time step. However, in a multi-tasking environment such an approach introduces accumulated deviations as the OS can stop and resume threads without any intervention possibilities from the user space. Based on this observation, we state that the coupling algorithm must include a certain mechanism to run the process model multiple times in each time step in order to reduce deviations.

Following this section we present two coupling algorithms: the first one is designed to minimize, while the

second one is designed to maximize the size of the critical section. Both coupling algorithms run the process model multiple times in each time step in order to reduce deviations. The main difference between the two is the positioning of the *acquire* and *release* function calls that mark the beginning and the end of the critical section, respectively. The protected resource is the PLC memory accessed by the physical model, TCCs, and in parallel by LCCs. The first algorithm minimizes the size of the critical section by releasing the resource when it is no longer used. This ensures that more R-PLC units can access the resource in each time step. The second algorithm maximizes the size of the critical section by keeping the resource locked until the deviation is reduced even if the resource is not used at all times. The second algorithm clearly violates general design principles for critical sections, as it keeps the resource locked when it is no longer used. However, in the following section we show that the second algorithm reduces deviations and enables the system to support even 100 PLCs, which is not possible with the first algorithm.

A. Minimal Coupling Algorithm

The CA reads from the PLC memory before running the process model in order to get the necessary model inputs (e.g. valve position, digital output) and writes to the PLC memory afterwards in order to provide the model outputs (e.g. pressure, temperature) to the other units. Within this context, the Minimal Coupling Algorithm (MiCA) ensures a minimal accessing time of the PLC memory. More specifically, it acquires the PLC memory resource only for two operations: reading and writing. After the read operation it immediately releases the resource and runs the process model. Afterwards, the resource is acquired again in order to write back the output of the model.

The structure of the MiCA is given in Algorithm 1. It starts by setting the simulation time (t_{sim}) with the help of the $@GetSysTime()$ function. The value of t_{sim} is then updated after each time step within the $while$ structure. Because the MiCA is running in user space of a multitasking OS, it must be prepared to handle task changes during which the current thread is not running. For this reason the MiCA includes a second $while$ structure that ensures a minimal difference between the system time and the running time of the process model. In other words, it provides a minimal *deviation* from the system time by running the process model as many times as necessary for reducing the time difference. Within this structure, the $@AcquirePLCMem()$ and $@ReleasePLCMem()$ functions are used to acquire and release the shared resource, respectively. The PLC memory is accessed through the $@ReadPLCMem()$ and $@WritePLCMem()$ functions while the physical model and TCCs are run with the $@RunPhysical()$ and $@RunTCC()$ functions, respectively. TCCs are running sequentially with the model and need to access the PLC memory before running the control code and afterwards. Thus, the shared resource must be acquired and released once again.

Calls received from LCCs are executed by a different thread than the one running Algorithm 1. These calls use the same $@AcquirePLCMem()$ and $@ReleasePLCMem()$ functions to acquire and release the PLC memory. After acquiring the resource, the calling thread reads or writes the PLC memory, according to the instructions received from LCCs. Because of its simplicity and space considerations this code is not presented in this paper.

Algorithm 1: Minimal Coupling Algorithm - MiCA

```
begin
    t_sim ← @GetSysTime()
    while FOREVER do
        t_diff ← @GetSysTime() − t_sim
        if t_diff ≥ TIMESTEP then
            t_step ← TIMESTEP
            while t_diff > t_step do
                @AcquirePLCMem()
                data_in ← @ReadPLCMem()
                @ReleasePLCMem()
                data_out ← @RunPhysical(data_in)
                @AcquirePLCMem()
                @WritePLCMem(data_out)
                @ReleasePLCMem()
                @AcquirePLCMem()
                @RunTCC()
                @ReleasePLCMem()
                t_step ← t_step + TIMESTEP
            t_sim ← t_sim + t_step
```

B. Maximal Coupling Algorithm

The Maximal Coupling Algorithm (MaCA) ensures that for each time step the shared resource is acquired and released only once. This means that the thread running the MaCA will only need to wait once to acquire the shared resource, after which it can run the model as many times as needed to reduce the deviation. In contrast, the thread running the MiCA must wait for the resource to become available again, thus accumulating more deviations after each release.

As shown in Algorithm 2, the PLC memory resource is acquired only once, before the second $while$ structure, and is released after the execution of the same structure. With this approach, the algorithm can not be blocked by the thread executing calls received from LCCs while reading and writing the PLC memory. In the next section we show that although minor, the changes in the algorithm have a major effect on the deviation.

Algorithm 2: Maximal Coupling Algorithm - MaCA

```
begin
    t_sim ← @GetSysTime()
    while FOREVER do
        t_diff ← @GetSysTime() − t_sim
        if t_diff ≥ TIMESTEP then
            t_step ← TIMESTEP
            @AcquirePLCMem()
            while t_diff > t_step do
                data_in ← @ReadPLCMem()
                data_out ← @RunPhysical(data_in)
                @WritePLCMem(data_out)
                @RunTCC()
                t_step ← t_step + TIMESTEP
            @ReleasePLCMem()
            t_sim ← t_sim + t_step
```

V. Performance Evaluation

The prototype of the proposed framework was developed and evaluated in the Joint Research Centre's (JRC) Experimental Platform for Internet Contingencies (EPIC) laboratory. The Emulab testbed includes nodes with the following configuration: FreeBSD OS 8, AMD Athlon Dual Core CPU at 2.3GHz and 4GB of RAM. In total, the experimental setup consisted of 8 hosts, 1 for running the SC unit, 1 for running the Master unit and 6 other hosts to run at most 100 R-PLC units (depending on the setup of the experiment). The previously presented algorithms have been implemented in the SC unit. Each R-PLC unit issues a write operation in each time step. Through our experiments we used the model of a Boiling Water Power Plant [11].

The measured deviation for the system using a MiCA is shown in Figure 2. For up to 10 PLCs (i.e. LCCs) the deviation increases up to a 1ms time step, after which it decreases down to 0.0021ms (for 100 LCCs). The explanation for the increasing deviation is that larger time steps lead to a larger number of R-PLC requests received in one time step. For time steps larger than 1ms the number of simultaneous accesses to the PLC memory decreases, thus the deviation is also decreased. For more than 10 LCCs and time steps smaller than 100ms, the deviation accumulates and exceeds in time 1s (i.e. after only a few seconds), highlighted with a dashed line. In these cases the system becomes unusable, as the MiCA tries to decrease the deviation by subsequently acquiring and releasing the shared resource. After each release operation the R-PLC units acquire the resource and block the thread running the MiCA when it tries to acquire it again. Nevertheless, the system is able to maintain a deviation smaller than 0.5ms for up to 20 LCCs, which is not a negligible performance.

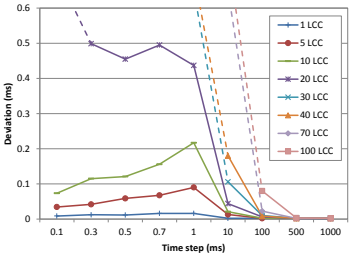

Figure 2: Deviation for MiCA

The deviation for a system using a MaCA is shown in Figure 3. Similar to the previous algorithm, the results show that the deviation increases up to a 1ms time step, after which it decreases down to 0.0022ms (for 100 LCCs). However, in this case there is a clear improvement for all series, as the accumulation for the deviation is completely eliminated. Although the deviation increases up to 34ms (for 100 LCCs), with the MaCA the system is now able to run with even the extreme case of 100 LCCs. As a result, by simply increasing the size of the critical section we are able to decrease the deviation up to 30 times (i.e. more than 1000ms for the MiCA and a maximum of 34ms for the MaCA).

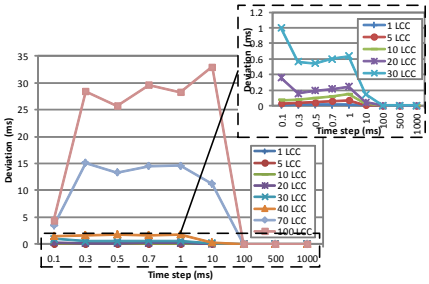

Figure 3: Deviation for MaCA

Another parameter that illustrates the impact of MaCA is the average deviation given as a percentage from the value of the time step. In Figure 4 we compare this effect for up to 30 LCCs and a maximum deviation of 1s for MiCA, although this accumulates in time to larger values. As expected, for the majority of cases a system with MaCA shows a much lower deviation percentage than a system using MiCA. The two exceptions (i.e. for 0.1ms and 10ms) do not show major differences from MiCA, if we consider that in the first case the value for MiCA could be even larger (we assumed a 1s deviation) and for the second case the difference is only of 0.45%. The execution of TCCs for both algorithms does not introduce any additional deviations to the ones generated by task switchings. We measured a maximum deviation of 80μs for 100 TCCs with 0 LCCs. Due to space considerations and the fact that TCCs do not have a major effect on deviations

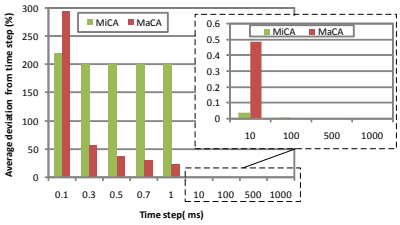

Figure 4: Average deviation percentage from time step

we do not provide a graphical representation for this setting.

Although the changes introduced in the MiCA reduce the deviation, these have an opposite effect on the number of LCCs that manage to read / write the remote PLC memory in each time step. With smaller time steps the number of simultaneous requests to access the PLC memory grows rapidly, also leading to *missed* operations, i.e. operations that could not be executed in the current time step. The term *miss rate* is used to denote the percentage of missed read/write operations by LCCs in each time step. By missing certain values, LCCs can not react to the changes of the model.

Although TCCs will not miss any values generated by the model (as TCCs run sequentially with the model), this is not the case for LCCs. The causes for this are: (i) execution time is not synchronized between units, (ii) nodes are running multitasking OSs, and (iii) there is a networking infrastructure that introduces additional delays. The results given in Figure 5 show that for time steps smaller than 10ms there is an average miss rate above 80% for both algorithms. However, the miss rate decreases significantly for time steps above 10ms and reaches zero for 100ms. By comparing the miss rate for the two algorithms we notice only a slight difference of 1.48% increase for the MaCA. Nevertheless, if we consider that the second algorithm provides a 30 times improved deviation, we can clearly state that the slight loss of 1.48% miss rate is negligible. On the other hand, if a system is highly sensitive in terms of miss rates, it can still implement the MiCA with the restriction that it can not include more than 10 PLCs.

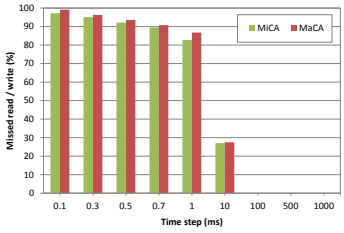

Figure 5: Average miss rate

VI. Concluding Remarks

This paper presents a new framework for the security analysis of NICS following a hybrid architecture that uses an emulation testbed to recreate cyber components such as PLCs and SCADA Masters and software simulation for capturing the physical processes. One of the main contribution of the paper is the analysis of performance differences that are introduced by the algorithms that perform the coupling of soft real time simulation with real components of an emulation testbed. The paper proposes two algorithms for implementing soft real-time simulation by coupling the simulation time to system time. The analysis of the proposed algorithms shows that resource sharing between the two environments (i.e. emulated and simulated) is a major concern and future algorithms should also consider a proper analysis of critical sections. As future work we intend to use the proposed framework for studying the propagation of perturbations in cyber-physical environments.

References

[1] I. N. Fovino, A. Carcano, M. Masera, and A. Trombetta, "An experimental investigation of malware attacks on SCADA systems," IJCIP, Vol. 2, No. 4, pp. 139–145, 2009.

[2] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," Proc. of the 5th Symposium on OS Design and Implementation, USA, pp. 255–270, 2002.

[3] M. Corti, "Approximating the Worst-Case Execution of Soft Real-Time Applications," PhD Thesis, Swiss Federal Institute of Technology Zurich, Doctoral Thesis ETH No. 15927, 2005.

[4] W. Chunlei, F. Lan, and D. Yiqi, "A Simulation Environment for SCADA Security Analysis and Assessment," Proc. of 2010 International Conference on Measuring Technology and Mechatronics Automation, China, pp. 342–347, 2010.

[5] C. Queiroz, A. Mahmood, J. Hu, Z. Tari, and X. Yu, "Building a SCADA Security Testbed," Proc. of the International Conference on Network and System Security, pp. 357–364, 2009.

[6] R. Chabukswar, B. Sinopoli, G. Karsai, A. Giani, H. Neema, and A. Davis, "Simulation of Network Attacks on SCADA Systems," Workshop on Secure Control Systems, 2010.

[7] S. Neema, T. Bapty, X. Koutsoukos, H. Neema, J. Sztipanovits, and G. Karsai, "Model Based Integration and Experimentation of Information Fusion and C2 Systems," Proc. of 12th Conference on Information Fusion, USA, pp. 1958–1965, 2009.

[8] J. O. Calvin and R. Weatherly, "An introduction to the high level architecture (HLA) runtime infrastructure (RTI)," Proc. of the 14th Workshop on Standards for the Interoperability of Defence Simulations, Orlando, USA, pp. 705–715, 1996.

[9] S. Brandt, G. Nutt, T. Berk, and M. Humphrey, "Soft real-time application execution with dynamic quality of service assurance," Proc. 6th Int. Workshop on Quality of Service, USA, pp. 154–163, 1998.

[10] J. Tromp, D. Komatitsch, V. Hjörleifsdóttir, Q. Liu, H. Zhu, D. Peter, E. Bozdag, D. McRitchie, P. Friberg, C. Trabant, and A. Hutko, "Near real-time simulations of global CMT earthquakes," Geophysical Journal International, Vol. 183, pp. 381-389, 2010.

[11] R. D. Bell and K. J. Åström, "Dynamic models for boiler-turbine alternator units: data logs and parameter estimation for a 160MW unit," Lundt Institute of Technology, Sweden, Report TFRT–3192, 1987.