Towards a HLA-based Hardware-In-the-Loop simulation runtime

Eugene Chemeritskiy
The Faculty of Computational Mathematics and Cybernetics
Lomonosov Moscow State University
Moscow, Russia
tyz@lvk.cs.msu.su

This paper considers the possibility of the Distributed Real-time and Embedded (DRE) system simulation with CERTI, a general-purpose distributed simulation runtime based on the High Level Architecture (HLA) standard. Although it does not address DRE simulation and a number of issues should be resolved, the paper focuses on CERTI performance in the first place and compares it with a specialized DRE simulation tool. During the analysis of the comparison results, we make proposals on CERTI attuning and the design of an efficient HLA-based runtime.

Hardware-In-the-Loop Simulation; High Level Architecutre; CERTI;

I. Introduction

A. DRE simulation

Distributed Real-time and Embedded systems (DREs) are used in a wide range of electronic devices. Although DREs of household appliance usually contain only a few sensors and processing units, some complex equipment, e.g. onboard car, plane, and ship systems, is often composed of hundreds of devices connected to each other by dozens of diversified data transmitting channels.

The wide scope of DREs often imposes various restrictions on its components. Some DREs have to provide a certain performance, using a limited amount of power, resist an aggressive environment conditions, or keep within certain limitations on physical sizes and weight. In case the existing equipment does not meet these requirements, some devices have to be redesigned.

The missing DRE devices are usually developed by several independent workgroups. Their tasks have significantly varying complexity and assigned periods of time. As a result, devise prototypes constantly have different readiness degrees, and their joint trials are often impossible. However, the majority of errors are detected at the stage of component integration, when the prototypes are nearing its completion and the cost of error correction is reaching its maximum.

DRE simulation provides a common approach for early detection of integration errors. It includes the following steps. At the stage of DRE blueprinting, developers create its coarse component-wise simulation model. It serves for verification of the simplest DRE properties and detection of the related design errors. The further development goes, the further device models are refined. Simulation accuracy gradually grows, and verification reveals more and more tricky errors. Finally, as the hardware prototypes become available, they join the simulation instead of their software counterparts. To sum it up, exploring the current model properties at each stage, DRE developers can identify and shortly fix device integration errors as soon as it possible.

Considered Hardware-In-the-Loop Simulation (HILS) composes the software models with the hardware equipment and requires a specialized simulation hardware-software environment to interconnect them to each other. The software part of this environment, the simulation runtime encapsulates the details of communication with the diverse DRE components and provides a common API over it. However, the runtime cannot hide their difference completely. Hardware devices are often unable to work properly without a strict binding to astronomical clock. For example, its implementation may accept the reply for only a short period of time after the request send. These real-time constraints impose additional requirements to the simulation runtime. Therefore, HILS runtime significantly differs from a general-purpose one [1].

B. The HILS STAND simulation environment

For years the Computer Systems Laboratory (CS Lab) of Lomonosov Moscow State University has been developing its own hardware-software environment for Hardware-In-the-Loop simulation called HILS STAND [1]. Actually the roots of HILS STAND go to more general discrete-event simulation system DYANA [2], developed by CS Lab in early 1990s. Since, DYANA has been serving as a basis for a number of experiments on applicability of new approaches to simulation of the diverse computer systems. Started as one of the proof-of-concept DYANA branches, HILS STAND has been used to accomplish a number of HILS projects and gradually evaluated into a powerful simulation suite.

The core of the HILS STAND suite is formed by the specialized discrete-event runtime. This runtime provides several independent model execution modes. While in as-fast-as-possible mode, the HILS STAND runtime provides an efficient execution of software-only models and acts similar to a general-purpose runtime. The soft, hard and scaled real-time modes address the different kinds HILS in the first place. Being in any of this mode, the runtime executes device models

This work was supported in part by the Ministry of education and science of the Russian Federation under Grant "Development of an integrated environment and complex analysis methods for distributed real-time computer systems functioning".

with the corresponding time constraints and allows their interactions with the connected hardware devices.

Besides the runtime, the suite includes a number of subsidiary simulation tools. For example, HILS STAND provides an integrated simulation data collector and a dynamic visualizer with the ability to modify model parameters during its execution and enables a human-controlled simulation. Although the runtime provides C++ API and encapsulates the details of interactions among the simulation participants, HILS STAND does not imply the model to be developed in a pure general-purpose language. Instead, it provides a specialized high-level model description language, its translator, and the corresponding IDE. Actually, these developments have a rich history either [3].

Because of the large number of DRE devices and a certain complexity level of their models, HILS STAND instances usually include a number of nodes connected by a private LAN. The nodes have a similar configuration, except some additional interfaces for external hardware simulation participants. The additional hardware channels can also interconnect several nodes directly. This configuration is used for experiments with the corresponding channel controllers and the other switching equipment [1].

C. *An advanced simulation runtime*

Although the HILS STAND is actively used in a number of different HILS projects, CS Lab has never ceased to follow the trends and test new simulation approaches. Recently it started new project on the rethinking of HILS environment and the development a new tool-chain, combining the experience gained in over twenty years of simulation experiments with latest applicable technology efforts. The research list of this large-scale project includes a new language for DRE simulation models; integrated trace tracking and visualizing tools; new verification engine; and an advanced HILS runtime, which is the subject of this paper.

During the last 60 years, discrete-event simulation runtimes made a significant advance [4]. There are several different classifications, but it is a common practice to divide them into a number of generations associated with some historical trends. One of the most noticeable trends now is a standardization of the runtime API. Thus, we believe it is giving a birth to the new runtime generation. Unfortunately, there is no any off-the-rack and well-fitted standard for HILS runtimes. However, exploring of adjoining simulation areas revealed some attempts to use High Level Architecture (HLA) for real-time simulation [5] and it is pretty close to HIL. So the idea of HLA adoption for HILS runtime appeared.

Although there are a lot of different HLA-based simulation runtimes, each of them requires a large amount of additional work. As it was shown by the analysis [6], CERTI happened to be the best initial approximation for the advanced HILS runtime we want to make. However, CERTI do not initially target real-time and HILS, and a number of related issues have to be resolved. Briefly considering their whole scope, the paper gives the first priority to performance of a HILS runtime. In particular, the paper introduces a couple of benchmarks that revealed significant advantage of HILS STAND over the CERTI. Due to a lower performance, out-of-the-box CERTI version cannot be used for the scope of simulation tasks HILS STAND can easily manage. Therefore, the paper contains the analysis of its architectural drawbacks and introduces a number of proposals on their reduction.

II. A HLA-BASED HILS RUNTIME

A. *Standardization trends*

Simulation as a method for exploration of diverse object properties and regularities among them outruns the advent of computers for many years. However, its rapid development started after the complex mathematical calculations had been assigned to fast and reliable computers. In the beginning of the 1950s, the term simulation acquired the default meaning of digital computer simulation. Subsequently the simulation was defined as a combination of designing of the observed system model and holding the necessary experiment set on digital computers [7].

From the very beginning of the simulation history the observed systems always tended to be represented in deeper detail level. This tension results in the increasing size and complexity of developed simulation model. This growth required a respective performance increase from computer systems, and this fact resulted in emergence of parallel simulation systems. These systems share the simulation task across multiple computing nodes. Typically such systems were implemented locally within the organization that wanted to use it.

The complexity of the models was not the only factor leading to computer simulation tool evolution. The scope of simulation has been growing either. After new simulation problem types appeared, the related requirements were imposed to modeling and simulation tools. For instance, distributed simulation is often required in case of joint product development when different product component are produced by a number of workgroups located in different organizations. This type of simulation intends encompassing of several geographically separated simulation systems, which in turn may consist of a single compute node, or be a parallel system. Historically, the appearance of this task type led to the creation of distributed simulation systems that provide an essential set of services to the simulation participants and ensure its consistent behavior [8].

Currently we believe that the next step in the runtime evolution is a standardizing of the distributed system interfaces. Uniform interfaces provide possibility to combine among a variety of independent simulation systems and create general models that can be handled by every distributed system corresponding to the standard specifications. One of these standards is described in the next section.

B. *The HLA distributed simulation standard*

HLA is a conventional standard in the field of distributed simulation. The roots for the HLA stem from distributed virtual environments. Such environments are used to connect a number of geographically distant users. The HLA standard is a conceptual heir of Distributed Interactive Simulation (DIS), which is a highly specialized simulation standard in the domain of training environments [8]. The primary mission of DIS is to enable interoperability among separated simulation systems and to allow the joint simulation of their participation. HLA standard remains relevant to the DIS principles and even extends them.

HLA appeared in 1993, when the Defense Advanced Research Projects Agency (DARPA) designated an award for developing of an architecture that could combine all known types of simulation systems into a single *federation*. The HLA standard initially addressed all kinds of as-fast-as-possible, soft and hard real-time, discrete-event and time-driven, fully-synthetic, human- and hardware-in-the-loop distributed simulations. However, hard real-time constraints were not supported until the latest HLA standard version, namely IEEE 1516-2010 (Evolved) released in the very end of 2010 [9]. The majority of HLA-based simulation tools were built on the previous HLA standard versions and do not offer a full HLA Evolved support yet.

Thereby, HLA-based HILS became possible quite recently and any researches in this area are innovations in some sense. However, these researches seem to be prospective because of a number of benefits HLA gives. At first, HLA strict support by both the runtime and the models provides their guaranteed compatibility. It means that HLA model developed with one runtime can also be used with other runtimes without any modification. In fact, HLA forms an independent market of out-of-the-box simulation models which can be used with any HLA-compatible simulation runtime.

Secondly, HLA is used as an external simulation interface by some non-distributed runtimes. This peculiarity enables joined simulation encompassing diversified runtimes and, consequentially, different model types. For example, a single simulation can include both time-driven fully-synthetic and discrete-event hardware-in-the-loop models simultaneously, and their developers do not have to adjust their models for this cooperation.

In addition, there are a lot of subsidiary runtime-independent HLA-based simulation tools, such as statistic collectors, simulation analyzers, high-level model describing languages and corresponding IDEs. These tools operate at the model level over the HLA API and do not require any additional support from the simulation runtime. Therefore, they can be reused with any runtime implementation.

HLA specification introduces its own terminology generally used by the developers of HLA-based simulation tools, and CERTI is not an exception. Therefore, we include a short notion of fundamental HLA terms. The simulation runtime specified by HLA is named the Run Time Infrastructure (RTI). RTI provides services a number of joined federates - simulation participants of any kind. The association of all federates forms federation.

C. *CERTI brief description*

CERTI is a HLA-compliant RTI developed by the French Aerospace Laboratory (ONERA). The project started in 1996 and its primary research objective was the distributed simulation itself whereas the appeared HLA standard was the project experiment field. CERTI implementation started with the implementation of the small subset of RTI services, and was used to solve the concrete applications of distributed simulation theory [10].

Since the CERTI project was open sourced in 2002, a large distributed simulation developer community has been formed around the project. In many ways due to contributions of enthusiasts, the CERTI project has grown from basic RTI into a toolset including a number of additional software components that may be useful to potential HLA users.

The CERTI project has always served a base for researches in the domain of distributed simulation, and a number of innovative ideas have been implemented with its use. Thus, the problem of confidential data leak was solved in context of CERTI RTI architecture, and the considered RTI guarantees secure interoperation of simulations belonging to various mutually suspicious organizations [11]. The certain interest for the considered project is a couple of application devoted to high performance and hard real-time simulation.

In spite of HLA is initially designed to support fully distributed simulation applications, it provides a framework for composing not necessarily distributed simulations. Thereby there was created an optimized version of CERTI devoted to simulation deployed on the same shared memory platform and composed simulation running on high-performance clusters [12].

Some experience could also be adopted from ONERA project on simulation of satellite spatial system. Each federate in this federation is a time-stepped driven one. It imposes an additional requirement of hard real-time: the simulation system should meet the deadlines of each step and synchronize the different steps of the different federates [13].

Despite the distribution of commercial products, the project development is still continuing in accordance with the HLA simulation standard progress. Thus, CERTI supports HLA IEEE 1516-2000 version since 2010 in addition to previous DMSO 1.3 version.

D. *Designing a CERTI-based Runtime*

There are a lot of difficulties on a way to a CERTI-based HILS runtime. First of all, the supported version of HLA standard does not currently address real-time simulation and a fortiori it does not address HIL. First, IEEE 1516-2000 specifications do not provide any method to specify end to end prediction requirement for federate. Second, CERTI encodes reliable and best-effort transportation types with TCP and UDP network protocols which are not suitable for real-time simulation. Finally, CERTI works over the operating system and is unable to control its resources. All the listed paragraphs have a significant affection to amount and predictability of the runtime overhead crucial for any real-time simulation [5].

Second group of issues concerns the hardware integration during the HILS. The runtime should have an extendable support of the diverse data transmitting channels. This fact implies a number of additional restrictions to both hardware and software components of the simulation system. For example, the hardware devices usually have strict data message format specifications. Therefore, RTI cannot use the only message to transmit both internal service data and federation one.

The final design challenge is the reuse of legacy tools from the HILS STAND software suite. Some of its components, such as the dynamic simulation visualizer and the generator of fault injections, cannot be efficiently implemented over the existing HLA interface and should be integrated into the RTI. Actually, their integration leads to additional research and development subtasks and requires a number of problems to be resolved.

Although each of the listed problems is important, this paper is devoted to the provision of the HILS STAND-comparable runtime performance level. Currently, HILS STAND is used to perform HILSs by a number of different DRE development projects and we are curios if the HLA-based simulation runtime is able to execute models with the similar complexities and real-time constraint sizes. The remainder of this paper is devoted to this issue.

III. CERTI PERFORMANCE EVALUATION AND ARCHITECTURAL ANALYSIS

A. *Runtime benchmarking*

During the HILS, each of the hardware participants interacts to the other DRE components according to a predefined time-related behavior specification. If the runtime does not meet requirements of this specification, the device may work incorrectly. It is simple to slow down a fast software model to correspond the device speed, but it is not possible to meet these requirements if the model works slower than the hardware expects. Thus, the speed of event handling is a crucially important property for any HILS runtime. Actually, its value can be used to determine the complexity of simulation tasks the runtime can efficiently solve. The smaller event handling time of the runtime, the wider range of simulation tasks it can solve. Moreover, the faster runtime works, the smaller its requirement to the hardware. For example, a slower runtime may need more nodes to run the same simulation model.

Making an assessment of CERTI applicability to the range of usual HILS STAND tasks, we choose two simple time-regulated client-server models from the HILS STAND test-suit and run them in as-soon-as-possible mode in both runtimes. Each of these models consists of a single server and a single client. The client sends messages to the server. Each message contains one integer parameter, whose value is decremented after each send, until it reaches zero. Thereby, the initial value of this parameter also sets a number of client messages to be transmitted. In the first model the server records the received values and works as a simple registrar. In the second model server also sends back to the client every message it receives, and the client do not send the next message until it gets a reply. The remainder of this paper refers these models as “Avalanche” and “Ping-Pong” tests respectively.

Although the described models are pretty simple, the similar simulation models are often used for the same purposes [14-15]. Federates of the Ping-Pong test are actually executed consequently. After the message send, client waits for a server reply. In a similar manner, server waits for the message instantly replies back to the client. Thereby, the time of simulation reflects the speed of message transmission rather accurate and can be used as a performance index for a runtime response time. Avalanche, in contrary, allows a fully parallel and logically unrestrained federate execution. Thus, the whole runtime can be considered as a media for data message transmission. Therefore, the simulation time can be treated as a reflection of a runtime throughput.

Both systems were tested using a hardware bench composed of two identical nodes. Each node ran a single model component, either client or server. The simulation time was measured by each model component independently of each other. The timer started right after the initial synchronization and stopped when the component had been ready to resign. Final results were formed as an average of two component readings for each model configuration.

As it is clearly shown by the benchmark results (Table I), overall CERTI performance is a several times lower than the one of HILS STAND. Although these results reduce the range of acceptable simulation tasks dramatically, the usual real-time requirements still accept CERTI as a HIL runtime. However, increase of its performance becomes an important direction of further development. The remainder of this segment presents a deeper CERTI analysis and introduces some proposals on its refinement.

TABLE I. THE AFFECTION OF MESSAGE NUMBER TO SIMULATION EXECUTION TIME, MS

Message number	**Avelanche**		**Ping-Pong**	
	CERTI	***HILS STAND***	***CERTI***	***HILS STAND***
10	4,1	1,6	10,2	2,3
100	38,1	7,6	94,4	22,8
1000	399,7	84,8	884,6	228
10000	6063	1127,6	8770,7	2280
100000	60601	11722,1	87643,2	22800

B. *CERTI architecture analysis*

Being a distributed simulation middleware, RTI provides a number of joined federates with API specified by the HLA standard. The main purpose of this API is to encapsulate any details of communication among the joined federates, network communication included. Thus, RTI includes a number of remote components corresponding to a number of federates joined. These components are generally known as Local RTI Components (LRCs).

Maintaining the federation consistency, RTI constantly synchs a set of joined federates. Therefore, the efficiency of their coordination affects the overall RTI performance significantly. Fully distributed architecture implies equal and self-sufficient LRCs, and its implementation requires complicated consensus algorithms. Developers usually avoid the excessive complexity by introducing the Central RTI Component (CRC) that stores shared data and implements some synchronization algorithms locally. Both centralized and decentralized RTI architectures have certain weak and strong sides, and their reasonable combination is a first cornerstone of the efficient RTI implementation [13]. This segment considers the approach implemented by CERTI.

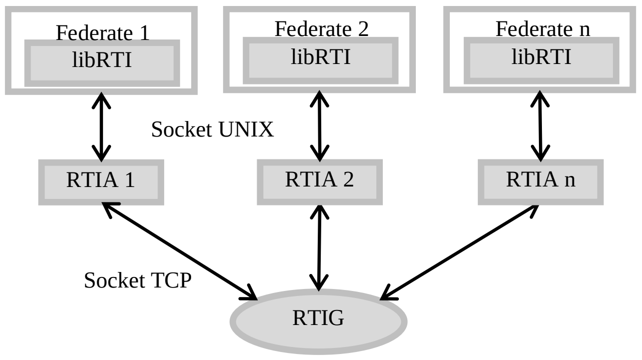

Figure 1. CERTI RTI architecture

Generally speaking, CERTI RTI consists of three components: RTI Gate (RTIG), RTI Ambassador (RTIA) and libRTI. RTIG is a process that runs on a separate host and serves as CERTI CRC. RTIA is a process that runs on the same host federate runs. Therefore, the number of RTIAs equals to the amount of joined federates. Both RTIG and RTIA are single-thread processes. Whereas they form RTI internals, libRTI runtime library implements API specified by HLA. libRTI links to the joined federate and connects it to the corresponding RTIA process by pipe (Unix socket). An aggregate of RTIA and libRTI forms CERTI LRC. Communication between CRC and LRCs goes through the network sockets [10]. Figure 1 presents a visual representation of the described architecture.

RTIA processes never communicate to each other directly. All data exchange among them goes through the RTIG. Thus, CERTI bases on a fully centralized architecture and its CRC component coordinates the joined federates single-handedly. Indeed, RTIG implements the most of RTI services whereas the sole purpose of RTIA and libRTI is the formation of a convenient communication infrastructure between RTIG and a number of joined federates. In other words, LRCs of CERTI generally serve as connectors between CRC and the end simulation participants.

Summing it up, CERTI RTI uses a fully centralized architecture and has a strong CRC. The simplicity of this organization gives a number of benefits to the RTI developers. First of all, concentration of all the control inside of a single process simplifies the implementation of RTI services. The absence of any direct links among the LRCs does not require network consensus algorithms and reduces the corresponding synchronization overhead significantly. Modification of a federation state made by RTIG is instantly propagated among all the related federates and all data received by LRCs can always be used without any additional conformation.

The second centralization profit is the simplicity and deterministic of communication among the distributed RTI components. Each data exchange among the joined federates always go through RTIG, where all the synchronization issues are solved locally and, therefore, do not require any network communication at all. Thus, coordination of the joined federates with RTIG always requires a certain and relatively small number of network messages, and uses network bandwidth rather efficiently. In contrast, fully distributed architecture, that include a number of equal self-sufficient LRCs and does not include any CRC, requires a complex synchronization algorithms, that are usually imply an intensive network communication to the consensus.

Third, strong CERTI centralization allows accelerated implementation and testing of the innovations. The proof-of-concept for a new runtime algorithm can be implemented locally inside RTIG. The developers get rid of complex network interactions and asynchronous changes of federation state. Their implementation should run in context of a single threaded process only. This peculiarity of CERTI, composed with its open source code, has proved this RTI as good foundation for the diverse simulation researches.

Finally, the centralized single-host execution of RTIG provides a convenient way to isolate the certain RTI subsystem and develop a number of diverse “surgical” benchmarks for them. The distributed kind of some RTI services (such as time and object management services) makes it really hard to get a fair estimation of their in-field implementation efficiency. The true results are often shaded and distorted by the network communication overhead that is hard to predict and, a fortiori, to avoid.

However, the same centralized architecture causes a number of problems. The most crucial of them is an excessive load of the RTIG. In case of small federations, composed of a few federates, RTIG does its service well. But the more federates join, the more RTIG is loaded, and the dependency is not linear. Each joined federates results in a new communication flow, and an increase in the complexity of its processing. For example, a single interaction send often leads to a number of notifications to its recipients. There comes a moment RTIG is unable to process the incoming data flow fast enough and becomes a bottleneck. In this case, it constantly makes the joined federates wait, and, as a result, federation is executed merely consequently. The next segment presents some approaches that can conceptually improve the current CERTI architecture and in some ways mitigate its problems.

IV. CERTI ATTUNING PROPOSALS

A. *Layered architecture*

As it was shown in the previous section, fully centralized architecture of CERTI results in an excessive CRC load during the execution of large federations, and it is a serious design drawback. There are a lot of RTI implementations that compose centralized and decentralized approaches in a more equitable way and get better performance in return [14]. However, architectural mixture requires a revolutionary alteration of CERTI internals and eliminates all the benefits of its clear and simple component structure. This section presents an alternative approach to CRC load reduction based on a layered RTI architecture.

Although each federate represents a certain component of real system, the level of their abstraction can be volatile and do not reflect logical structure of the system. For example, a model of onboard system may include one coarse federate corresponding to a number of its secondary subsystems and a bunch of fine-grained federates corresponding to components of the most important subsystem simultaneously. Fine-grained federates are clearly less abstract than the coarse one because they correspond to smaller elements of the logical structure. Only their aggregate may form the new logical subsystem that can interact to the other subsystems on equal footing. Thus, federates of the aggregate depend on each other. They are logically linked.

During the simulation federate aggregation can be distinguished by the intensity of their interaction. As it is shown by the practice, members of a certain aggregate interact to the each other far more frequently than to any external federate. Federates are clustered into a number of aggregates and encapsulate the majority of communication traffic inside of them. Only a little part of traffic goes beyond and connects members of different aggregates.

CERTI has a centralized architecture and implements federate interaction of any kind using its CRC regardless to the model logical structure. In case the number of data exchanges is large enough, CRC is overloaded, becomes a bottleneck, and slows the simulation down. However, federates do not really care about the inner communication of aggregates they do not belong. Thus, the simulation traffic can be separated according to the model logical structure. The only thing we need is a dedicated middle-level CRC for each federate aggregation. From one hand, it will control the aggregated federates, encapsulate their inner traffic, and take some load of the real CRC. From the other hand, the real CRC will see it as a regular federate that runs in accordance to the common HLA execution rules and generates traffic flow corresponding to the bunch of aggregated federates. The middle-level CRC can be implemented as a new LRC frontend and does not result into significant increase of the RTI complexity.

There are several natural extensions of the described idea. First, federates can be clustered by a number of attributes differed from logical structure of the simulated system. For example, the non-uniform distribution of the federate communication intensity is a sufficient aggregation criterion. Second, the same trick can be used several times. In their turn, aggregated federates can be separated into a number of smaller groups, and form a new simulation control layer. Therefore, the described RTI architecture is referenced as a layered architecture.

To sum it up, introduction of the layered architecture results into a number of benefits. First of all, it solves the excessive CRC load problem and increases scalability of RTI. Indeed, the middle-level CRCs process the internal aggregate traffic independently and take a part of responsibilities from the real CRC. Each middle-level CRC can be executed by a separate host, thus, the RTI control is distributed automatically without any data replication or sophisticated coherence control algorithms.

Second, aggregation allows reducing of the synchronization losses. Internal interaction of aggregated federates goes through the middle-level CRC and does not take into account the most of external dependencies. Therefore, it is more efficient than the regular one. However, their external interaction is less efficient and includes two mediators, namely, the middle-level CRC and the regular one. If aggregation can be chosen *sufficiently* well, the synchronization losses can be reduced respectively.

Third, aggregation allows accurate RTI attuning. Federates can be clustered according to a set of services they use. Unused RTI components can be safely removed from the middle-level CRC and its complexity will reduce respectively. The remaining services can be also attuned to the requirements of the joined federates. For example, each middle-level CRC may implement its own time management algorithm that is effective for the aggregated federates independently of other RTI components. In case of HILS, the main CRC should always use conservative time management algorithms as a core. However, middle-level CRC component may use optimistic algorithms in case it is more efficient.

Finally, aggregation is a way to increase the efficiency of interactions within a single node. Centralized architecture does not take into account the relative position of federates. Even in case they are running on a single node, every data exchange goes though the RTIG. Thereby, RTI uses two network communications to transmit data between two processes on a single host. This wasteful data handling results into a significant performance decrease. Aggregation of all federates on the node actually allows their direct interaction without any network involvement. Thereby, the concept of node aggregation brings some advantages of the decentralized peer-to-peer architecture without any changes in a current CERTI logic.

The weak side of the layered RT architecture is indeterminism of RTI structure and its dependence on the executed model structure. It also requires some automated static and dynamic model analysis tools responsible for the criterion selection and the corresponding suboptimal federate segregation. However, the benefits it may give seem to worth efforts, and this approach appears to be rather prospective.

B. *Thread-based LRC*

CERTI LRC consists of libRTI library and RTIA process connected by UNIX pipe. Although libRTI is linked to federate process and provides HLA API, the library does not really implement RTI logic. The library just redirects the incoming method calls to the connected RTIA. In more details, every time federate calls RTI service, libRTI sends to RTIA a message with the associated method identifier and a set of supplied arguments. RTIA handles these queries and replies back with results. Thereby, libRTI can be considered as LRC frontend whereas RTIA corresponds to LRC backend. Due to this modular LRC structure, changes of HLA API will not affect the LRC backend directly and the corresponding RTI changes will require a minimum of effort. Currently CERTI uses this flexibility to maintain both DMSO 1.3 and IEEE 1516 2000 HLA versions.

Another advantage of the two-component LRC against the single-component one is the increase of simulation security and reliability. Both libRTI and RTIA run in their own context and verify every incoming message. Therefore, there is no way the federate can read or modify any internal RTI data, except the calls of the HLA API. For the same reasons, failure of the joined federate never leads to a failure of the whole RTI.

However, the flexibility of the composed LRC decreases an overall RTI performance. Every time the federate calls RTI method, at least two internal messages are generated, and this number may increase in case of RTI callback requests. Transmission through the pipe requires message parameters to be serialized during the send and deserialized on its reception. These data format conversions inevitably result into undesirable memory copying and additional CPU load.

There are several ways to avoid the unnecessary communication overhead. The first one is to replace the pipe with a set of queues in a shared-memory. This approach does not require any data reformatting during the transmission and, therefore, decreases CPU load. However, it requires a support from operating system and a number of corresponding system calls. A more performance emphasized approach is to include RTIA right into the federation process as the additional thread. Thereby, libRTI and RTIA will automatically share the same address space. Thread-level data exchange can be more effective than the process-level one. Moreover, modern multi-core CPUs are able to provide multi-threaded execution with the additional performance gain. Unfortunately, integration of libRTI and RTIA in a single process breaks the simulation security and reliability. Still, this solution fits the purposes of HILS development and seems to be a preferable option.

Although it is not obvious at the first glance, both considered approaches require serious modification of RTIA process. RTIA is a single-thread process, and it has to wait both RTIG and libRTI messages simultaneously. There are two well-known paradigms of its implementation, namely, polled and related waiting. During the polled waiting, process just looks for incoming messages in a cycle. This requires additional CPU time. During the related waiting process asks the system to notify it when the message comes, and suspends until reception of this notification. This approach is more complicated and its efficiency is inversely proportioned to the frequency of incoming messages. RTIA receives message rare enough, thus CERTI uses related waiting. However, there is no standard way to implement related waiting of the socket and either shared memory or thread using the simple single-threaded process. Thereby, any of them requires RTIA to use polling or implement related waiting using multiple threads.

V. Conclusion

According to the conducted performance benchmarking, out-of-the-box CERTI RTI lags far behind the HILS STAND. Although CERTI cannot be used as a runtime for the same range of simulations, the absolute values of its latency and throughput are acceptable for the average HILS model. Therefore, the performance gap between two systems is not a critical one and can be further reduced after implementation of the stated proposals.

Acknowledgment

Special thanks to Dmitry Volkanov for reading the paper and giving many helpful remarks.

References

[1] V. V. Balashov, A. G. Bakhmurov, M. V. Chistolinov, R. L. Smeliansky, D. Yu. Volkanov, N. V. Youshchenko, “A hardware-in-the-loop simulation environment for real-time systems development and architecture evaluation,” International Journal of Critical Computer-Based Systems (IJCCBS), vol. 1 - issue 1/2/3, 2010.

[2] A. Bakhmurov, A. Kapitonova, R. Smeliansky, “DYANA: An Environment for Embedded System Design and Analysis,” in Proceedings of 5-th International Conference TACAS'99, Amsterdam, The Netherlands, March 22-28, 1999. pp.390-404.

[3] R.L. Smeliansky, Yu. V. Bakalov, “A Language for Specifying Distributed Programm Behavior,” Proceedings of the VII. International Workshop on Parallel Processing by Cellular Automata and Arrays, Parcella ’96, Berlin, 1996, pp.85-92.

[4] R.E. Nance, “A history of discrete event simulation programming languages,” Blacksburg, USA, 1993.

[5] M. Adelantado, P. Siron, and Chaudron J.B., "Towards an HLA Run-time Infrastructure with Hard Real-time Capabilities," in Proceedings of International Simulation Multi-Conference, Ottava, Canada, 2010.

[6] Chemeritskiy E.V., Savenkov K.O. Towards a real-time simulation environment on the edge of current trends // In Proceedings of the 5-th Spring/Summer Young Researchers' Colloquium on Software Engeneering, SYRCoSE-2011, Yekaterinburg, Russia, may 12-13 2011, pp. 128-133.

[7] R.G. Sargent, "Requirements of a Modeling Paradigm,", Winter Simulation Conference WSC’92, Arlington, USA, 1992, pp. 780- 782.

[8] Richard D. Fujimoto, “Parallel and Distributed simulation systems,” 2000.

[9] IEEE Std 1516-2010, “IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification,” 2010.

[10] E. Noulard, J.Y. Rousselot, and P. Siron, “CERTI, an Open Source RTI, why and how,” Spring Simulation Interoperability Workshop, San Diego, USA, 2009.

[11] P. Bieber, D. Raujol, and P. Siron, “Security Architecture for Federated Cooperative Information Systems,” Annual Computer Security Applications Conference, New Orleans, USA, 2000.

[12] M. Adelantado, J.L. Bussenot, J.Y. Rousselot, P. Siron, M. Betoule, “HP-CERTI: Towards a high Performance, high Availability Open Source RTI for Composable Simulations,” Fall simulation interoperability workshop, Orlando, USA, 2004.

[13] B. d’Ausbourg, P. Siron, and E. Noulard, “Running Real Time Distributed Simulations under Linux and CERTI,” European Simulation Interoperability Workshop, Edimburgh, Scotland, 2008.

[14] L. Malinga and WH. Le Roux, “HLA RTI Performance Evaluation,” European Simulation Interoperability Workshop, Istanbul, Turkey, 2009, pp. 1-6.

[15] M. Karlsson, P. Karlsson., “An In-Depth Look at RTI Process Models,” in Proceedings of 2003 Spring Simulation Interoperability Workshop. 03S-SIW-055, 2003.