Mixed-criticality management of Networked Real-Time Systems with ARTEMIS Simulator

Olivier CROS
ECE Paris - LACSC
37 quai de Grenelle, 75015 Paris
Email: cros@ece.fr

Laurent GEORGE
Université Paris-Est - LIGM / ESIEE
Bat Copernic - 5, bd Descartes
77454 Champs sur Marne, France
Email: lgeorge@ieee.org

***Abstract*—Nowadays, providing guarantees of performances and reliability in real-time systems implies to have simulation tools in order to test and emulate the systems. The real-time network infrastructures are not an exception to this rule, and needs their own simulators too. Our goal here is to present a new network simulator, ARTEMIS, which is designed to integrate mixed-criticality management in real-time networked systems. Our point here is to show simulation results of ARTEMIS, especially in mixed-criticality context, and to present the main different modules of this software.**

I. Introduction

A. About real-time simulators

Industrial systems with high constraints of mobility like in defense, public transports or automotive systems rely on real-time network architectures to exchange real-time information. Most of new functionalities in those systems are becoming electronically managed (brake by wire, steer by wire, ...) and interconnected. Organizing and prioritizing information among the different functions makes the scheduling need more important. Furthermore, these systems are becoming more and more complex, as we add them new functionalities with safety, security and real-time constraints. This represents a drastic increase in complexity, having impact in terms of costs and physical weight hence energy consumption. In order to satisfy these constraints, network infrastructures should be precisely designed and evaluated. One need is to be able to analyze the performances of these network infrastructures without paying the cost of designing real workbench, hence the need for real-time network simulators.

B. What is ARTEMIS ?

In the context of uniprocessor and multiprocessor systems, several simulators already exist such as SimSo [1]. In real-time networked systems, we have several network infrastructures like CAN [2] (automotive) or AFDX [3] (avionics). In order to answer to the needs of these infrastructures (in terms of performances, for example), several network simulators can be used. We can cite OPNET or OMNeT++ [4] tools, or even NS [5], but these tools do not answer to our needs.

Our goal is to introduce mixed-criticality (MC) management inside network simulation. Given that MC is not integrated in currently existing simulators, we decided to design Another Real-Time Engine for Message-Issued Simulation (ARTEMIS).

ARTEMIS is a real-time network simulator, based on four main development guidelines (see Figure 1). As real-time scheduling is a research domain implying mathematical, physical and industrial approaches not necessarily related to software development, the point was to build a network simulation tool easy to install and to use in order to be also used by non-developers. ARTEMIS is a web-oriented tool, which makes it totally independent from the underlying operating system.

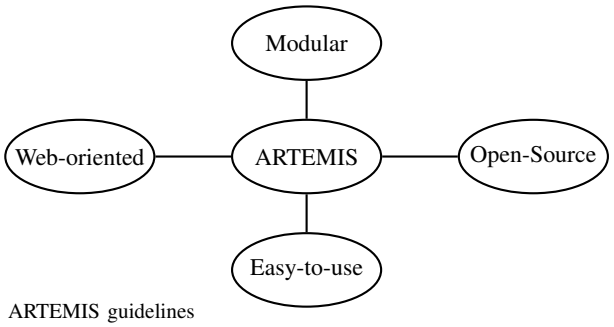

Fig. 1. ARTEMIS guidelines

ARTEMIS is a tool firstly designed for educational and research purposes : its main objective is to provide simulation results for scheduling models in a network context, as well as to design and simulate different kinds of topologies and infrastructures. In order to focus on MC integration problems in network infrastructures, ARTEMIS can model specific interconnected architectures like Network-on-Chip (NoC) or AFDX. But the point is not to make a real simulation of specific materials corresponding to a physical implementation, but to design a mathematical simulator in order to integrate real-time computation models inside a network design tool.

C. What's new ?

The first version of ARTEMIS was designed and finalized in may 2014. The main goal, at that time, was to design a functional network simulator, able to perform performance analysis for given topologies. Since, new modules have been integrated in the kernel and the graphical interface has been updated in order to improve its ergonomy. Among all, the main module added to ARTEMIS is the MC management system, which allows us to simulate MC scheduling scenarios in a network context. The point of this paper is to show the implementations brought by these new modules, and to focus on how the MC model has been integrated in ARTEMIS.

In section II, we briefly remind the architecture of ARTEMIS : its main modules and development axes. In section III, we focus on the implementation of MC modules. In section IV, we show several simulation scenarios obtained with the MC management module. Then, we present in section V the current work and the incoming modules. Finally, we conclude in section VI.

II. ARTEMIS DEVELOPMENT

A. Three-Element Modelization

ARTEMIS main structure has been designed to easily modelize a network topology. Each topology is represented by a set of 3 types of elements:

- **Nodes**: they are the end points and the transit points (switches) of the network. Each node represents a physical network interface, having its own network address (a unique identifier of the node in the network).
- **Links**: physical and electronical connection between two nodes. Basically, each link is just a set of 2 connected nodes. As we are mainly focusing on oriented networks, one node is considered as the input, and the other as the output, but this configuration can be managed to make a full duplex link.
- **Flows**: It is the information structure we want to send through the topology. Each flow is caracterized by a set of parameters (Worst-Case Transmission Time (WCTT), Period, Offset, Criticality levels, ...). We consider that each flow sends frames periodically, and that a frame is a set of bytes of a given size.

All the scheduling models and structures are based on this representation. It allows us to propose a generic approach in the conception of a network infrastructure, which can provide results that can be adapted to most of concrete situations. Using this genericity, we can then personalize and parameterize each type of element by adding different specifications (precising the type of nodes, flows format, ...) without any loss or changes in the simulator structure. That is why the modelization by the three different elements allows us to create an abstraction layer in ARTEMIS, between the modelization part and the simulation part.

As shown in [6], ARTEMIS kernel is splitted in two parts: the modeler and the simulator. The modeler represents and builds the topology respecting the previous constraints : modelizing the nodes, links, flows and building a static topology. The simulator dynamically emulates flow scheduling in the network: it makes the system evolving with a discrete time-based simulation, simulating the transmission of flows through the topology.

ARTEMIS consists in a software module built to simulate the transmission of a given set of flows inside a network topology. It consists in a Java kernel, a set of module basing their communication with the kernel on an XML file layer [6]. The first main point of unifying different services results in building a totally independent graphical interface, linked to the kernel by this XML layer.

B. Utilization

Creating and simulating a scheduling scenario with ARTEMIS relies on the definition of the 3 previously defined basic elements: nodes, links and flows. That is why we based the user experience, through the Graphical User Interface (GUI), on the sequential building of these elements in the network. In order to keep it easy to use, building a scenario is splitted in successive simple steps.

First, we define the network topology. It consists in the creation of the nodes and the links. We define each node of the network as a pair: its name (associated with an auto-generated network address), and its internal scheduling policy. By default, we define only basic scheduling policy for the nodes (FIFO, Fixed Priority scheduling), but the modular structure of ARTEMIS allows us to integrate new ones. Finally, we define the different links between the nodes, with possibly a specified bandwith for each.

In the second step, we define flows with their properties. A flow m in ARTEMIS is characterized as a 4-tuple $\{\mathcal{P}_m, \vec{C_m}, T_m, O_m\}$:

- Path P_m: the path of nodes followed by the frames of the flow in the network. It is statically-defined by the user.
- WCTT: the maximum time needed for a node to analyze and route any frame of the flow. In non-critical mode, we have only one WCTT per flow, denoted as C_m. In MC mode, we have one WCTT per criticality level i, C_m^i is then the WCTT associated to the criticality level i. In order to generalize this modelization, we denote the set of WCTT for each flow as $\vec{C_m}$, with the size of the set equal to the number of different criticality levels in the network.
- Period T_m: the inter-arrival time between two frames of flow m.
- Offset: a possible offset configuration between a reference time origin of the initialization of the modelisation tool and the emission of the first frame of the flow. If not defined, flows transmission is supposed to be non-concrete (the first release time of the flow can be chosen arbitrarily).

Basically, each flow path is supposed to be static and with a specified WCTT for each criticality level. The network is defined by default without MC managament activation, so there is only one WCTT per flow. MC is an option which has to be activated (see III). By doing this way, realizing a scheduling scenario in a simple topology does not need particular configuration. Finally, we can adjust the network simulation properties: total simulation time, electronical latency in each visited node, the maximum number of criticality levels, and the parameters needed to configure automatic task generation (the number of generated tasks, the final network load and their particular WCTT). Once these parameters has been defined, we can launch the simulation.

III. Mixed-criticality management in ARTEMIS

A. Network model

The first goal of ARTEMIS kernel is to simulate MC management in networks based on the hypothesis that the network criticality level is an information updated by a centralize entity in the network: one central point (one node) is responsible of updating the criticality level in the network. Each other nodes store a local copy of the current network criticality level and receive updates from the central point with a real-time reliable multicast. We consider that changing the criticality level in the network takes a bounded time (the maximum time needed to update this information on all the nodes of network). To preserve the consistancy of the criticality information stored in all nodes, a reliable real-time multicast can be used [7]. With this reliable real-time multicast we can guarantee that all nodes have the same criticality level at any time, provided that the criticality switch in all nodes occurs after a bounded time (the maximum needed to receive the information in all nodes). We therefore need to characterize the worst case end-to-end delay send and receive a multicast in all destination nodes.

Our point is to simulate a set of interconnected nodes, and to apply them a MC policy management in order to focus on the scheduling results it provides.

We consider that a network $\mathcal{N}$ is a set of nodes, links and flows. The global criticality level of the network is noted as Γ. This criticality level can take several values $\Gamma_1, \Gamma_2, \dots \Gamma_{max}$. Furthermore, we assume that the WCTT is increasing as a function of the criticality level. This corresponds to the case where the payload of flow increases as a function of the criticality level. For example, for a flow corresponding to video transmission, this corresponds to increase the quality of the video transmission. More generally, this corresponds to the case where the payload of flows increases as a function of the criticality level, due to more complex or more precise computations.

$$\forall (i,j) \in [1;\Gamma_{max}], \Gamma_i < \Gamma_j \implies \forall m \in \mathcal{N}, C_m^i < C_m^j$$

This hypothesis implies that each criticality level is defined hierarchically compared to the others. All criticality levels builds a global hierarchy of importance between the lowest (non-critical level) and the highest (with the highest WCTT).

Furthermore, we consider that for given flow, the WCTT in each node (particularly, the switches) is identical. This corresponds to case where the speed is identical for all links (100 Mo/s, 1 Go/s, ...).

B. Mixed-criticality change

In order to implement MC in ARTEMIS, and to simulate MC management in a network topology, we developed a module to integrate MC in its simulation models. This module is based on several hypotheses for our development.

First, all frames in the network are non-preemptive. We consider, in a network context, that we cannot interrupt a frame once transmitted to send another one and then resume the transmission of it. Hence, when a criticality change occurs, a specified delay (called the non-preemptive delay [8]) may be neeed to complete the transmission of the current flow before starting the transmission of critical flows.

Next, we consider that the criticality information is a stored variable which can be modified in bounded time in all nodes (with a real-time reliable multicast updating the criticality level after the maximum time needed to receive it in all nodes of the network, as defined in [7]). Given that, in our simulation context, the criticality changes are statically-defined by the user, we can consider that they occur precisely when we specify it (contrary to real implementations, where a criticality switch can be triggered automatically from information of the physical environment). It implies that, when a criticality change is defined by the user at a given time, it occurs in constant time.

Last, we place the context inside Hard-MC management. It means that, with a given criticality level Γ_i, only flows corresponding to this specified level are transmitted. In other words, we do not allow the simulator to send non-critical flows during high critical modes. The priority in the design and simulations of ARTEMIS is to focus on how to assure critical flows transmission.

In ARTEMIS core, managing MC is just a problem of selecting the correct flow to send in the network. Each node is associated to a set of input and output buffers which can contain frames. Applying MC management just consists in filtering, in this buffers, which frames are considered as critical or not, given the current criticality level. Then, picking and managing frames in a node in ARTEMIS corresponds to the following algorithm :

```
for time from 0 to END do
    while (current == ∅) do
        next = Pick(input_buffer, policy);
        if (next == ∅) then
            Break;
        end
        if (getWCTT(next, Γ) != -1) then
            current = next;
        end
        else
            Drop(next);
        end
    end
    if (current != ∅) then
        Analyze(current);
    end
end
```

C. Managing mixed-criticality in ARTEMIS

Through the GUI, the user can first define several criticality levels. By Default, ARTEMIS proposes only one criticality mode, which corresponds to single criticality level. Through the interface, a user can define new criticality levels (one at a time), by defining: the name of the level and its abbreviation code (for example, C for critical, SC for safety critical, etc...).

After defining the different criticality levels, the user defines for each flow its WCTT vector through a second part of

ID	Path	Period	Offset	WCTT Non critical	WCTT Critical	Edit	Add	Delete
2	ES2,S2,S1,OUT	50	0	6	10	Edit	-	Delete
3	ES3,S3,S1,OUT	20	0	2	-1	Edit	-	Delete
4	ES4,S3,S1,OUT	40	0	4	-1	Edit	-	Delete
6	ES1,S2,S1,OUT	30	0	10	20	Edit	-	Delete
-						-	Add	-

Fig. 2. Messages table

the GUI (see Figure 2). It means that we must define for each flow a WCTT for each criticality level of the system (critical: 2 and non-critical: 1 in the example). By Default, when a flow is not considered as part of the specified criticality level, its WCTT for this level is set to -1.

Time	Level	Add
40	2	Delete
70	1	Delete
	Non critical	Add

Fig. 3. Criticality change table table

After defining these levels, the user can define the time instants when each new criticality level happens, after the completion of the reliable multicast sent by the control node (see the table on Figure 3). At a given time, the user can define a criticality-change, which will make the scheduler to swap to the new defined criticality level at this time during the simulation. These criticality changes are defined with the destination criticality level, at the time at which they occur.

D. Java implementation

ARTEMIS core is implemented in Java. The Java Virtual Machine (JVM) allows us to build an abstraction layer, which makes ARTEMIS independent from the underlying Operating System. The core structure makes the kernel independent from the GUI and other modules through an XML abstraction layer, conformed to the architecture showed in Fig. 4. It works with the following structure: the user sends orders and data which are stored into a database, the simulation launcher builds these data as a set of XML input files, the kernel emulates the network topology based on this data and provides results as XML files, which are parsed by different modules (performance evaluator, GUI, grapher, ...).

For MC management in ARTEMIS, we implement a dedicated module integrated in the kernel. We split the representation of a flow in the CoreManager in two different parts:

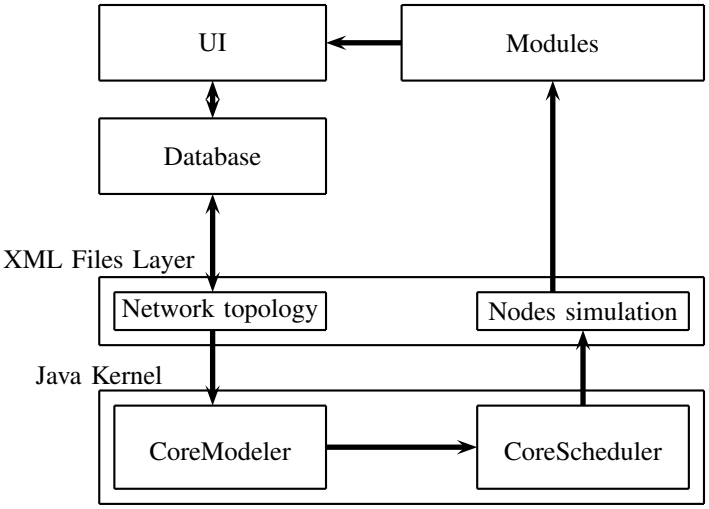

Fig. 4. Modules architecture for ARTEMIS

one classical flow scheduling, and one MC flow scheduling. The point of operating this split was to keep a very simple scheduling mode in the kernel, made for classical scheduling, without having to integrate MC models. We define for this a simple flow class model named NetworkFlow. In a second approach, we integrate a task model called MCFlow to represent a flow with several criticality levels. That gives us two different models for a flow m :

- A single criticality model, where $m = \{\mathcal{P}_m, \vec{C_m}, T_m, O_m\}$ and $\vec{C_m} = \{C_m^1\}$
- A MC-model, where $m = \{\mathcal{P}_m, \vec{C_m}, T_m, O_m\}$ and $\vec{C_m} = \{C_m^1, C_m^2, ...C_m^i\}$ for a network of i different criticality levels.

IV. Simulation of mixed-criticality scenarios

In order to simulate a MC-managing scenario with ARTEMIS, we selected three different scheduling scenarios, based on the same topology (see Figure 5). The point was to compare the different effects on flows transmission of criticality switching in the topology. We specified for all this scenarios a common set of flows : m_1, m_2, m_3, m_4.

Each flow m_i in the network results in the transmission of periodic frames m_i. We propose to focus on the MC managing in ARTEMIS through 3 different scheduling simulation scenarios inside the described topology.

For all scenarios, we define a set of global parameters, needed to set up the network :

Time simulation	200 ms
Electronical latency	0 ms
Scheduling	FIFO

1) Scenario 1: The first scenario is a single criticality scenario. It allows us to define the primary properties of flows : path, period, non-critical WCTT and offset. We specify a simple network of 4 flows $m_1, m_2, ..., m_4$. A real execution context would rather consist in 40 or 50 flows minimum, but

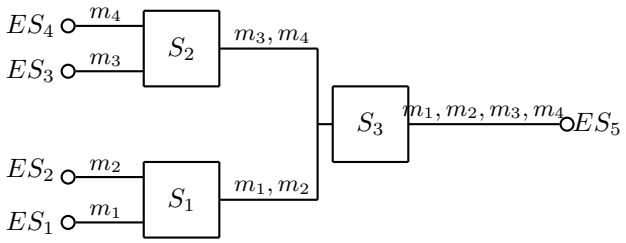

Fig. 5. ARTEMIS simulation topology

we limited here to a very low number of flows for the sake of proof of concept of the result graphs.

Flow	m_1	m_2	m_3	m_4
O_i	0	0	0	10
C_i^1(ms)	60	60	60	30
T_i (ms)	120	150	400	100

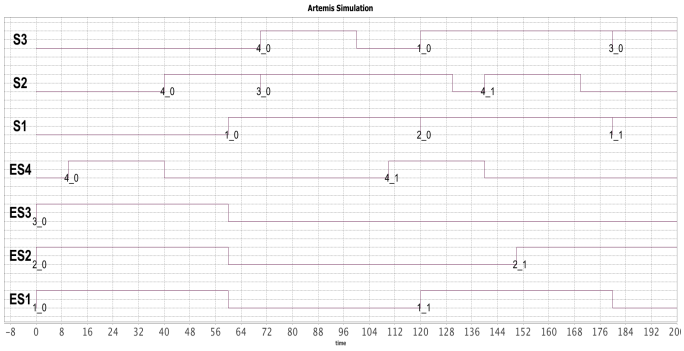

Fig. 6. First simulation scenario

We obtain a simple scheduling scenario for the 4 given flows (see Figure 6). We set by default the electronical latency between two nodes is equal to 0 ms. All flows are scheduled correctly, according to their parameters. In order to focus on the effects of switching criticality, we focus on nodes S_1, S_2, S_3.

2) Scenario 2: In this second scenario, the point is to keep the same previous parameters, but adding a second criticality level to flows m_1 and m_2. We consider in ARTEMIS that the WCTT of a packet which is not critical at a given criticality level is equal to -1. That is why we add the following parameters to the previous ones :

Flows	m_1	m_2	m_3	m_4
C_i^2(ms)	100	80	-1	-1

We add to the topology a criticality change at $t = 40ms$. We obtained the results shown in 7. We can see that, starting from $t = 40ms$, all single criticality frames are no more transmitted. The critical flows (m_1, m_2) have a corresponding increase in their WCTT starting from $t = 40ms$, which corresponds to the criticality change.

3) Scenario 3: This last scenario consists in representing a combination of the two previous scenarios : we consider the same topology, and the same parameters. We just add another criticality switch, from criticality level 2 to criticality level 1,

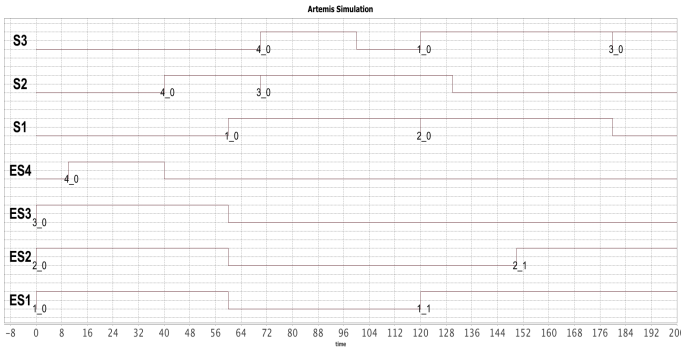

Fig. 7. MC simulation scenario

at $t = 150ms$. This way, we can obtain the complete behavior of the focused topology during a specified phase of criticality switching.

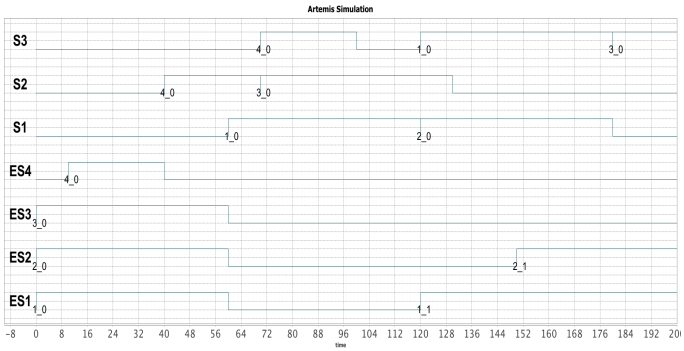

Fig. 8. Double change simulation scenario

As expected, we have a correct transmission of critical frames from flows m_1 and m_2 during the critical phase $40 - 80ms$. But, before and after this critical phase, we observe that all the non-critical packets are transmitted. This corresponds to a network criticality level equal to 1. We give the different changes of criticality level in figure 9.

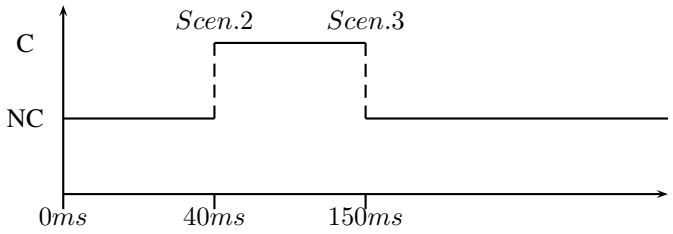

Fig. 9. Criticality switching scenarios

We can conclude that, through this simple example with two criticality levels, ARTEMIS correctly simulates the behavior of each node during the different non-critical and critical phases.

V. Current Work

Once integrating the theorical simulation models in ARTEMIS, our goal is now to link it with real architectures to take into account real traffic generation. We want ARTEMIS to communicate with the operating systems in order to send

and receive Ethernet frames. But the problem we are currently working on is that the JVM represents a software abstraction layer that cuts off real-time management. We had two possible solutions to solve this : by making the software real-time with Real Time Specification for Java (RTSJ) [9], or by establishing a real-time communication between ARTEMIS and the OS through a real-time layer.

We chose this second solution because of the hard-maintainability of RTSJ in embedded devices, and so we chose to work with Xenomai [10] in our approach. Xenomai is a real-time patch which is made as a module for the linux kernel. It is a free and open-source solution to make the linux kernel real-time oriented.

The socket server is the central point of the real-time integration of ARTEMIS we are currently working on. Our goal is to be able to send and receive Ethernet frames from real connected devices. But, in order to support real-time communications, Ethernet frames should be generated at the lowest layer of the ISO stack. The JVM does not allow this. That is why we need an intermediate element between the real network infrastructure and the ARTEMIS core.

Our solution is to establish a socket server in C++ which is an entry point to ARTEMIS. This socket server is responsible for the communication between Xenomai layers (the different OS devices) and ARTEMIS client. We obtain the architecture described in Fig 10.

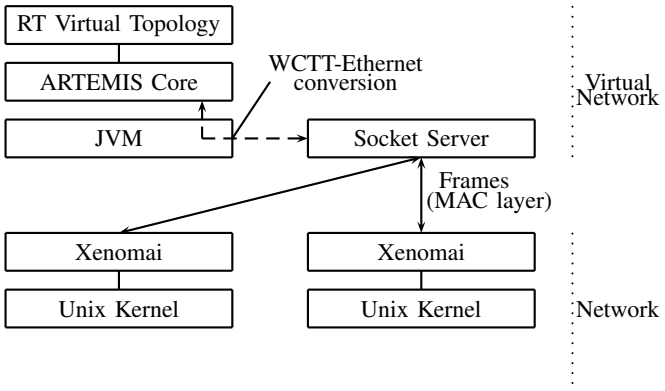

Fig. 10. Xenomai integration architecture

VI. Conclusion and perspectives

A. Conclusion

This new version of ARTEMIS now provides management of more complex scheduling scenarios and integrates new models of scheduling through MC management. We show that the scheduler part of the simulator integrates dynamic criticality changes inside the model. MC criticality level updates are managed by a central node that multicasts updates with a real-time reliable multicast. ARTEMIS allows us to provide reliable simulation results for these new simulation model.

MC integration in networks domains such as avionics, defense or smart vehicles. ARTEMIS is a tool to simulate topologies inside these industrial contexts.

B. Perspectives

Now that ARTEMIS virtual simulation is functional, our goal is to use currently developing modules to integrate this virtual simulation to real network infrastructures, and to be able to simulate complex real-time topologies integrating AFDX or CAN networks, in order to provide simulation results for such networks.

References

[1] M. Cheramy, P.-E. Hladik, and A.-M. Deplanche, "Simso: A simulation tool to evaluate real-time multiprocessor scheduling algorithms," in *WATERS'14*, 2014.

[2] K. Tindell and A. Burns, "Guaranteeing message latencies on Controller Area Network (CAN)," in *Proceedings of the 1st International CAN Conference*. Citeseer, 1994.

[3] H. Charara, J.-L. Scharbarg, J. Ermont, and C. Fraboul, "Method for bounding end-to-end delays on an AFDX network," in *Proc. the 18th Euromicro Conference on Real-Time Systems (ECRTS)*, Germany, July 2006, pp. 192–202.

[4] A. Varga, "The omnet++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference*.

[5] T. R. Henderson, M. Lacage, and G. F. Riley, "Network simulations with the ns-3 simulator," in *In Sigcomm, 2008*.

[6] O. Cros, F. Fauberteau, and X. L. Laurent George, "Simulating real-time and embedded networks scheduling scenarios with artemis," in *WATERS'14*, 2014.

[7] L. George and P. Minet, "A fifo worst case analysis for a hard real-time distributed problem with consistency constraints," in *Proceedings of the 17th International Conference on Distributed Computing Systems*, 1997.

[8] L. George, N. Rivierre, and M. Spuri, *Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling*, 1996.

[9] G. Bolella and J. Gosling, "The real-time specification for java," vol. 33, pp. 47–54, 2000.

[10] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Talierco, "Performance comparison of vxworks, linux, rtai and xenomai in a hard real-time application," in *Real-Time Conference, 2007*, 2007.