Proceedings of 2024 IEEE 12th International Conference on Intelligent Systems (IS)

Discrete-Event Co-Simulation Interface for Time-Triggered Organic Computing

Mario Qosja
Embedded Systems
University of Siegen
Siegen, Germany
mario.qosja@uni-siegen.de

Simon Meckel
Embedded Systems
University of Siegen
Siegen, Germany
simon.meckel@uni-siegen.de

Roman Obermaisser
Embedded Systems
University of Siegen
Siegen, Germany
roman.obermaisser@uni-siegen.de

***Abstract*—In response to the increasing complexity of distributed embedded systems, self-organizing systems have emerged. One such system is organic computing, which draws inspiration from biological organisms. It improves the adaptability and robustness of distributed embedded systems by allocating tasks to computing nodes on demand. However, this approach has drawbacks in terms of determinism and dependability. To address these limitations, time-triggered communication concepts have been incorporated to form a time-triggered organic computing architecture (TTOC). Herein, task executions and message communications are predefined by a table schedule based on a list scheduler. The schedule table, i.e., the order in which application tasks get executed, has been validated by a previously introduced TTOC simulator. The next step towards a real-world usage of TTOC is a discrete-event simulator that includes the simulation of actual time-triggered communication protocols such as TSN. OMNeT++ provides such discrete-event simulations. Both the TTOC simulator and OMNeT++ are discrete-event simulators. Consequently, this paper presents a discrete-event-based co-simulation interface that can control the execution of the TTOC simulator and OMNeT++.**

***Keywords*—Organic Computing; Time-Triggered; Co-simulation; Interface;**

I. Introduction

Distributed embedded computer systems are a significant part of our daily lives and play a major role in most industries. The automotive industry, medical devices, factory robots, smart devices (e.g., watches, phones), aerospace, and smart homes are some applications where embedded computing systems are employed. The demand for these applications is increasing, and it is expected that the functionality requirements will increase further in the future. As technology advances, embedded systems are becoming more and more complex. However, this complexity presents a challenge when it comes to managing and maintaining these systems [1], particularly when the environment around them is constantly changing. In order to ensure the reliable provision of services, embedded computing systems must be designed to handle complexity autonomously and adapt to changes in real-time. A system is defined to be autonomous if it exhibits self-x properties, such as self-organization, self-configuration, self-healing, or self-describing [2].

Organic Computing (OC) is a way of organizing distributed computing systems that brings flexibility and self-organization. It is inspired by the principles of biological systems and uses so-called Artificial DNA (ADNA) to build embedded systems and an Artificial Hormone System (AHS) to organize them. Just as DNA is stored in every cell and contains a complete description of the biological entity, ADNA encodes the organization and structure of the system. AHS serves as a middleware to make complex embedded systems more adaptable and robust by allowing computing nodes to exchange small messages, called artificial hormones, that help determine the suitability of task allocations, especially during start-up, node failures, and system service degradation.

However, the AHS lacks support for dependability, determinism, and composability, which is necessary for safety-critical systems. Distributed embedded computing systems utilizing time-triggered communication show specific characteristics. In these systems, control signals are generated at specific points in time on a synchronized global time base. The communication of messages is scheduled and predictable, ensuring resource adequacy and predictability. By combining self-x features with time-triggered concepts, the system becomes more reliable, deterministic, and dependable [3]. This is because the outcome of the system can be predicted with greater accuracy. These systems ensure resource adequacy and predictability through a priori scheduled tasks and messages. They help contain faults in the time domain and simplify certifiability by providing knowledge of the permitted temporal behavior of components.

Therefore, organic computing technologies of AHS and ADNA are combined with time-triggered concepts to create Time-Triggered Organic Computing (TTOC). In TTOC all operations (i.e., task executions, message communications) are predefined by the table schedules. Each schedule is generated dynamically at each node by a list scheduling algorithm that is designed specifically according to TTOC requirements. To validate the new concept, we have developed a simulator for the TTOC architecture [4]. In the current stage, the execution of tasks is simulated based on the generated schedule. Application tasks are placeholders, which means that they do not exchange application messages between them. We

979-8-3503-5098-2/24/$31.00 ©2024 IEEE

want to introduce application tasks that contain application messages to simulate time-triggered communication between the nodes. For that reason, we want to simulate a time-triggered network in a network simulator. OMNeT++ and its framework INET can provide the required tools to perform a network simulation. Time-Sensitive Networking (TSN) was selected from the currently supported networks due to the supported time-triggered features.

To further validate our approach, it is necessary to couple the two simulations. This paper presents a discrete-event co-simulation interface for simulating network communication in a distributed embedded system based on organic computing. The interface will serve as a coordinator for the simulations, reading the execution time of the next event and determining which simulation must be executed based on the earliest time. The following sections provide an overview of the current state of the art in organic computing and co-simulation interfaces in Section II. A concise overview of the TTOC simulator can be found in Section III. Section IV presents details about the network simulator and the selected communication network. Section V explains the co-simulation interface, and Section VI provides an evaluation of the newly designed interface. Finally, the paper closes with a conclusion and future work in Section VII.

II. Related Work

A. Organic Computing

Research for the self-x properties of autonomous computing, especially the self-organization systems, has been conducted for several years before [5]. One of these systems that has drawn attention is Organic Computing. It was established by the German National Science Foundation (Deutsche Forschungsgemeinschaft) to adapt self-organizing principles of biological systems with the objective of handling unexpected behaviors that may arise from complex distributed embedded computer systems [6].

The organic computing architecture is based on the observer/controller architecture to provide self-x properties to distributed embedded systems. A distributed self-organization OC based on the observer/controller architecture with the ability to control unexpected behaviors is proposed [7]. This OC system was tested on a traffic light controller [8]. This architecture is used in robotics, too. OSCAR [9] is an adapting robot that exhibits self-x properties to test walking behaviors in different environments. In addition, a middleware for ubiquitous computing has been developed, following the observer/controller design [10]. The middleware contains an automated planner that combines the functionality of self-x properties into a single component. Organic computing features are also implemented in embedded real-time systems. CAROS is a real-time operating system developed to provide a solid base for implementing self-x techniques [11].

In addition, new architectural concepts have emerged. Biological DNA concepts have been transferred to computer systems, thus introducing the artificial DNA. Similar to how genetic instructions are encoded in DNA and stored in each cell, the structure, as well as the organization of embedded systems, are encoded in a computer file [12]. ADNA files, too, are stored in every computing node of the system. Another OC technology that follows the same ideas is the Artificial Hormone System [13]. It is a real-time middleware that exhibits self-organizing properties by allocating application tasks to computing nodes based on suitability level. The suitability level is calculated by each node from exchanging hormones with all the other ones.

B. Co-Simulation Interfaces

The emerging challenges of complex engineered systems, which include physical and software aspects, have led to a need for co-simulation research [14]. In order to address these challenges, the Functional Mock-up Interface (FMI) [15] was developed. The FMI is a standardized interface that enables the construction of complex Cyber-Physical Systems (CPS) through the coupling of different simulators based on the master/slave architecture. It has gained usage in many fields of industry, including automotive publications [16], [17], robotics [18], CPS energy systems [19], and prototyping self-adaptive systems [20]. In most of the aforementioned scenarios, two simulators from different domains are involved. The simulators may be Discrete-Event (DE) or Continuous-Event (CT). DE co-simulation is employed for modelling CPS in [21], smart grid applications [22], and electric grid co-simulation with communication networks [23].

There is a research gap regarding distributed systems based on organic computing that offer determinism and predictability. Therefore, we combined organic computing with time-triggered concepts to build distributed embedded systems that exhibit both high reliability and flexibility as well as determinism and predictability. The simulator in [4] performs TTOC task execution based on calculated table schedules. Message communication is simulated in a network simulator. To couple both systems, a co-simulation interface is needed. Our approach is a DE-based co-simulation and does not utilize an FMI interface. Detailed explanations of the developed co-simulation interface can be found in the following sections.

III. Time-Triggered Organic Computing Simulator

Based on the existing simulators for the evaluation of the ADNA and AHS concepts [24], the TTOC simulator was designed to demonstrate the time-triggered organic computing concepts by simulating task dispatcher (order of task execution) and scheduling algorithm (schedule generation). This section provides a brief overview of how the simulator works.

A. ADNA & AHS

As addressed in Section I, organic computing systems based on ADNA can be constructed by decoding the system structure and its organization in a single file and storing it in each computing node [25]. All entities of the system (e.g., filters,

actuators, sensors) and their functionalities in terms of tasks and messages are represented as basic elements containing an *ID*, *Sourcelink* and *Destinationlink*. The *ID* distinguishes different basic elements, the *Sourcelink* denotes a reactive link that responds to incoming requests whereas *Destinationlink* is an active link for sending requests to other tasks.

In order to read the ADNA file and build the system during runtime (self-build), it is necessary to have the AHS middleware. AHS is capable of organizing the system (self-organization) by assigning application tasks to computing nodes based on their suitability levels (e.g., tasks from the PID controller will have higher suitability on computing nodes that perform arithmetic calculations better). Each node calculates its suitability level by executing the hormone loop and exchanging hormones with the other computing nodes. Furthermore, in the event of failures (e.g., tasks or nodes), the AHS reconfigures the system (self-reconfiguration) by allocating the failed tasks to other nodes to prevent the loss of functionality.

The three main hormone types are *Eager values*, *Suppressors*, and *Accelerators*. Eager values express the momentary suitability of a node to perform a particular task. An eager value is determined in the hormone loop by considering the initial local (i.e., node-specific) eager value, which is then decreased by suppressor values and increased by accelerator values, respectively. In the hormone loop, in the first phase, each node sends accelerator and suppressor values to the other nodes. In the second phase, the local eager values are calculated and sent to all nodes, and the third phase is reserved for (organ) accelerators. At the end of the task decision phase, each node assigns or passes a task based on the calculated suitability level.

B. Time-Triggered Architecture

Ensuring that the system behaves consistently is essential in distributed embedded systems. This is achieved by processing events on all nodes in a consistent order. A global time base is used to execute operations to ensure that the same order is followed by all nodes, thus establishing determinism. Being deterministic increases overall reliability and safety as the system is predictable. It extends the area where they can be used (e.g., safety-critical systems). The global time also helps to detect faulty situations. For example, in the case of a Babbling Idiot failure, where a computing node sends untimely messages, all nodes have predefined times when they are allowed to communicate. Communication outside of the specified time slots can be blocked by bus guardians. Any communication network that supports time-triggered features is suitable for TTOC (e.g., TSN, TTEthernet).

Figure 1 gives insight into the TTOC computing node. Each node contains a task dispatcher and a time-triggered schedule. The dispatcher reads the schedule and executes the task with the earliest slot. In TTOC, the schedule gets dynamically calculated at each node by a list scheduling algorithm. In the architecture, alongside the application tasks, there are the

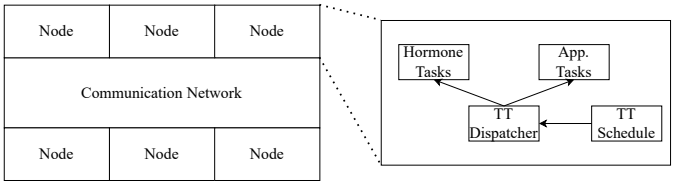

Fig. 1. Inside view of TTOC node

TTOC middleware tasks for the hormone message exchange, which is also time-triggered. More details about the task dispatcher and schedule are given in Section III-C.

C. TTOC Simulator

The TTOC simulator is built to test the time-triggered concepts using SystemC threads (SC_THREAD). SystemC is a modeling and simulation language written in C++ classes that provides an event-driven simulation (also referred to as discrete-event simulation). Discrete-event simulations provide sequential execution of events at particular instants in time, with each event representing a state change of the system. When switching between events, no change takes place in the system, so the simulation time progresses until the time of the next event. This is known as the next-event time progression.

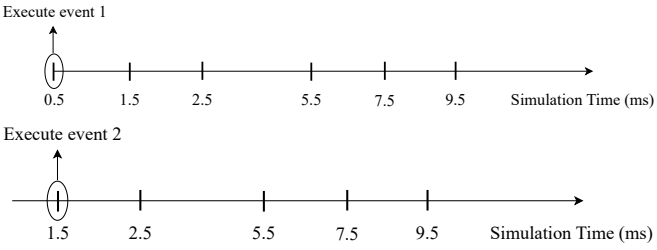

Fig. 2. Discrete-event simulation

Figure 2 describes a list of multiple events at different times. The first event is executed at $0.5\,\mathrm{ms}$, and then the simulation time jumps to the time of the second event at $1.5\,\mathrm{ms}$. The TTOC simulator starts by implementing AHS features, thus reading the ADNA file and constructing the computing node and the task's data structures. SC_THREAD simulates the functionality of the task dispatcher and the scheduling algorithm. The task dispatcher reads the schedule and executes the next event. An event is a hormone loop execution, schedule calculation, or application task. Recall that in a discrete event simulation, we simulate sequentially, and in the case of multiple threads, it is impossible to have concurrent execution by default. Parallelism is achieved by using a SystemC wait call that suspends the thread, thus allowing the execution of other threads. This is important to allow other nodes, executed in other threads, to receive hormone values, which are essential for accurate task allocation decisions. For this reason, Hormone Tasks (HT) are created

with each hormone loop phase representing one HT. Hormone tasks have fixed slots in the schedule (i.e., the hormone task period). When the AHS decides on the suitability of a task, it activates the list scheduling algorithm to calculate the temporal allocation of application tasks. It also handles the temporal and spatial allocation of messages.

Figure 3 shows the generation phase of the schedule for each computing node with application tasks denoted as AT and hormone tasks HT.

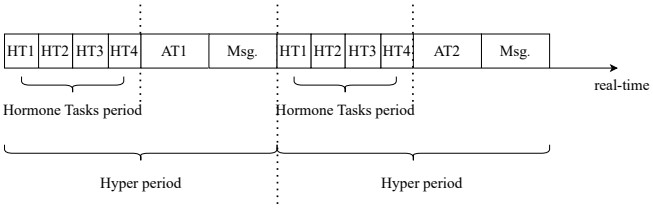

Fig. 3. Schedule generation phase

IV. Network Simulator

To simulate message communication between TTOC computing nodes, a network simulator that supports time-triggered features is required. OMNeT++ is a component-based C++ simulation framework that allows the construction of network simulators. One of these frameworks is INET, which includes TSN communication capabilities. In this paper, the network simulator is built using OMNeT++ 6.0.1 and the TSN is based on the INET 4.5 model.

A. OMNeT++

OMNeT++ is an object-oriented framework for discrete-event network simulation [26]. It consists of simple modules written in C++ that communicate via messages. Simple modules can be grouped to form more complex modules called compounds. A network itself is a compound module. OMNeT++ uses Network Description Files (NED) to represent simple and compound modules. The parameters of modules (simple and compound) are configured using a configuration file called the INI file.

Figure 4 gives an overview of OMNeT++'s modular architecture. The user interfaces available are Cmdenv and Qtenv. The ENVIR module is a library containing all the code shared by all the user interfaces, and SIM is the module containing the simulation kernel and class libraries. All network components like simple modules, channels, messages, compound modules, and the corresponding C++ objects can be found in the Model Component Library. A simulation can only run network models if all components are linked. The model to be simulated is located in the Executing Model module. ENVIR is the module that has full control over the simulation. It determines the model to be used and contains the simulation loop in which events are executed. Furthermore, it invokes the event scheduler and handles any errors that occur during the simulation. The simulation concept of OMNeT++

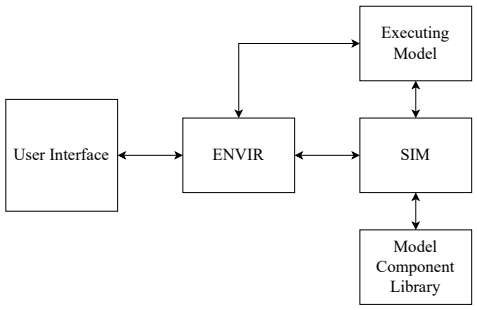

Fig. 4. OMNet++ simulations architecture [27]

is based on discrete-event simulation. As previously stated in Section III-C, a discrete-event system is defined as a system in which state changes (events) occur at specific points in time, and events take no time to occur. It is assumed that nothing of interest happens between two successive events, i.e., there is no change of state in the system between events. In OMNeT++, events are represented as messages and stored in a data structure called the Future Event Set (FES). The event loop is responsible for executing and processing all events from the FES. Causality is maintained by a timestamp, which ensures that no current event can have an effect on previous events.

B. INET

Any communication network that supports the time-triggered concepts is suitable for time-triggered organic computing. Time Sensitive Networking (TSN) is an IEEE standard (802.1) that transmits data in an Ethernet network in a time-sensitive manner. TSN features like time synchronization (IEEE 802.1AS), stream redundancy, cut-through switching, per-stream filtering and policing (IEEE 802.1Qci), scheduling and traffic shaping (IEEE 802.1Qbv), frame preemption, frame replication, and elimination (IEEE 802.1CB), automatic network configuration for failure protection, and gate scheduling are implemented in OMNeT++ by the INET framework [28]. INET provides three network devices that implement TSN-specific parameters.

- **TSN Device** models a computing node capable of supporting TSN features.
- **TSN Switch** implements TSN features upon an Ethernet switch.
- **TSN Clock** handles time synchronization.

V. Co-simulation Interface

To simulate message communication between computing nodes in a distributed system based on time-triggered organic computing, we utilize a TSN network running in OMNeT++ that operates based on the generated schedule. The execution of the TTOC simulator and the network in OMNeT++ must be coordinated to obtain correct behavior and output from both processes. Bringing together two different simulations into one single system is realized by a co-simulation interface that

controls the execution of the TTOC simulator and OMNeT++, see Figure 5.

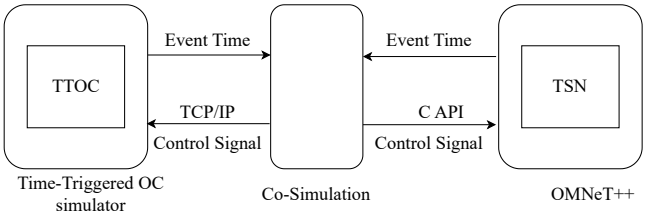

Fig. 5. Co-simulation framework architecture

Since both simulators are discrete-event, the co-simulation interface will also be discrete-event-based. The connection with the TTOC simulator is realized via Transmission Control Protocol/Internet Protocol (TCP/IP) and with OMNeT++ via C Application Programming Interfaces (API). In this paper, we focus on the control plane of the interface.

Figure 7 depicts the co-simulation logic of the newly designed interface. Once the co-simulation has commenced, the controller acquires the time of the subsequent events from the TTOC simulator and OMNeT++. It then compares the

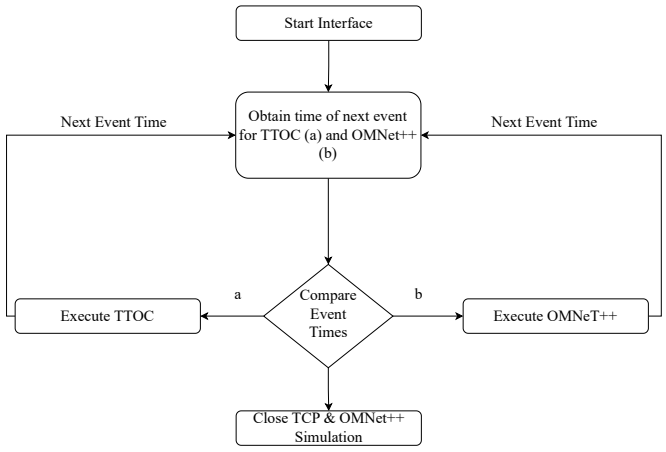

Fig. 6. Co-simulation logic

received time values and sends a trigger command to execute the event with the earliest time. If the TTOC simulator has the earliest time, a trigger signal will be sent and a new time will arrive. The same procedure happens in case OMNeT++ has the earliest time. This continues until there are events present, otherwise, the controller closes the TCP/IP socket and OMNeT++ simulation. In addition to the co-simulation architecture, this section also describes the modifications made to both simulators to adapt them to the time-triggered design.

A. TTOC Simulator Modifications

As illustrated in Figure 5, the TTOC simulator is linked to the co-simulation controller via the TCP/IP network protocol, where it provides the time of the next event, and the co-simulation controller sends the control signal. The control signal is a trigger command from the controller to the simulation to initiate the execution of the event. Consequently, the computation logic of the TTOC simulator must be modified. As stated in Section III-C, SystemC threads are employed

Algorithm 1 TTOC SystemC procedure

```
Ensure: time = 0
Ensure: connected = false
Ensure: execute = true
  Start TCP connection
  while true do
    if execute then
      if connected then
        sc_start(time,SC_MS)
        Obtain next event time
        Send time
        Wait co-simulation response
      end if
    else
      Wait co-simulation response
    end if
  end while
  Close TCP connection
```

in the TTOC simulator. In SystemC, the method sc_start() is responsible for the simulation phase. It comprises three distinct phases. The first is the elaboration, during which the execution of all the statements preceding the sc_start() is carried out. This is important to construct key data structures required for the simulation. The second phase is the execution of the threads until the final events in the scheduler. The last one is the post-processing, during which the data structures are destroyed and the simulation is finished.

The manner in which sc_start() is invoked determines the subsequent course of the simulation. If it is called once and without arguments (usually simulation time), the scheduler runs until the end of the simulation time. With the integration of the TCP/IP protocol and triggered execution of events, the simulation logic changes. Algorithm 1 outlines the newly developed procedure. First, the TCP/IP connection is established and the procedure commences. The time of the next event is retrieved from the SystemC scheduler and sent to the co-simulation interface. If the response of the co-simulation is a Boolean value of 1, execution is triggered via sc_start(), and the time that was sent is passed as an argument. If the response is a value of 0, the thread halts until the interface sends a value of 1. If the TCP/IP connection is not established or if there are no further events to be processed, the socket will be closed, and the procedure will conclude.

B. OMNeT++ Modifications

Similar modifications to the TTOC simulator should be performed also for OMNeT++. The difference is that instead of TCP/IP protocol, C API calls and functions have been used. Recall Section IV-A, ENVIR is the module that

controls simulation. It needs a runnable ENVIR which can be instantiated as a user interface. For this reason, we have constructed a user interface based on Cmdenv and new functions that are added to OMNeT++ source files. In the previous approach, after the ENVIR had been built, the system automatically loaded the network description files and the configuration file and executed the events of the FES. This process is modified and divided into multiple functions representing different stages of the simulation. These stages include Setup_Omnet, Build_Network, and Execute_Simulation. Setup_Omnet creates a user interface that loads the configuration file network description and constructs an active simulation. Build_Network handles the creation of C objects of the simulation (i.e., simple, compound modules), and Execute_Simulation triggers the execution of events. In the new design, only one event is executed by the function call, like in TTOC where events were executed only after the TCP/IP signal. In addition, to obtain the time of the event in the OMNeT++ scheduler, a new function is created.

VI. Evaluation of Co-simulation Interface

In the evaluation part, the newly designed co-simulation controller, the combination of self-x properties with time-triggered concepts in the TTOC simulator, and message communication in the network simulator are tested. The physical and application models are described in ADNA files. In OMNeT++, the network description file is written based on the same physical models as in the TTOC simulator. The selected communication network is TSN, and the results of each component of the system (controller, TTOC simulator, OMNeT++) are logged. The physical model described in the example ADNA files consists of six computing nodes (end systems) and three switches.

A. Co-simulation Interface

In the co-simulation interface, we evaluate the co-simulation functionality by testing if the interface obtains the execution times of both simulations and if it correctly sends the trigger signal for executing the events. Algorithm 2 specifies the procedure that is built for testing the co-simulation interface.

The first steps include declaring two boolean values (TTOCtimeflag, OMNeTtimeflag) as true, enabling the TCP connection, and activating OMNeT++ simulation. After obtaining the next event time for both simulators, we compare the results to determine which event to execute. The event with the earliest time must be executed. In the case of TTOC execution, the OMNeTtimeflag becomes false as the value previously obtained remains the same, and a trigger signal is sent via TCP with the value 1. If OMNeT++ time is earliest, TTOCtimeflag becomes false, and the OMNeT++ event is executed. In addition, a TCP trigger with value 0 is sent to notify TTOC that OMNeT++ simulation is running. An automated cross-checking of the co-simulation, the TTOC simulator, and OMNeT++ log files is performed. In our analyses, we run various scenarios of ADNA files (i.e., different simulated physical and application models). The outcomes show that the co-simulation interface obtains correct event times and that both simulations perform the event executions from the trigger command of the co-simulation.

Algorithm 2 Co-simulation procedure

```
Ensure: TTOCtimeflag = true
Ensure: OMNeTtimeflag = true
  Start TCP connection
  Start OMNeT++ simulation
  while true do
    if TTOCtimeflag then
      Obtain TTOC event time
    end if
    if OMNeTtimeflag then
      Obtain OMNeT event time
    end if
    Compare event times (TTOC time and OMNeT time)
    if TTOC then
      TTOCtimeflag = true
      OMNeTtimeflag = false
      Execute TTOC = 1
    end if
    if OMNeT then
      TTOCtimeflag = false
      OMNeTtimeflag = true
      Execute OMNeT
      Execute TTOC = 0
    end if
  end while
  Close TCP connection
  Close OMNeT++ simulation
```

B. TTOC Simulator

The modifications made to the TTOC simulator, as outlined in Section V-A, must not affect the self-x properties and the execution of tasks based on the generated table schedules. The analysis of the log file from multiple scenarios serves to validate the new design. The simulator creates all the data structures, establishes the TCP/IP connection, transmits the time of the next event, and initiates task executions in response to the signal received from the co-simulation controller. Furthermore, the self-organization and self-configuration of the AHS are also tested in case of computing node failures. If a node fails, its tasks must be transferred to the remaining ones. This is evaluated by creating timed events that represent node failures. After a node fails, the system gets reconfigured, the tasks of the failed node will be assigned to the other nodes, and new schedules will be created.

C. OMNeT++

In Section V-B, we discussed the construction of a new user interface based on Cmdenv. It allows the OMNeT++ and the co-simulation to be integrated into a single executable. The

communication network selected for the simulation is TSN, which is included in the INET framework. The number of TSN nodes and switches provided by the ADNA file. For the configuration of the network, examples provided by INET were employed. It is of paramount importance to be able to interrupt the execution loop of the network, thus enabling us to regulate the execution of events. Upon examination of the log file, it becomes evident that the modifications were successfully implemented. The simulator is responsible for constructing the network, while the execution of events occurs when the controller calls the Execute_Simulation function.

VII. Conclusion & Future Work

The time-triggered organic computing architecture improves the reliability, safety, and determinism of distributed embedded computer systems by combining self-x properties of ADNA and AHS with time-triggered concepts. In the TTOC simulator, the execution of tasks based on the generated schedule can be tested. However, this simulator contains no communication network. To perform the message exchange between computing nodes in a time-triggered manner, OMNeT++ simulates a network that supports these features. In order to achieve acceptable results, it is necessary to coordinate the two simulations. This is why, in this paper, we have introduced a co-simulation controller that controls the execution of TTOC and OMNeT++ simulators. To test the interface, various scenarios that include different physical and application models described in ADNA files were used. Examination of the log files showed that the co-simulation controller, TTOC simulator, and OMNeT++ behaved correctly.

Future work will involve integrating the entire system with an autonomous driving vehicle simulator (i.e., CARLA). The CARLA simulator will provide real-world application tasks, which will further test and validate the TTOC architecture.

Acknowledgment

This work was supported by research project SelfAutoDOC funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK).

Fig. 7. BMWK Logo

References

[1] A. Ranganathan and R. H. Campbell, "What is the complexity of a distributed computing system?" *Complexity*, vol. 12, no. 6, pp. 37–45, 2007.

[2] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[3] R. Obermaisser, *Time-triggered communication*. CRC Press, 2018.

[4] M. Qosja, S. Meckel, and R. Obermaisser, "Simulator for time-triggered organic computing," *Procedia Computer Science*, vol. 220, pp. 127–134, 2023.

[5] G. Jetschke, *Mathematik der Selbstorganisation*. Springer, 1989.

[6] "Dfg schwerpunktprogramm 1183 organic computing," last accessed: 2024-02-15. [Online]. Available: https://gepris.dfg.de/gepris/projekt/5472210

[7] U. Richter, M. Mnif, J. Branke, C. Müller-Schloer, and H. Schmeck, "Towards a generic observer/controller architecture for organic computing," in *INFORMATIK 2006 – Informatik für Menschen, Band 1*, C. Hochberger and R. Liskowsky, Eds. Bonn: Gesellschaft für Informatik e.V., 2006, pp. 112–119.

[8] J. Branke, M. Mnif, C. Müller-Schloer, H. Prothmann, U. Richter, F. Rochner, and H. Schmeck, "Organic computing–addressing complexity by controlled self-organization," in *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (isola 2006)*. IEEE, 2006, pp. 185–191.

[9] B. Jakimovski, M. Litza, F. Msch, and A. e. Sayed Auf, "Development of an organic computing architecture for robot control," in *INFORMATIK 2006 – Informatik für Menschen, Band 1*, C. Hochberger and R. Liskowsky, Eds. Bonn: Gesellschaft für Informatik e.V., 2006, pp. 145–152.

[10] M. Roth, J. Schmitt, R. Kiefhaber, F. Kluge, and T. Ungerer, "Organic computing middleware for ubiquitous environments," in *Organic Computing—A Paradigm Shift for Complex Systems*. Springer, 2011, pp. 339–351.

[11] F. Kluge, J. Mische, S. Uhrig, and T. Ungerer, "An operating system architecture for organic computing in embedded real-time systems," in *Autonomic and Trusted Computing: 5th International Conference, ATC 2008, Oslo, Norway, June 23-25, 2008 Proceedings 5*. Springer, 2008, pp. 343–357.

[12] U. Brinkschulte, "An artificial dna for self-descripting and self-building embedded real-time systems," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 14, pp. 3711–3729, 2016.

[13] U. Brinkschulte, M. Pacher, and A. v. Renteln, "An artificial hormone system for self-organizing real-time task allocation in organic middleware," in *Organic Computing*. Springer, 2009, pp. 261–283.

[14] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation: State of the art," *arXiv preprint arXiv:1702.00686*, 2017.

[15] "Functional mock-up interface for cosimulation v1.0," last accessed: 2024-01-20. [Online]. Available: https://fmi-standard.org/

[16] L. Belmon, Y. Geng, and H. He, "Virtual integration for hybrid powertrain development; using fmi and modelica models," in *Proceedings of the 10th International Modelica Conference*, 2014, pp. 10–12.

[17] A. Abel, T. Blochwitz, A. Eichberger, P. Hamann, and U. Rein, "Functional mock-up interface in mechatronic gearshift simulation for commercial vehicles," in *9th International Modelica Conference. Munich*, 2012.

[18] U. Pohlmann, W. Schäfer, H. Reddehase, J. Röckemann, and R. Wagner, "Generating functional mockup units from software specifications," in *9th Modelica Conference*, 2012, pp. 765–774.

[19] A. Elsheikh, M. U. Awais, E. Widl, and P. Palensky, "Modelica-enabled rapid prototyping of cyber-physical energy systems via the functional mockup interface," in *2013 workshop on modeling and simulation of cyber-physical energy systems (MSCPES)*. IEEE, 2013, pp. 1–6.

[20] C. M. Legaard, C. Gomes, P. G. Larsen, and F. F. Foldager, "Rapid prototyping of self-adaptive-systems using python functional mockup units," in *Proceedings of the 2020 Summer Simulation Conference*, 2020, pp. 1–12.

[21] M. Neghina, C.-B. Zamfirescu, P. G. Larsen, K. Lausdahl, and K. Pierce, "Multi-paradigm discrete-event modelling and co-simulation of cyber-physical systems," *Studies in Informatics and Control*, vol. 27, no. 1, pp. 33–42, 2018.

[22] H. Lin, S. Sambamoorthy, S. Shukla, J. Thorp, and L. Mili, "Power system and communication network co-simulation for smart grid applications," in *ISGT 2011*. IEEE, 2011, pp. 1–6.
[23] B. M. Kelley, P. Top, S. G. Smith, C. S. Woodward, and L. Min, "A federated simulation toolkit for electric power grid and communication network co-simulation," in *2015 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. IEEE, 2015, pp. 1–6.
[24] U. Brinkschulte, M. Pacher, and B. Betting, "A simulator to validate the concept of artificial dna for self-building embedded systems," in *2014 IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems Workshops*. IEEE, 2014, pp. 160–169.
[25] U. Brinkschulte, "Technical report: Artificial dna-a concept for self-building embedded systems," *arXiv preprint arXiv:1707.07617*, 2017.
[26] A. Varga, "Omnet++," in *Modeling and tools for network simulation*. Springer, 2010, pp. 35–59.
[27] "Omnet++ homepage," last accessed: 2024-05-24. [Online]. Available: https://doc.omnetpp.org/omnetpp/manual/
[28] L. Mészáros, A. Varga, and M. Kirsche, "Inet framework," *Recent Advances in Network Simulation: The OMNeT++ Environment and its Ecosystem*, pp. 55–106, 2019.