Co-Simulating the Internet of Things in a Smart Grid use case scenario

Johannes Kölsch, Axel Ratzke, Christoph Grimm
Chair "Design of Cyber-physical Systems"
TU Kaiserslautern
Kaiserslautern, Germany
koelsch|ratzke|grimm@cs.uni-kl.de

***Abstract*—This paper presents a virtual environment for simulation and validation of IoT networks in real life scenarios. The virtual environment is used to generate simulation models for realistic situations with inputs from IoT use cases and their requirements. The interaction between discrete computing parts and continuous-time dynamic parts is demonstrated in a smart grid use case. However, the cross-domain interoperability between grid components and communication infrastructures is incorporated into the use case as well. The platform is developed within the VICINITY project funded by the EU Horizon 2020 program.**

***Index Terms*—Co-simulation, Internet of Things, Smart Grid**

I. Introduction

Nowadays, the number of applications in which IoT networks are deployed grows rapidly. These infrastructures operate in different domains such as smart cities, energy, eHealth, and transportation and behave like isolated islands in the global IoT ecosystem. Their interconnection is a big challenge that still needs to be resolved. One promising approach towards interoperability of IoT networks across different domains is proposed by the VICINITY project. The project is supported by European Union Horizon 2020 program with the duration of 4 years (Jan. 2016 - Dec. 2019) and the consortium of 15 partners from 7 different countries.

A. The VICINITY project

The goal of the VICINITY project is to develop a platform that connects isolated IoT infrastructures into one global ecosystem called **virtual neighborhood** where users can select to which other systems their smart objects should be connected in a peer-to-peer network. The semantic interoperability of smart objects coming from different operators and using different standards is enabled with the semantic model integrated into the VICINITY platform [1]. The VICINITY architecture offering interoperability "as a service" is shown in Figure 1. It is based on a decentralized, bottom-up and cross-domain approach that resembles a social network, where users can configure their setups, integrate standards according to the services they want to use and fully control their desired level of privacy in a P2P (peer to peer) network.

As a part of the VICINITY project, this work has been supported by EU (European Union) program Horizon 2020 under grant agreement number 688467.

978-1-7281-0637-3/19/$31.00 ©2019 IEEE

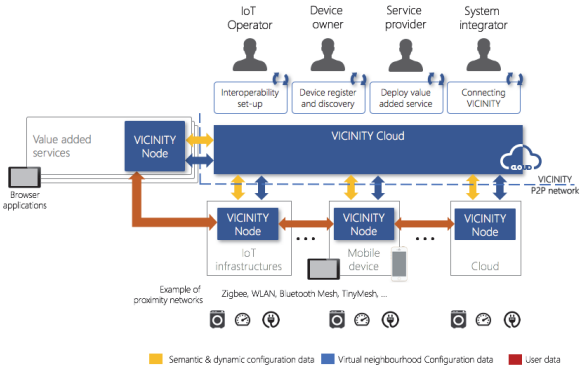

Fig. 1. High-level VICINITY architecture [2]

The focus of the work presented in this paper is to build a virtual environment for simulation and validation of IoT infrastructures and their use cases in real life scenarios before their real deployment. Here, a smart parking and energy use case at one of the VICINITY pilot sites, located in Tromsø, Norway, will be used for demonstration.

B. Simulation of IoT networks in realistic scenarios

The size of the IoT market today is growing at enormous speed and will continue to do so. The number of connected devices has already exceeded the worlds human population [3]. In such complex IoT networks simulation plays an important role for the early-phase validation before their deployments in the real-life world.

[4] proposes the agent-driven *Smart Shire* (S^3) simulator that supports large-scale simulations with different communication mechanisms such as TCP/IP, MPI and shared memory. In [5] the authors extend *Smart Shire* towards IoT for the multi-level cross-domain simulation where the agent-driven S^3 simulator [4] is used for modeling objects and services at the higher level and the discrete-event OMNeT++ simulator[1] for modeling communication between objects at the lower level. After a number of experiments, they came to the conclusion that the simulation performance degrades as the number of entities simulated at the lower level with OMNeT++ increases.

[1]https://www.omnetpp.org

To deal with this problem, [6] proposes an approach based on parallel and distributed simulation (PADS). The approach in [6] is based on the use of a hybrid simulator at the lower level where OMNeT++ is combined with the Matlab/Simulink-based simulator ADVISOR and works with it in succession. However, this approach does not provide solutions for general problems of the PADS simulation, such as lack of interoperability among simulators and lack of approaches for the automatic deployment and management of simulators on the distributed infrastructure.

Another hybrid simulation approache was proposed by [7] and [8]. [7] replaces the agent-based simulator S^3 with the Agent-based COoperating Smart Object (ACOSO) simulator for modeling smart objects in IoT networks. [8] presents the general-purpose hybrid simulation platform that supports simulation of interconnected IoT devices characterizing them by mobility, communication, and energy models.

This paper brings the following novelties over state of the art: The proposed simulation framework supports multi-level simulation using only one simulation technique based on discrete-event simulation. The framework allows dynamic switching between models at different levels of abstraction that simplify significant portions of a simulated IoT network with fairly rough-grained time resolution. This further allows us to dynamically observe details that are relevant and filter ones that are not of interest for a particular simulation scenario.

The following section describes the proposed OMNeT++-based framework for simulation of real-life scenarios for IoT platforms and infrastructures in detail. The rest of the paper is organized as follows. Section III describes the demonstration of the proposed framework on a smart parking use case of the VICINITY pilot site in Tromsø. Sections IV and V conclude the paper and identify future work, resp.

II. OMNeT++-based Simulation Framework for IoT Networks

This section describes the proposed multi-level simulator including its core framework and the integration with the OMNeT++ network simulator. Before the implementation started, the prevailing requirements for a simulator of IoT networks from SoA (State of the Art) have been collected. In addition, some requirements arisen from the VICINITY project have also been considered. These requirements are listed in the following:

1) Possibility to simulate thousands of interconnected devices, as stated in [5]
2) Support for running proactive approaches [5]. While highly detailed simulation runs can provide insight into the fine-grained processes within interactions of different devices in IoT networks, simulation performance degrades as the simulation runs tend to be very slow.
3) Capabilities to perform the simulation of hardware in the loop
4) High scalability of scenarios to be simulated. Real-time capabilities of a simulation framework should not be lost in the case of a large scale simulation of thousands of entities that communicate with each other.
5) Possibility to employ parallel and distributed simulations, either out of the box or through later adaption of these features, where needed [6]
6) Fast model development for fast deployment in use cases.
7) Possibility to handle heterogeneity of cross-domain IoT networks
8) Possibility to integrate further domain-specific simulators into the framework with little effort.

A. The Core Simulation Framework

The core simulation framework is based upon a lightweight implementation of the Parallel Discrete-Event Systems (PDEVS) formalism [9]. The goal of Parallel DEVS is to enable models to receive multiple external events simultaneously and process them in a single step. In addition to the two state change functions for internal events and external events of atomic models, PDEVS introduces a third state change function for confluent events. Basically, it is a special case of the external transition function where the elapsed time equals the result of the time advance function of the model. It is used to determine a new state of the model whenever an internal and an external event collide.

The core simulation framework contains a simulator class that manages the simulation by advancing the models' simulation time, scheduling autonomous events, and routing input and output between models. The behavior of the simulator class is adjustable to different challenges through specialized implementations of its components like container classes, event listeners or the future event schedule. The general composition of the framework is shown in Figure 3.

Hybrid models. Following the PDEVS formalism, the used models can be divided into two categories:

1) Atomic models. They form the smallest building block within the system model. Their state is defined indirectly by their state change functions for internal, external and confluent events, their output function and the time advance function.
2) Network models. Network models are a collection of atomic models and possibly other network models connected. Their primary task is to provide routing information to and from the models it contains. The state of a network model is indirectly defined by the state of all its components' models.

In order to represent continuous time systems, hybrid models are used. These implement the same interface as plain atomic models but also provide mechanisms to approximate continuous time.

B. Hierarchy Middleware

The hierarchy middleware implements in details the various levels throughout the model topology. The goal is to exchange atomic models with network models that are more detailed during runtime. The atomic model is used to approximate the

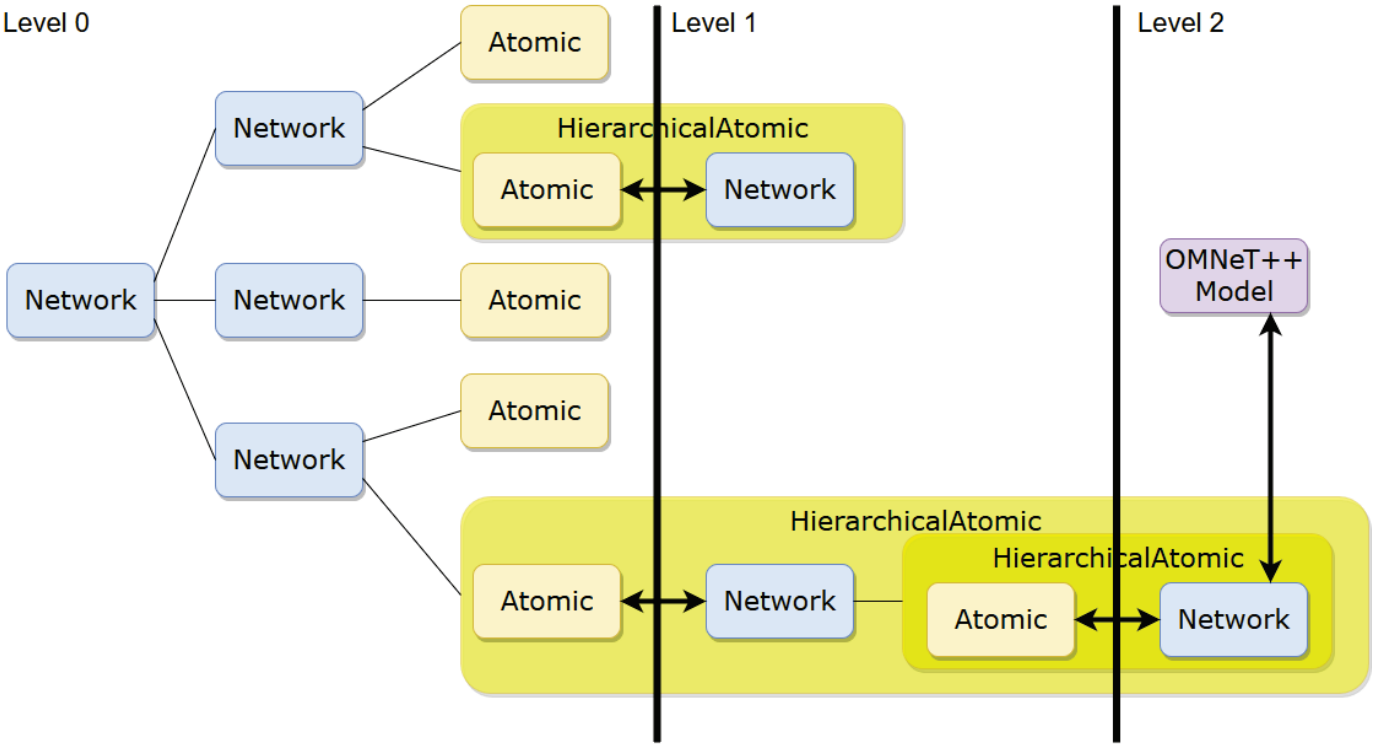

Fig. 2. The model tree and organization of hierarchical levels

desired behavior at a higher level of abstraction for better simulation performance, while the dynamically interchangeable network model offers a higher level of detail at the price of smaller steps in time.

This is realized through the introduction of a new kind of model, the HierarchicalAtomic. It is composed of the aforementioned atomic and network models and an instance of a simulator. From the outside, it appears to be a regular atomic model but internally it is able to switch from one model to another. When the model switches from its internal atomic model to the internal network model, the resolution of simulation time becomes finer grained due to smaller time advances within the Network.

The organization in atomic and network DEVS models and DEVS' closure under coupling allow for easy (re-)arrangement of model building blocks to more advanced and complex models in a tree-like structure. This procedure is supported by the here proposed approach. As can be seen in figure 2, the here introduced model that contains parts of the level architecture can easily be treated just as plain atomic DEVS model that is part of a network model in all positions in the tree structure. Moreover, the atomic models that belong to the contained network model can be exchanged with the new model as well, thus forming the different levels of hierarchy within the level of detail and time advancement of the simulation.

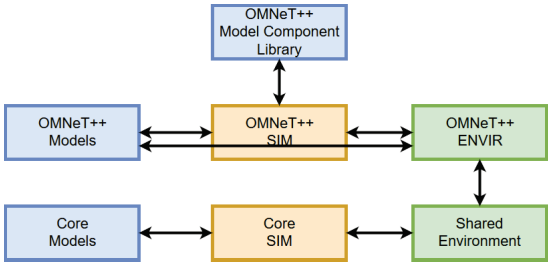

Fig. 3. Architecture of the framework

Regarding the approach to modeling, a bottom-up procedure is recommended, where the deepest and most detailed level is designed in its whole and then partitioned into smaller leaves of the tree. As the performance gain of the dynamic exchange between models with varying levels of detail is dependent on the specific scenario, it is possible that more efficient partitions of models along the tree are only eventually evaluated during simulation runtime. The bottom-up construction here ensures that the main part of the scenario has to be developed only once and then merely has to be partitioned accordingly. Also, if the models that are to be exchanged are the kind of whole protocol-stacks or space-divided parts of the environment, they, in turn, can be heavily reused.

C. *OMNeT++ integration*

The powerful network simulator OMNeT++ is used as a basis for the simulator proposed in this work. Together with its INET extension, it offers simulation tools for the communication, energy consumption, and movement of cyber-physical systems.

1) Shared Environment: To combine both, OMNeT++ and the core simulator described in Section II-A in a single simulation loop, a shared environment is proposed. It provides the functionality of both simulators. To that end, the command-line off-the-shelf environment of OMNeT++ is extended with capabilities needed to supervise the core simulation framework. Additionally, the sequential scheduler of OMNeT++ is extended to consider the future event schedules of both simulators. It produces synchronization events for both simulators that determine which simulator is executed first. These events are executed through a callback to the shared environment which then advances the state of the simulators.

2) OMNeT++Interface: Models of both simulators can interact through a designated class that has to be specialized for a concrete scenario. It implements characteristics of the modules used by OMNeT++ and the event listener integrated into the core framework. Hence, it is able to react to inputs from both simulators and can switch from one simulator to another. This way, the models of both simulators can communicate with each other across the level hierarchy.

III. Case Study: Smart Energy Use Case

To illustrate the applicability and performance of the developed multi-level simulator, we modeled and simulated a smart energy use case. This particular use case describes a smart energy scenario within an city. The city has a photovoltaic system and a windmill as power suppliers and a parking lot and a couple of houses as consumers. Electric vehicles can move inside the city and the parking lot. By using a smart parking service through a mobile app, users of the system can request to reserve their parking slot of choice within the participating parking facilities. The availability of the parking slots is then displayed through the mobile app as well as through the optical indicators located on the respective parking slots for random people, that do not participate in the smart parking service.

The described scenario has been modeled and simulated using the proposed approach at three distinct levels of abstraction: The first two higher levels have been implemented only using classes provided by the core simulator, presented in Section II. The third (lowest) level has been implemented with OMNeT++ 5.4.1[2] and its INET extension 4.0 [3].

A. The highest abstraction level - Level 0

The highest level of abstraction models only the most abstract processes that are needed to provide basic information for the following lower levels of the simulation scenario. This is shown in Figure 4.

The *CarGenerator* atomic model acts as a source to the rest of the modules and provides the information needed to simulate users and random cars at the lower abstraction levels. This information is then passed to the *CarProcessor*.

The *CarProcessor* then determines if the received information is used to simulate a scenario with a random visitor of the parking facility or with a user of the smart parking mobile app.

[2]https://www.omnetpp.org/21-articles/3752-omnet-5-4-1-released

[3]https://inet.omnetpp.org/2018-06-28-INET-4.0.0-released.html

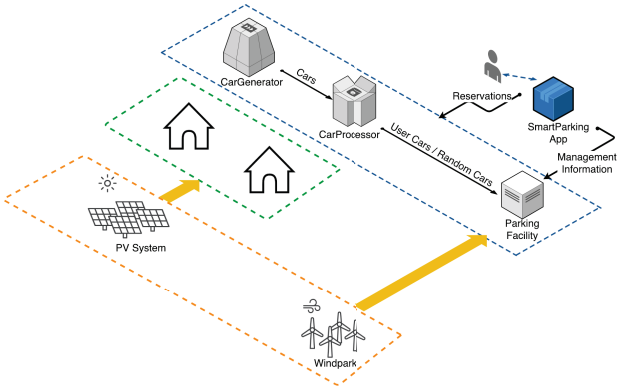

Fig. 4. Smart Energy use case: Level 0

In the latter case, information about the desired parking slot is generated and used in an attempt to make a reservation via the model of the smart parking app. If this reservation fails, the information of the app user is treated like information about a random visitor of the facility and sent further to the *ParkingFacility*.

Once entering the *ParkingFacility* model, the received information is used to model the abstract behavior of both random visitors and smart parking service users competing for available parking slots, parking and subsequently leaving the facility again.

Users of the app that succeed with a reservation will directly target their desired parking slots while random arrivals and users that failed to reserve their desired slot will choose the first free available parking slot. When arriving at the slot, it will be determined if it is still free or in the meantime has already been reserved by a user or taken by another random car that arrived first. If it has already been taken, they will head for the first free parking opportunity again. If they do not succeed in finding one, the car will leave the parking facility. If the parking process succeeds, the car will occupy the chosen parking slot for a while and then subsequently leave the parking facility again.

B. The middle abstraction level - Level 1

The following level of abstraction has been used to further detail the processes inside the *ParkingFacility*. It is shown in Figure 5. It divides the raw *ParkingFacility* into three different parking decks that internally mimic the behavior of the parking facility in Tromsø.

The information about car arrivals will be forwarded to the different parking decks in sequence; When entering the facility, the parking decks have to be traversed until the desired parking spot is reached. When a car leaves the *ParkingFacility*, the decks have to be traversed again in order to reach the exit. Additionally, the *ParkingDeckControl* is used to send information from and to the model of the app.

C. The lowest abstraction level - Level 2

The lowest level of the simulation scenario has been modeled with OMNeT++ 5.4.1 and INET 4.0. Here, the informa-

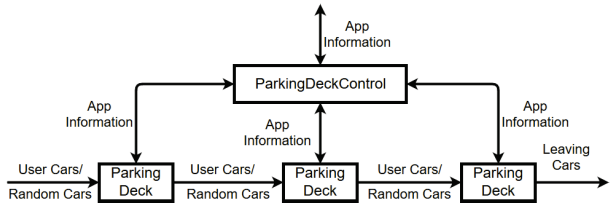

Fig. 5. Smart Energy use case: Level 1 - the Parking Facility

tion produced by the higher levels described above is used to dynamically instantiate simulated entities and to represent the communication between sensors, actuators, and the app with the advanced capabilities of INET.

At this level, the parking facility has been modeled as an OMNeT++ *compound module* and the single parking decks as submodules of it. Although the different submodules can interact with each other across submodule boundaries, only the ones which are associated with the respective active higher level parts of the simulation will be actually active.

If the higher level parking deck model switches to the respective part of the OMNeT++ module at Level 2, cars will be created as mobile nodes with specific characteristics. The characteristics depend on the information generated at the levels above and the cars behavior is determined by the corresponding states at Level 1. Depending on if a car is now in the phase of searching for a parking slot or if it is already parked or even leaving the parking deck, the wireless node will be created and its goals will be set accordingly. One such parking deck modeled in OMNeT++ can be found in Figure 6.

Every Car has a battery, that is discharging as long as the car is moving inside the environment. When a car is parked inside the parking facility its accumulator is charged. When it is fully charge, it stops charging and the car is ready to drive away.

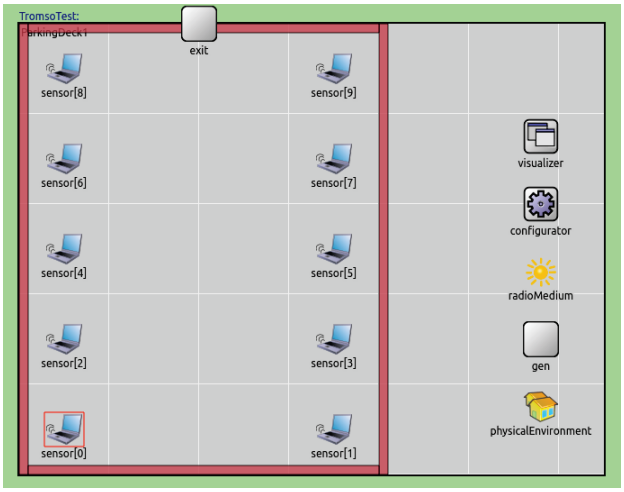

Fig. 6. Smart energy use case: Omnet++ Model of Parking Deck at Level 2

D. Results

In order to examine the scaling capabilities of the proposed simulator, the lowest abstraction level has been divided into three parts (one for each parking deck at Level 2). In the following, they will be denoted as L2a, L2b, and L2c. Level 0 and Level 1 will be denoted as L0 and L1, respectively.

All experiments have been performed on an Ubuntu virtual machine installed on a Lenovo T420s. It uses 2 cores of the underlying Intel Core i5-2520M architecture and has 4 GB RAM.

As stated in requirement (4) in Section II, the ability to run simulations in almost real-time is of particular importance and for this reason, the experiments have also been centered around the wall clock time (WCT) as an indicator.

The simulation time for each simulation run was set to 120 seconds and the wall clock time for the single runs was recorded. The wall clock times reported in Table I represent the average of several independent simulation run times.

As expected, the simulation runs with only the first level L0 and the first two levels L0 and L1 active have achieved similar average wall clock times. Since L0 was used to produce information for the levels below, the runs at this level finished relatively fast. L1 served only as a space division for the L2, and therefore the overhead added by this level is negligible (only 0.256 seconds). This is also shown in Table I.

As expected, the first real spike in the average WCT has occurred with the activation of L2a. In this case, the average WCT increases from 4.516s (with the activated L0 and L1) to 6 seconds. However, with the addition of the next two levels L2b and L2c respectively, the overhead did not increase dramatically and even for the last case, the WCT dropped back to 5.903 seconds.

This was a direct consequence of the architecture of the underlying model; the three parking decks that are represented by the levels L2a - L2c, are traversed by cars in sequence. Therefore, we suppose that the simulation time of 120 seconds is not sufficient to create an adequate number of nodes at the later levels L2b and L2c respectively. For this reason, the simulation with all levels activated was repeated with an additional number of cars as mobile nodes created from the beginning of the simulation instead of dynamically relying on the information provided by L0. The WCT for this scenario can be seen in Table II; it is again expressed as the average of times required for single simulation runs.

Level	WCT
L0	4.260315s
L0 + L1	4.516574s
L0 + L1 + L2a	6.003036s
L0 + L1 + L2a + L2b	6.627343s
L0 + L1 + L2a + L2c	5.9030517s

TABLE I
WALL CLOCK TIME OF COMBINATIONS OF DIFFERENT LEVELS

Level	WCT
L0 + L1 + L2a + L2b + L2c	9.442795s

TABLE II
WALL CLOCK TIME WITH INITIAL POPULATED LEVELS

IV. Conclusion

In this paper, we presented an approach towards modeling *Internet of Things* infrastructures together with the implementation of a prototype of a multi-level simulator. The approach proposes interconnection between models at different abstraction levels within the *discrete event simulation framework* and has been demonstrated on a smart parking use case of the *VICINITY* pilot site in Tromsøtogether with energy considerations.

The specific solution for this use case has used 3 levels of simulation. The first level has been used to generate abstract information on the general movement of simulated entities and communication between them. Furthermore the energy structures around the use case have been modeled. The second level has served as a space division for the lower level. It produced more detailed information about the movement that has been used as a basis to dynamically activate the different parts of the lowest level - Level 2. Level 2 has then used the powerful network simulator OMNeT++ with the INET framework to simulate the details of a smart parking service, the movement of users, the communication between them, the charging behaviour of cars and the environment.

The experiments executed on the use case show that with all three levels active the execution time increases almost two times. With respect to interoperability, the simulator has proven to fulfill the requirements for an IoT simulator, particularly Requirements (7) and (8) listed in Section II. This is achieved through the following capabilities:

1) Dynamic switching between models at different levels of abstraction
2) Spreading multiple simulation engines across the model tree shown in Figure 2
3) Modeling and simulation of mobile system entities and their communication through the OMNeT++ integration

V. Future Work

The extension of this work towards hardware in the loop simulation is work in progress. We think it can easily be integrated with the approach presented in [10] at the lower abstraction layers. For the higher abstraction layers, the basic idea is to open ports on the host machine for direct TCP/IP traffic with the hardware to be integrated. First tests with the VICINITY Gateway running on a Raspberry Pi are going on at the moment.

In its current state, the prototype supports the usage of continuous-time models through *hybrid models*. However, continuous interaction between such models can still pose some problems regarding the discrete-time architecture of the simulator. A solution to this was proposed in the generalized DEVS specification [11]. It uses polynomial events to approximate continuous output.

Large-scale IoT scenarios can massively profit from parallel and distributed simulation techniques. Through the integration of MPI, OMNeT++ supports parallel and distributed execution. Future work should use this provided architecture and enable the developed simulator to make full use of OMNeT++ capabilities.

The importance of the functional mock-up interface (FMI) for the simulation of CPS is quite obvious. For the integration of more domain-specific languages and simulators at the lowest level, FMI needs to be integrated into OMNeT++. We already integrated SystemC models into the simulation without using FMI. However, the need for more exact simulation results can greatly benefit from the integration of FMI and languages like Modelica.

Acknowledgment

As a part of the VICINITY project, this work has been supported by EU (European Union) program Horizon 2020 under grant agreement number 688467. [11]

References

[1] R. García Castro, "Vicinity d2.2: Detailed specification of the semantic model," Tech. Rep., 2017. [Online]. Available: https://www.vicinity2020.eu/vicinity/sites/default/files/documents/vicinity_d2.2_vicinitysemanticmodel_v1.0.pdf

[2] V. Oravec (leading author), "D1.6: VICINITY Architectural Design," Tech. Rep., 2017. [Online]. Available: https://www.vicinity2020.eu/vicinity/content/d16-vicinityd16architecturaldesign10

[3] D. Evans, "The Internet of Things How the Next Evolution of the Internet Is Changing Everything," Cisco Internet Business Solutions Group (IBSG), Tech. Rep., 2011. [Online]. Available: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[4] S. Ferretti and G. D'Angelo, "Smart Shires: The Revenge of Countrysides," in *Proceedings of the IEEE Symposium on Computers and Communications*, ser. ISCC '16. Washington, DC, USA: IEEE Computer Society, 2016.

[5] G. D'Angelo, S. Ferretti, and V. Ghini, "Multi-level Simulation of Internet of Things on Smart Territories," *Simulation Modelling Practice and Theory (SIMPAT)*, vol. 73, pp. 3–21, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X16302507

[6] ——, "Distributed Hybrid Simulation of the Internet of Things and Smart Territories," *To appear in Concurrency and Computation: Practice and Experience*, vol. 30, no. 9, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4370

[7] G. Fortino, R. Gravina, W. Russo, and C. Savaglio, "Modeling and Simulating Internet-of-Things Systems: A Hybrid Agent-Oriented Approach," *Computing in Science Engineering*, vol. 19, no. 5, pp. 68–76, 2017.

[8] G. Brambilla, M. Picone, S. Cirani, M. Amoretti, and F. Zanichelli, "A Simulation Platform for Large-scale Internet of Things Scenarios in Urban Environments," in *Proceedings of the First International Conference on IoT in Urban Space*, ser. URB-IOT '14. Brussels, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014, pp. 50–55. [Online]. Available: http://dx.doi.org/10.4108/icst.urb-iot.2014.257268

[9] G. A. Wainer, *Discrete-event modeling and simulation: a practitioner's approach*. CRC press, 2009.

[10] J. Kölsch, C. Heinz, S. Schumb, and C. Grimm, "Hardware-in-the-loop simulation for Internet of Things scenarios," in *2018 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, 4 2018, pp. 1–6.

[11] N. Giambiasi, B. Escude, and S. Ghosh, "Gdevs: a generalized discrete event specification for accurate modeling of dynamic systems," in *Proceedings 5th International Symposium on Autonomous Decentralized Systems*, March 2001, pp. 464–469.