# WoT-Phyng-Sim: Integrating Physics Simulations with IoT Digital Twins using the Web of Things

Fady Salama[1], Anatolii Tsirkunenko[1], Ege Korkan[2], Sebastian Käbisch[2], Sebastian Steinhorst[1]
[1] *Technical University of Munich*, Germany, Email: {fady.salama, anatolii.tsirkunenko, sebastian.steinhorst}@tum.de
[2] *Siemens AG*, Germany, Email: {ege.korkan, sebastian.kaebisch}@siemens.com

***Abstract*—The high fragmentation in the IoT landscape necessitated the introduction of the Web of Things (WoT), a set of standards by W3C. Among other things, the WoT defines a standardized human- and machine-readable JSON document called the Thing Description (TD) that is able to describe the Application Programming Interface (API) provided by devices on a network. This description already provides enough information to generate a simulation that can mimic the cyber behavior of the Thing. However, the TD does not capture the physical properties and behavior of Things and, therefore, cannot be used to generate a cyber-physical simulation of a Thing. In this paper, we introduce a new method and an open-source framework for setting up and automatically generating WoT-compliant cyber-physical Digital Twins (DTs) in Computational Fluid Dynamics (CFD) simulators that can simulate physical processes, can be controlled in real-time and require minimal human effort. Such a simulation can be used to verify the behavior of cyber-physical systems before and after deployment. We demonstrate the viability of our method using a conjugate heat transfer simulation of a smart home environment with multiple devices running in real-time. Furthermore, we perform a scalability analysis and conclude that our approach is able to simulate multpile devices while maintaining real-time behavior. Using our proposed approach, the process of setting up physically realistic WoT-compliant DTs becomes fast, easy, which bridges the gap between cyber-physical systems and the WoT.**

***Keywords*—Internet of Things, Web of Things, CFD, Digital Twins, Simulation**

## I. Introduction

Developing Internet of Things (IoT) applications is a complicated endeavor that requires a developer to investigate all the devices and their capabilities in a system and use them together to perform a certain task. But given the fragmented nature of IoT technologies, such an endeavor is costly in terms of time and effort and can be highly error-prone. Furthermore, IoT applications usually involve systems in which software and physical components interact together, adding to the complexity of developing and verifying the behavior of such networked cyber-physical systems. As such, IoT development benefits greatly from an ecosystem that allows for an easier and more streamlined way of interconnecting devices and that is capable of providing the cornerstones needed for a simulation-driven development process of the IoT applications.

The Web of Things (WoT) [1] promises to ease the development of IoT applications by introducing a set of standards based on well-established web standards that allow for describing the capabilities of IoT devices and entities, called Things in the context of this paper. The most important of these

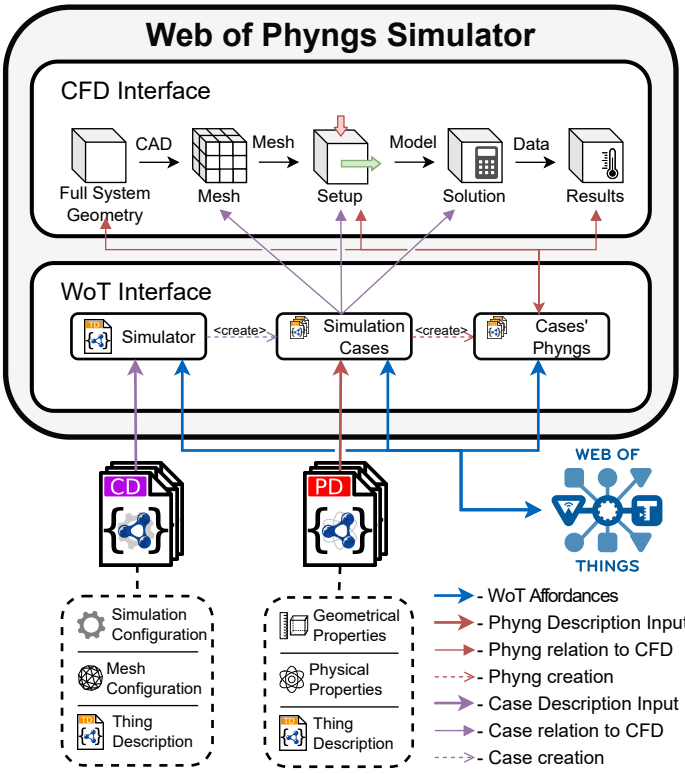



**Fig. 1:** Our proposed method **WoT-Phyng-Sim** is able to automatically generate Computational Fluid Dynamics (CFD) simulations for Digital Twins (DTs) using only descriptive JSON formats as an input. The simulator and simulated DTs provide Web of Things (WoT)-compliant interfaces to the outside, enabling remote setup and control.

standards is the Thing Description (TD) [2], a human- and machine-readable JSON-Linked Data (JSON-LD) document describing a Thing based on its network-facing interfaces and additional metadata. By having the TD of a Thing, a developer has enough information to develop IoT applications efficiently, without the need to understand low-level details about the system's devices.

***Problem Statement:*** Not only does the TD streamline the development of IoT systems, but it can also provide the descriptions needed for generating an interactable Digital Twin (DT) that provides the same interface as the described Thing and simulates its behavior. [3] already shows how to generate such a virtual instance that is able to mimic the behavior of a Thing described with a TD. However, let us imagine simulating a simple cyber-physical system such as a smart home scenario that includes a smart heater and a

```
1  {"@type" : "Thing",
2  "title": "Heater",
3  "links" : [{"rel": "type",
4    "href": "http://example/htModel.tm.json",
5    "type": "application/tm+json"}],
6  "properties" : {"state":{
7    "title": "Heater state",
8    "type": "string",
9    "enum": ["on","off","error"],
10   "forms": [{"href": "http://ht/state"}]}},
11 "actions": { "turnOn":{
12   "title": "Turn the heater on",
13   "forms": [{"href": "http://ht/turnOn"}]}}}
```

**(a)** An example Thing Description (TD) of a smarter heater

```
1  {"@type" : "tm:ThingModel",
2  "title": "HeaterModel",
3  "links" : [{"rel": "self",
4    "href": "http://example/htModel.tm.json",
5    "type": "application/tm+json"}],
6  "properties" : {
7      "state": { "title": "Heater state",
8          "type": "string",
9          "enum": ["on","off","error"]}},
10 "actions": {
11     "turnOn": {
12         "title": "Turn the heater on"}}}
```

**(b)** An example Thing Model (TM) of a smarter heater

**Listing 1:** Listing (a) shows an example Thing Description (TD) of a smart heater that exposes one property affordance called `"state"` (Line 6) and one action affordance called `"turnOn"`. (Line 11). The TD also states that it implements a Thing Model (TM) by having a `"links"` entry that points to the Thing Model (TM) resource and stating that the `"rel"` (relation) of this link is `"type"` (Lines 3-5). The corresponding TM is shown in Listing (b), and is structured similarly. The key differences are the `"@type"` entries (Line 1 in both listings) and the missing `"form"` entries in the interaction affordance (Lines 10, 13 in (a))

temperature sensor. Here we can already identify the following shortcomings in current IoT simulators:

- IoT systems interact with the physical world around them and, therefore, simulating only cyber aspects of these systems is not sufficient for capturing the full behavior of these systems. In our scenario, we need to not only simulate how the heater and sensor interact with on the network, but also how the heater changes the environment and how this change propagates to the temperature sensor.
- Current DT simulators focus mainly on simulating the kinematics and dynamics of robotics and industrial machinery such as conveyor belts and lifts. There is a lack of simulators that simulate accurate physical interactions such as thermal processes and light propagation, which is needed to simulate IoT devices such as smart lights, heaters, and Air Conditioners (ACs).
- Simulators that do capture the physical impact of systems on the environment, such as CFD simulators, usually simulate slower than real-time and cannot be controlled in real-time. Consequently, virtual systems in these simulations cannot be considered real-time DTs.
- Physical simulators require a lot of training to learn how to use them, and setting up simulation scenarios in them is an arduous process.
- The TD does not have enough information to describe physical entities for simulations, i.e. it lacks descriptions for physical properties such as dimensions and materials.

***Contributions:*** In this paper, we try to solve the aforementioned shortcomings by introducing **WoT-Phyng-Sim**, a method and simulation framework that allows for generating WoT-compliant cyber-physical DTs that can be controlled in real-time and require minimal human effort for setting up. In particular, we perform the following contributions by:

- Introducing a method for automatically generating CFD simulations based on a JSON format called Phyng Description (PD), introduced in Section III-A2.
- Introducing the Case Description (CD) in Section III-A1, a JSON format for describing simulation cases and their parameters.
- Developing a methodology that takes both CD and PD as an input and automatically generates the corresponding Digital Twin in a CFD simulator as an output and monitors its real-time behavior, outlined in Section III-A.
- Implementing the aforementioned methodology as a fully open-source service provided by a WoT-compliant server that is both human and machine-controllable, described in Section III-B.
- Showcasing our method by solving conjugate heat transfer in a smart home scenario in Section V.

The rest of the paper is structured as follows: In Section II, we briefly explain both the W3C TD and TM, as well as give a brief overview of the process of setting a CFD simulation, which is needed to understand the method we propose in this paper. Section VII discusses related work and Section VIII concludes.

## II. State of the Art

### A. The Web of Things (WoT) and the Thing Description (TD)

The Web of Things (WoT) is an architecture that aims to facilitate the interoperability of IoT devices and virtual entities by giving each entity, called a Thing in this context, the ability to describe its API and any additional metadata needed to interact with it. This description is standardized as the W3C Thing Description (TD), a JSON-LD document that is both human- and machine-readable. An example of a TD is shown in Listing 1a. Things that produce their own TD, i.e. expose their TD to the network, are called Producers, while Things that consume a TD, i.e. can parse TD and use it to interact with the described Producer, are called Consumers. The TD abstracts all possible interactions with a Thing into 3 types of interaction affordances. These are:

1) **Property Affordances**, which represent exposed states of a Thing. These can be read or written to. Listing 1a exposes a property affordance called `"state"` that is of type `"string"` and can either be `"on"`, `"off"` or `"error"` (Lines 6-10).

```
1  {"title": "ChtCase",
2  "links" : [{"rel": "type",
3      "href": "http://example/chtModel.tm.json",
4      "type": "application/tm+json"}],
5  "caseConfig": {
6      "meshQuality": 55,
7      "parallel": true,
8      "cores": 8,
9      "endtime": 60000}}
```

**(a)** An example Case Description (CD)

```
1  {"title": "HeaterPhyng",
2  "links" : [{"rel": "type",
3    "href": "http://example/heaterModel.tm.json",
4    "type": "application/tm+json"}],
5  "phyProperties": {
6      "dimensions": [0.1, 0.8, 0.5],
7      "location": [0.1, 1.0, 0.2],
8      "material": "aluminium"}}
```

**(b)** An example Phyng Description (PD)

**Listing 2:** Listing (a) shows a Case Description (CD), a JSON format that describes the parameters of a simulation case. These parameters are listed using the `"caseConfig"`. The exact parameters may differ based on the simulation software and therefore, the CD must explicitly state that it conforms to a Case Model (CM) provided by the simulator, by linking to the CM's Universal Resource Identifier (URI) (Lines 2-4). Similarly, we introduce the Phyng Description (PD), shown in Listing (b), which lists all physical properties of a Thing that are needed to simulate it. The exact parameters may differ based on the simulation software and the type of device and, as such, the simulation accepts only PD that link to a Phyng Model (PM) provided by the simulator (Lines 2-4).

2) **Action Affordances**, which can be invoked to start a process in a Thing that usually takes time to execute. Listing 1a exposes an action affordance called `"turnOn"`. (Lines 11-13)
3) **Event Affordances**, that represent notifications and event stream that a Consumer can subscribe to.

All interaction affordances in a TD must include the `"form"` key (Lines 10,13). `"form"` takes an object as a value that contains a URI to access the interaction affordance and additional protocol-specific metadata if needed. Protocol semantics can be bound to interactions as describe in [4]. Additionally, any other links to the resources on the can be described using the `"links"` keyword, which takes an array of objects (Lines 3-5). Each of these objects contains the URI of the resource's link (Line 4), as well as the type of relation between the link and the TD (Line 3).

***Thing Model (TM):*** In addition to the TD, the specification also defines the notion of a Thing Model (TM). An example of a TM is shown in Listing 1b. The key difference between a TD and a TM is that a TM is designed to describe a Thing abstractly without any instance-specific data. As such, TMs do not include instance-specific metadata, `"form"`s or protocol-specific data. TDs can specify that they follow a certain TM as a template. Hence, TMs are akin to abstract classes in object-oriented programming languages, while TDs can be considered instanced objects. A TD may specify that implements a specific TM by adding an entry in its `"links"` array with the URI of the TM and specifying that this is a `"type"` relation (Lines 3-5 in (1a)).

### *B. Computational Fluid Dynamics (CFD) Simulation*

Computational Fluid Dynamics (CFD) is a numerical approach for solving and simulating fluid mechanics, with both liquids and gases being considered as fluids. There are many tools on the market, open-source and proprietary, that are able to perform CFD. However, the methodology for simulating remains the same for all of them, which can be described using the following steps:

1) **Geometry Construction:** A pre-processing step in which a human agent defines the geometry of the simulation. This step is typically done inside the simulation software or imported from a preexisting Computer-Aided Design (CAD) model. Geometries can consist of multiple different volumes separated by surfaces.
2) **Meshing:** This is a process of splitting the surfaces and volumes of any arbitrary, continuous geometry into discrete shapes and volumes, called elements. The accuracy of the simulation is proportional to the size of these elements, which smaller elements usually yielding more accurate solvers. However, if the speed of a fluid is particularly high, a very small element size could result in a worse solver. Modern CFD software allow the human agent to specify how many processor cores will be used for the simulation. Based on that, elements will be split between the cores.
3) **Simulation Setup:** In this step, a human agent tunes the simulation settings such as simulation duration, time step and solver, and also sets up boundary and/or initial conditions of the simulation. Boundary condition are the rules that specify how a fluid behaves on the surface/boundary of a volume, for example, specifying the direction and speed in with which a fluid exists a volume to enter the adjacent volume.
4) **Solution:** This step involves solving the partial differential equations that model the simulated physical phenomena numerically;
5) **Post-Processing:** a process of retrieving, analyzing, and visualizing or displaying the results.

All of these steps are meant to be done by human agents and require prior knowledge, training and experience to perform them. Important parameters to fine tune are for example the simulation time step and mesh quality. Furthermore, CFD simulations rely on having all rules and conditions specified before running the solver, and they are not meant to change during real-time. As such, these steps alone do not permit a real-time simulation of IoT devices that may change simulation conditions dynamically.

## III. WoT-Phyng-Simulator

CFD simulators provide a reliable method for simulating devices that affect fluids around them, which makes them excellent tools for simulating realistic DT effect on its physical

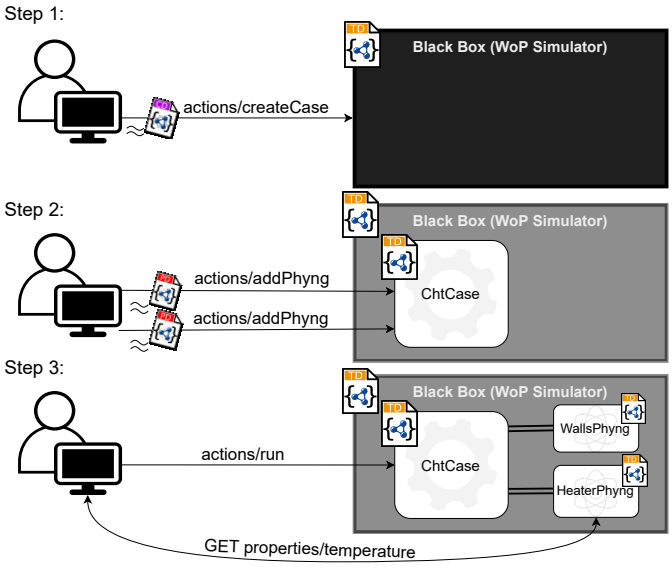



**Fig. 2:** The process of setting up a simulation is straightforward. In step 1, the simulation is launched with no cases initialized. A user then invokes an action to create a case with given Case Description. By using the exposed Thing Description of the case in step 2, the user is able to invoke an action to create a Phyng using a PD as an input. Finally, in step 3, the user can run the simulation case and interact with the created *heaterPhyng* heater.

environment. However, as stated in Section I, using CFD simulators requires prior experience and knowledge about the tooling, making it inaccessible to the wider community of IoT developers, who do not necessarily have the knowledge required to set up a CFD simulation. Additionally, CFD simulators are not meant to have rules and boundary conditions that change dynamically during run-time. This is, however, essential for mimicking the behavior of smart devices changing their parameters based on interactions. Furthermore, CFD simulators are not meant to simulate in real-time and, as such, do not provide a way to monitor their real-time capability.

### A. Method

To solve the shortcomings stated above, we propose the **W**eb **o**f **T**hings **Phy**sical Thi**ng** **Sim**ulator (**WoT-Phyng-Sim**), a method that permits the setup of simulation cases using a descriptive approach. Instead of manually setting up the geometry, the meshing and the boundary condition, a human agent needs only to provide standardized description formats that describe the simulation case, the simulation environment, and the smart devices to the simulator, which can be then parsed by the simulator and contain enough information to automatically generate all the needed components for a functioning simulation. Additionally, our method enables real-time control of CFD boundary condition and real-time monitoring.

In particular, **WoT-Phyng-Sim** takes a proposed JSON format called Case Description (CD) and a set of a proposed JSON format called **Phy**sical Thi**ng** Description (Phyng Description (PD)) as an input and is able to automatically generate a set of DTs that correspond to their PDs in a CFD simulation environment.

The proposed method separates the logic for setting up simulations into 3 distinct components. These are:

- **Simulator**: is the top level component that provides an interface for a client to create and delete simulation cases, allowing multiple clients to manage different cases with different parameters, solvers and Phyngs inside them.
- **Simulation Case**: can be interpreted as the CFD simulation instances and are contained with the simulator component. These entities are characterized by simulation properties, such as simulation time, step and mesh quality, and vary in regard to the Phyngs created inside them. It provides the interface for clients to change cases' properties, and create or delete their Phyngs;
- **Case Phyng**: is a component inside a simulation case. A physical representation of a Phyng typically includes its geometry, location, and physical properties. A client can then interact with a Phyng as if it was a real device, e.g., turn on or off a Heater Phyng.

The simulator component can be instantiated automatically without any prior setup, however, both the cases and the Phyngs need a corresponding description format that describe their capabilities, how their interface should look like, as well as the metadata needed to instantiate them. Therefore, we introduce two new JSON format namely the Case Description (CD) and the Phyng Description (PD) that carry the needed information for both simulation cases and Phyngs.

*1) **Case Description (CD):*** is a proposed JSON format that contains simulation case configurations and serves as a basis for the construction of simulation case. The simulation case configurations are named `"caseConfig"`, which is a mapping of the configuration parameters and their value. The exact configuration parameters may vary depending on the CFD software used and the specific implementation, as well as the type of case and solver used in that case. To ensure that Consumers are able to communicate CDs that the simulator is able to understand, the simulator provides a list of Case Models (CMs), similar to the TMs introduced in Section II, as resources that each can be accessed through an assigned URI. Hence, the simulator only accepts CDs that explicitly state that they conform to a CM provided by the simulator, allowing the simulator to instantiate the case correctly based on the URI of the linked CM.

*2) **Phyng Description (PD):*** The PD is another proposed JSON format that contains all parameters, required for a CFD simulation project to instantiate and control a Phyng, and its representation in WoT context. It describes physical properties under the `"phyProperties"` mapping, such as position, rotation and material of Phyng. The exact properties may vary depending on the software used, but also the type of the simulation case and the type of Phyng. Furthermore, each implementation may provide a different API for controlling their instantiated Phyng. Therefore, similar to the CD, the simulator provides a list of PMs in the same manner as shown for CD that shows the parameters and the interface of all Phyngs that the simulator supports. Hence, it is possible for the simulator to automatically generate a DT by only providing

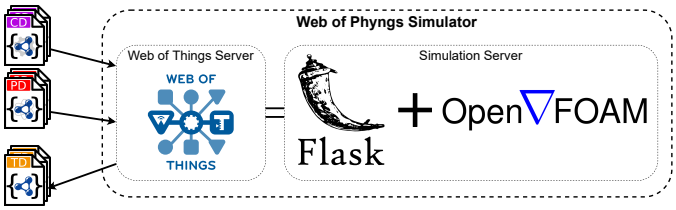



**Fig. 3:** The WoT-Phyng-Sim implementation consists of two servers: the simulation server built using OpenFOAM coupled with a Python Flask server, and the Web of Things (WoT) server built using node-wot. The simulation server provides a RESTFUL-API, which is encapsulated using the WoT server so that the whole implementation provides a WoT-compliant interface to the outside. The WoT server, the simulator, is able to accept Case Descriptions and Phyng Descriptions as inputs and provide Thing Descriptions of the simulator, cases and simulated Phyngs as output.

a TD that links to a TM provided by the simulator. The DT exposes the same interface as the one described in the TM. If the interface of the provided TD is different from that of the TM, a human-agent needs only to write a small program that consumes the DT's TD and then re-exposes the interaction affordances as defined in the TD.

*3) **Simulation process**:* The process of setting up a simulation can be summerized in the following steps:

1) Once the simulation server is running, a Consumer is able to consume the exposed TD of the simulation server and then create a simulation case by providing the respective CD.
2) Once the simulation case is instantiated, it exposes its TD, which can be used by a Consumer to add new Phyngs to the case by providing the corresponding PDs.
3) The instantiated Phyngs expose their TDs, which can be consumed by any Consumer to interact with them.

This flow is illustrated in Figure 2.

*4) **Mid-simulation control**:* Mid-simulation control, a requirement for a DT, is achieved by the following steps:

1) The simulator pauses the simulation.
2) In the case of a multicore simulation, if the mesh elements of the Phyng are split between multiple cores, these elements have to be composed back.
3) The changes to the Phyng's parameters are applied by going through all the mesh elements of the Phyng and changing the corresponding parameters.
4) In the case of a multicore simulation, mesh elements have to be decomposed.
5) The simulator starts the simulation again with the updated parameters.

*5) **Real-time Monitoring**:* To ensure that the simulator is able to maintain real-time solving speed, our approach monitors the time it takes to solve a time step, i.e. it checks if

$$\Delta t_{sol} < t_{step} \tag{1}$$

where $\Delta t_{sol}$ is the solving time for the current time step and $t_{step}$ is the time step. If the solving time is larger than the time step, the simulation is considered not running in real-time and the simulation notifies about it by raising a flag.

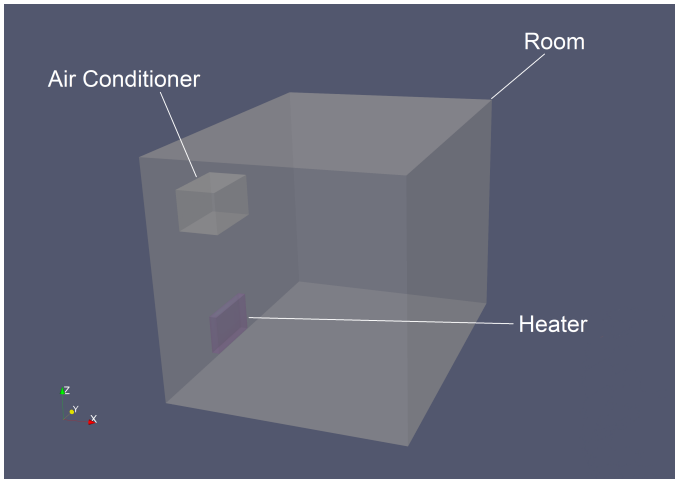



**Fig. 4:** Our use-case setup consists of a $3\text{m} \times 4\text{m} \times 3\text{m}$ room (outer cube) with a WoT-enabled heater (the lower red device on the left), AC (upper device on the left) and sensor (at position [1,1,1], not visible in image). This image was generated by ParaView[6], the GUI bundled with OpenFOAM to view the post-processed data.

On the other hand, if the solving time is much smaller than the time step, the simulator pauses the simulation solving to synchronize both the current time step with the elapsed time.

### B. Implementation

The implementation of the WoT-Phyng-Sim consists of two servers: the simulation server built using OpenFOAM[1] coupled with a Python Flask[2] server, and the Web of Things server built using node-wot[3]. OpenFOAM is an open-source CFD simulation software that offers multiple solvers for different simulation cases such as heat transfer, fluid flows and magnetism. A package named PyFOAM[4] allows a Python script to control OpenFOAM programmatically without the need for a user interface. Additionally, OpenFOAM uses a file system to store simulation cases and time steps during simulation, allowing other processes to access the simulation during run-time and modify it. As such, we expose the control mechanism of OpenFOAM using the Python Flask script that provides a RESTFUL-API to other processes listening in the local environment. This allows the coupling between OpenFOAM and a WoT-server, which provides the WoT-compliant interface that can be consumed to control the server from the outside. We implement cases that use the Conjugate Heat Transfer (CHT) solvers, which permit solving transient processes of heat transfer and airflow, however, extending our implementation with further solvers is possible.[5]

## IV. Case Study: Smart Home Simulation

To showcase our study, we go through the steps for setting up a CHT real-time simulation scenario from scratch to simulate the effect of smart home devices in a room, following the steps demonstrated in Section III-A and Figure 2.

[1] https://www.openfoam.com/ [2] https://flask.palletsprojects.com/en/2.2.x/
[3] https://github.com/eclipse/thingweb.node-wot
[4] https://pypi.org/project/PyFoam/ [5] Our implementation is open source and be viewed through https://github.com/tum-esi/WoT-Phyng-Sim

```
{"links": [{
    "rel": "type",
    "href":
↪   "http://localhost:8081/tms/cht/chtCase"}],
  "title": "smarthome",
  "sysProperties": {
    "meshQuality": 99,
    "cores": 4,
    "parallel": true,
    "blocking": false,
    "realtime": true,
    "endTime": 0,
    "background": "air"}}
```

**(a)** Case Description (CD)

```
{"links": [{
      "rel": "type",
      "href": "http://localhost:8081/tms/cht/
↪   chtHeater"}],
  "title": "heater",
  "phyProperties": {
    "dimensions": [0.1, 0.8, 0.5],
    "location": [0.1, 1, 0.2],
    "material": "copper"}}
```

**(b)** Heater Phyng Description (PD)

**Listing 3:** Listings (a) shows the CD used as a payload for setting up our use case similar to Listing 2a. Listings (b) shows the PD used as a payload for adding the heater Phyng to the room as described in Listing 2b. The payloads for the AC and the sensor can be viewed in the projects repository

The case study was performed using a dockerized version of our solution running on an 11th Generation Intel Core™ i7-11700KF Processor running on 3.60GHz and 32 GB of RAM. Our setup consists of a $3\text{m} \times 4\text{m} \times 3\text{m}$ room with a WoT-enabled heater, AC and sensor. We start creating a case by sending a request to the `"wopsimulator/actions/createCase"` endpoint provided by the simulation server. The request contains a payload with a link to the CHT case model provided by the `"tms/cht/chtCase"` endpoint, the title of the case `"smarthome"`, as well as the simulation properties of our case, including the `"real-time"` flag. The exact payload can be viewed in Listing Listing 3a Using this setup, one second of simulation took at the worst case ten seconds of calculation to simulate.

Once the case is set up, we can start building the simulation itself by sending POST requests to `"/smarthome/actions/addPhyngs"` with payloads containing links to the respective models provided by the simulation server, and the physical properties needed for simulation, such as dimensions and position and body material. We start by adding the walls, then all the listed devices. The exact payloads for adding the heater can be viewed in Listing 3b, the payloads for the other devices can be viewed in the project's repository. An overview of the resulting room can be seen in Figure 4

After adding the devices, the simulation can be started. The simulated Phyngs are then accessible through their respective endpoints and can be controlled during the simulation. Turning the heater on can be achieved by writing to the `"temperature"` property of the heater Phyng, which causes the temperature of the heater to change in the simulation as seen in Figure 5a and Figure 5b. On the other hand, the AC Phyng can be turned on by invoking its `"turnOn"` action, which causes the AC Phyng to push air in the room that a temperature, velocity and vertical angle as written to its respective properties.

Sensors added to the simulation, is set to return the value of the temperature, the position [1,1,1] inside the room. Currently, sensors do not have a corresponding physical representation in the room and offer the `"temperature"` property that is can only be read.

[6] https://www.paraview.org/

| Phyng | Dimensions, m | Temperature, K | Velocity, m/s | Angle, ° |
|---|---|---|---|---|
| window | 1 x 0 x 1 | 303.15 | -5 x 0 x 0 | – |
| door | 1 x 0 x 1 | 296.15 | 5 x 0 x 0 | – |
| heater | 0.1 x 0.8 x 0.5 | 350 | – | – |
| AC | 0.5 x 0.8 x 0.5 | 283.15 | 5 | 45 |

**TABLE I:** The dimensions, locations and specific values of Phyngs used in each evaluation metric are presented in this table. The dimensions are presented in $x \times y \times z$ in meters, the temperature in Kelvin. The velocity values refer to the air velocity in meters per second and the angle value in degrees for the AC refers to the angle in which the air exists from its slit

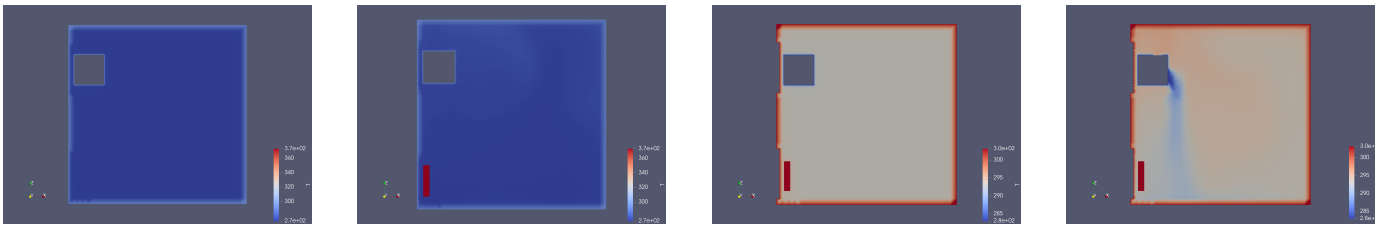
## V. Evaluation

We evaluate our approach in regard to its ability for simulating devices in real-time. We perform our evaluation using a server with an Intel(R) Xeon(R) E5-2680 v4 CPU operating with a maximum frequency of 3300 MHz and 14 cores, as well as a 128 GB DDR4 RAM with a frequency of 2400 MHz.

We investigate the solving speed of the simulator with regard to the mesh quality, the number of simulated devices, as well as the number of cores used for solving the simulation. We test each one of these parameters individually by fixing the other 2 parameters. The simulation time for each evaluation case is set to 60 s. Each test was then performed 5 times, and then the results were averaged. Our tests are performed separately on 4 simulated entities: a window acting as an air inlet to a room, a door acting as an air outlet for a room, a simple smart heater, and an actuatable AC, that is both an inlet and an outlet at the same time. All the entities, except the heater, are able to directly induce an airflow in the room. The exact physical properties of each entity are shown in Table I. The maximum number of simulated devices were 50 heaters and 9 doors, windows and ACs respectively.

### A. Mesh quality

To test the effect of the mesh quality on the solving time, we define the quality of the mesh as a percentage value, with $0\%$ being equal to 0.7 m and $100\%$ being equal to 0.1 m block size. Based on these metrics, the coarsest mesh quality possible for windows, doors and ACs is $13\%$, while the coarsest for



**(a)** Heater off **(b)** Heater on **(c)** AC off **(d)** AC on

**Fig. 5:** The above figures are heatmaps taken from cross-section of the room introduced in Figure 4. They show the action of turning on the heater (a)-(b) and the AC (c)-(d). The heat map is color coded, with blue corresponding to low temperatures and red corresponding to high temperatures. Each image includes a scale on the right denoting the color mapping based on the temperature in Kelvin.

the heaters is 50%. Otherwise, the simulator is not able to simulate. We evaluate the simulation time for 1 and 50 heaters, as well as 1 and 9 windows, doors and ACs, respectively. The results show that the mesh quality boundary for real-time simulation is 75% for 50 heaters, 67% for 9 windows and 9 doors, and 35% for 9 ACs, which are the most complex entities to simulate. For all entities, the solving time in relation to the mesh quality can be described with a cubic fit. The results for heaters and ACs are illustrated in Figure 6a and Figure 6b respectively. The results for doors and windows are omitted for brevity, but can be viewed using the following link[7].

### *B. Number of processor cores*

Another important metric is the impact of the number of cores on the simulation time. For this evaluation, the mesh quality was set heuristically to an amount that would allow the simulating of multiple instances of each entity in a reasonable time, which is 61% for heaters, 53% for windows and doors and 21% for ACs. The mesh decomposition step for multiprocessing is included in the solution step to enable run-time Phyngs' parameters adjustments.

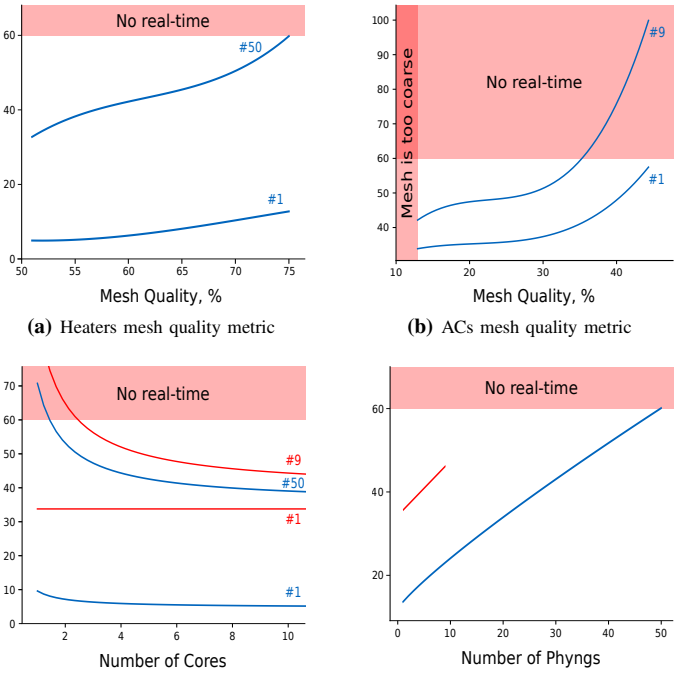
As can be seen in Figure 6c, the solving time decreases hyperbolically with increasing number of cores, reaching the point of diminishing returns at about 10 cores. The benefit of using multiple cores decreases due to the rising costs of processes communication between the decomposed regions.

### *C. Number of simulated entities*

The last metric evaluated was the effect of the number of simulated entities on the solving time. For this evaluation, we used the same mesh qualities as presented in the previous evaluation, and the number of cores was fixed at 12 cores. The results, which can be reviewed in Figure 6d, show that the solving time increases non-linearly with the increase of the simulated entities. However, given the stated mesh qualities, it was possible to always simulate the maximum number of simulated entities in real-time.

***Verdict:*** Based on these evaluations, we are confident that our approach is able to simulate multiple devices affecting the physical environment and still maintain real-time behavior and, as such, is viable as a simulation environment for DTs based on the metric proposed in Equation 1.

[7] https://github.com/tum-esi/WoT-Phyng-Sim




**(a)** Heaters mesh quality metric **(b)** ACs mesh quality metric

**(c)** Number of cores - heaters and ACs **(d)** Number of Phyngs - heaters and ACs

**Fig. 6:** We evaluate the solving speed of a CFD simulator using our proposed approach to test if it is viable for simulating Digital Twins that can be interacted with during time. We evaluated the solving time of our simulator for a sixty-second simulation depending on the mesh quality of the simulation, the number of simulated devices and the number of cores used. For all simulations, expect the latter, 12 cores were used. The figures above show our results for simulated heaters and ACs. Please note that all y-axes are displaying the average solving time in seconds and that Figures (c) and (d) combine the graphs of both heaters (blue) and ACs (red). The number near a graph corresponds to the number of devices simulated simultaneously.

## VI. Current Limitations

*a) Potential lower Modeling Capability:* While our approach tries to bridge the gap between the WoT and cyber-physical systems by abstracting away from the fine details of physical modeling, this abstraction limits the ability for modeling more complex scenarios. We try to solve this issue by allowing the client interacting with our simulation server to upload their own meshes in the form of an STL file, which will be used by the simulation instead.

*b) Modeling Complexity and Real-time:* The scalability of our approach relies heavily on the computing power of the server and the underlying physics solver algorithm. Numerical Physics simulators are inherently computationally intensive because the amount of calculations that are needed for each simulation case is proportional to the number of cells in it. To solve this, the simulator exposes the simulation time to allow the entities using it to sync their time with the simulation time and act accordingly. Furthermore, our proposed approach and architecture can be used with any kind of simulator and solvers, and is not restricted to the solver proposed here. Presenting faster numerical physics algorithms is beyond the scope of our work, but would bring a great benefit to the DT ecosystem and can be integrated into our open-source solution.

## VII. Related Work

There have been several approaches in the literature that aimed at simulating IoT and Industrial IoT cyber-physical systems. However, approaches like [3] and [5] focused mainly on networking aspects of IoT systems. On the other hand, [6], [7], aimed at automatically generating DTs and are also able to simulate physical interactions in the case of the latter. Yet both approaches are tailored for a very specific use case and cannot be used universally. [8] introduces a simulation toolkit that relies on the DPWS standard to automatically generate DTs. The kit also includes the notion of physical spaces. However, 3D environments rely on external software and the toolkit does not simulate physical processes. [9] introduced another approach for automatically generating WoT-based DT through learning and modeling the behavior of the Thing using a Markov Decision Process, but the approach does not simulate or model the effect of the Thing on the environment.

Commercially, Ansys simulation software provides the Twin Builder solution [10], which is capable of building, validating, and deploying DTs. Twin Builder can be used to manually build DT using model-based, 1D, and 3D simulations or a combination of these, which can then be connected to commercial IIoT Platforms to send and receive data in real-time and gain insights in regarding the monitored and simulated system. Received data can help parameterize the artificial intelligence model used for predictions, which is in turn used to perform better predictions in the feature.

Compared to these previous approaches in the literature, our approach is the first to try to bridge the gap between the cyber-physical systems and the WoT by permitting any entity, real or virtual, to automatically generate real-time DTs using a fully WoT-compliant Web API, needing only descriptive documents based on Web standards as inputs, and without the need for any prior knowledge regarding the inner workings of physical modeling or physical simulation techniques. The generated DTs are themselves interactable via the automatically generated WoT-compliant server for each device in real-time. Our approach is not tailored to a specific use-case or specific simulation software and is able to be extended or adopted with different software.

While commercial tools such as Ansys Twin Builder offer extended features, the open-source nature of our tool allow it to be easily extended and improved by the community.

## VIII. Conclusion

In this paper, we have presented **WoT-Phyng-Sim**, a method and simulation framework for simulationg WoT-enabled devices. Our approach takes descriptive JSON formats called CDs and PDs as an input, and is able to automatically generate a fully functioning CFD simulation with actuatable DTs as an output, with minimal human intervention and prior knowledge of physics simulators. We have demonstrated the viability of our approach by building a functioning DT of room with a smart heater, AC and sensor in a CHT case. Additionally, we performed a scalability analysis to evaluate the ability of our framework to simulate multiple DTs in real-time and have discussed potential limitations of our approach. Our work is a significant contribution to bridging the gap between Web technologies and physical simulations. Comparing our approach to related work, we argue that our tool is a first step towards this direction, which can be easily extended due to its open-source nature. We hope that our framework will inspire further research in this area and facilitate the development of WoT-compliant cyber-physical simulations and applications.

## References


[1] M. Kovatsch, R. Matsukura, M. Lagally, T. Kawaguchi, K. Toumura, and K. Kajimoto, "Web of Things (WoT) Architecture," 2020. [Online]. Available: https://www.w3.org/TR/2020/REC-wot-architecture-20200409/

[2] S. Kaebisch, T. Kamiya, M. McCool, V. Charpenay, and M. Kovatsch, "Web of Things (WoT) Thing Description," 2020. [Online]. Available: https://www.w3.org/TR/2020/REC-wot-thing-description-20200409/

[3] E. Korkan, E. Regnath, S. Kaebisch, and S. Steinhorst, "No-Code Shadow Things Deployment for the IoT," in *WF-IoT*, 2020, pp. 1–6.

[4] M. Koster and E. Korkan, "Web of Things (WoT) Binding Templates," 2020. [Online]. Available: https://www.w3.org/TR/2020/NOTE-wot-binding-templates-20200130/

[5] 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.

[6] G. S. Martínez, S. Sierla, T. Karhela, and V. Vyatkin, "Automatic Generation of a Simulation-Based Digital Twin of an Industrial Process Plant," in *IECON 2018*, 2018, pp. 3084–3089.

[7] S. Spellini, R. Chirico, M. Panato, M. Lora, and F. Fummi, "Production Recipe Validation through Formalization and Digital Twin Generation," in *DATE*, 2020, pp. 1698–1703.

[8] S. N. Han, G. M. Lee, N. Crespi, K. Heo, N. Van Luong, M. Brut, and P. Gatellier, "DPWSim: A simulation toolkit for IoT applications using devices profile for web services," in *WF-IoT*, 2014, pp. 544–547.

[9] L. Sciullo, A. Trotta, F. Montori, L. Bononi, and M. Di Felice, "WoTwins: Automatic Digital Twin Generator for the Web of Things," in *WoWMoM*, 2022, pp. 607–612.

[10] Ansys, "Ansys Twin Builder — Simulation-Based & Hybrid Analytics," 2021, accessed on 08.02.2023. [Online]. Available: https://www.ansys.com/content/dam/product/digital-twin/twin-builder/ansys-twin-builder-technical-datasheet.pdf