IoT Modelling for Digital Twins

Introduction

Welcome to the IoT Modelling for Digital Twins documentation. This project focuses on creating comprehensive models for Internet of Things (IoT) systems that can be represented as digital twins in virtual environments.

What are Digital Twins?

Digital twins are virtual representations of physical objects, processes, or systems that serve as the real-time digital counterparts of physical entities. In the IoT context, digital twins enable:

- Real-time monitoring and visualization
- Predictive analytics and simulation
- Enhanced decision-making capabilities
- Optimized performance and maintenance

What is CAPS?

The CAPS Modeling Framework was created to support the engineering of **Situational Aware Cyber-Physical Systems (SiA-CPS)**. SiA-CPS are systems consisting of a set of IoT devices such as sensors, cameras, RFID, and other monitoring tools used to continuously observe given indoor and outdoor spaces. The data gathered during this process is then transformed into actionable insights.

SiA-CPS systems require monitoring in both time and space, meaning they consist of:

- **Software components** (drivers and code running on the sensors)
- **Hardware components** (sensors, cameras, RFID, etc.)
- **Environmental interactions** (how software and hardware components interact with their surroundings)

Designing such systems poses challenges in ensuring consistency between managing software, hardware, and environmental views. To address this, the CAPS Modeling Framework provides a **multi-view approach** based on the IEEE/ISO/IEC 42010 standard, ensuring structured and coherent system modeling.

CAPS Installation Guide

Prerequisites

Before proceeding with the installation, ensure that your system meets the following requirements:

- Windows operating system (64-bit)(Tried on Windows expecting same will work for other systems as well)
- Internet connection
- Sufficient disk space (at least 5GB free)

Step 1: Download and Install Java 8

1. Visit the Java SE 8 Downloads page.
2. Accept the license agreement and download the Windows x64 JRE.
3. Run the installer and follow the on-screen instructions.
4. Verify the installation by opening the Command Prompt (`Win + R` , type `cmd` , and press Enter) and running: `sh`
   ```
   java -version
   ```
   Ensure it displays Java 1.8.

Step 2: Download and Install Eclipse Modeling Tools

1. Visit the Eclipse download page.
2. Download the Eclipse Installer for Windows (64-bit).

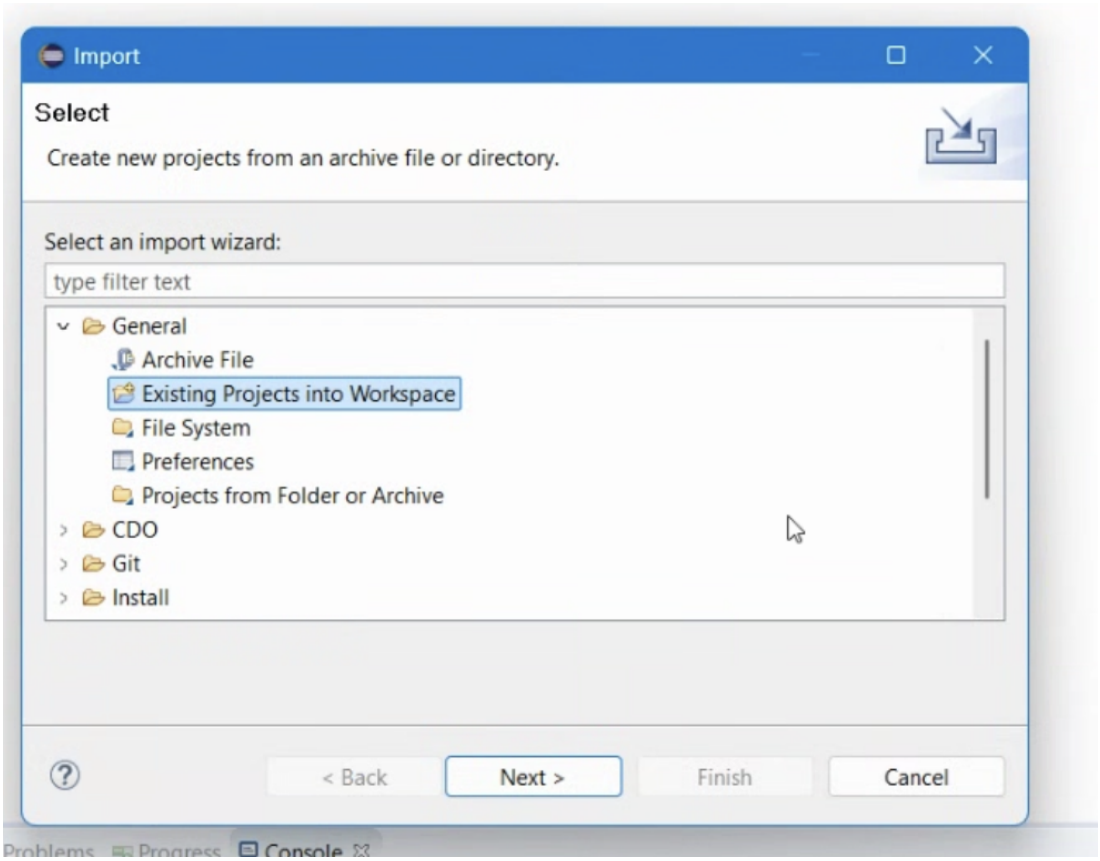
Eclipse CAPS Integration Guide

Step 1: Download the CAPS Project Files

1. Obtain the latest CAPS project archive (`CAPS.7z`) from the official repository or provided link: Download CAPS.7z
2. Save the file to a preferred location (e.g., `C:\CAPS`).
3. Extract the contents of `CAPS.7z` using a tool like WinRAR, 7-Zip, or Windows Explorer.
4. Ensure the extracted folder contains a `Packages` directory with the required Eclipse projects.

Step 2: Import Projects into Eclipse

1. Open Eclipse.
2. Navigate to `File` → `Import` .
3. Select `General` → `Existing Projects into Workspace` and click `Next`



CupCarbon Installation Guide

Step 1: Download CupCarbon

- Go to: https://cupcarbon.com/download_klines.php
- Click the download link for your platform (Windows, Mac, or Linux)

Step 2: Unzip the file

- On Windows: Right-click the downloaded ZIP file → "Extract All"
- On Mac: Double-click the ZIP file to unzip it automatically

Step 3: Run `cupcarbon.jar`

- Open the unzipped folder
- Double-click `cupcarbon.jar`
  If it doesn't open:
- Open a terminal/command prompt
- Navigate to the folder with `cupcarbon.jar`
- Run:

```
java -jar cupcarbon.jar
```

(You need Java installed; if not, download it from https://www.oracle.com/java/technologies/javase-downloads.html

PythonPDEVS Installation Guide

1. **Prerequisites**
   - Make sure you have **Python 3.x** installed.
   - You may need **pip** (Python package manager).
   - Better to create virtual environment
   - **Clone the repository**
     Open your terminal or command prompt and run:

   For Python 3.6 and earlier

   ```
   git clone https://github.com/capocchi/PythonPDEVS.git
   ```

   For Python 3.7 and later:

   ```
   git clone https://github.com/likhithkanigolla/PyPDEVS-Updated.git
   ```

2. **Navigate to the project directory**

   ```
   cd PythonPDEVS
   ```

   4. **Install PythonPDEVS**
   Run the following command to install it using `setup.py`:

   ```
   python setup.py install
   ```

   Alternatively, if you prefer installing in **development mode** (helpful if you plan to modify the source):

   ```
   python setup.py develop
   ```

PythonPDEVS Introduction

PythonPDEVS is a Python implementation of the **Parallel Discrete Event System Specification (PDEVS)** formalism. It provides tools to model and simulate systems as a combination of **atomic** and **coupled models**, enabling hierarchical and modular system design.

Atomic Models:

- The **basic building blocks** of a DEVS system.
- Define **state variables**, **internal and external transitions**, **output functions**, and **time advance** functions.
- Handle input and output events independently.
- Each atomic model simulates the behavior of a single component.

Coupled Models:

- Allow **combining multiple atomic models (or other coupled models)** into a larger system.
- Define the **connections** (couplings) between submodels.
- Provide **hierarchical composition**, enabling complex systems to be built from simpler components.
- Do **not** define behavior themselves, but manage the structure and communication of submodels.

In PythonPDEVS, you build a system by first creating **atomic models** (with logic and state changes), then integrating them into a **coupled model** to define how they interact and exchange events.

PyDEVS Motion Light System

This project implements a motion-activated lighting system using the Parallel DEVS (Discrete Event System Specification) formalism through PyDEVS.

System Overview

The system simulates a smart lighting setup where motion sensors trigger lights through a routing component. It demonstrates the principles of discrete event simulation with loosely coupled components communicating through well-defined interfaces.

File Structure and Concepts

`model.py`

This file defines the top-level coupled DEVS model (`GeneratedModel`) that composes the entire simulation. In DEVS formalism, a coupled model contains:

- Components (atomic or coupled models)
- Input and output ports
- Coupling relationships between components

The `GeneratedModel` class instantiates the motion sensor, router, and light components, then establishes the connections between them to form the complete system topology.

`motionsensor.py`

This file implements the `MotionSensor` atomic DEVS model that simulates a physical motion sensor. Key concepts:

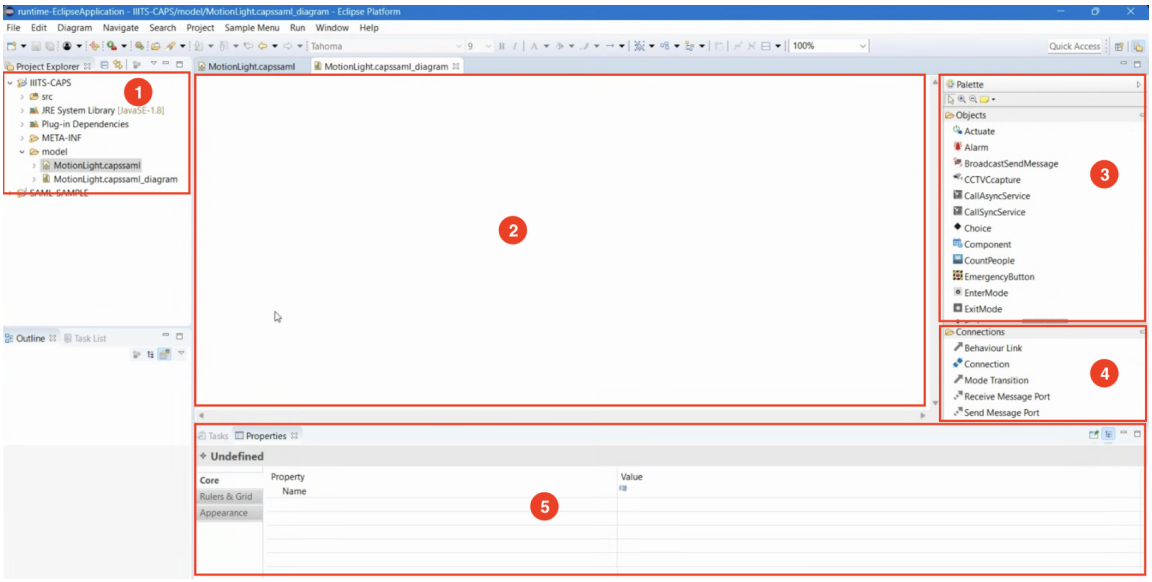
- `MotionSensorState`: Maintains the state variables for the sensor component
- `MotionSensor`: Implements the DEVS atomic model with:
- `__init__`: Sets up the sensor with its name, initial state, and ports
- `timeAdvance`: Determines when the next internal transition will occur
- `outputFnc`: Generates output events when motion is detected (randomly produces 0 or 1)
- `intTransition`: Updates the component's state after an internal transition
- `extTransition`: Handles external events (input messages from other components)

SAML (Software Architecture Modeling Language) Guide

Overview

SAML (Software Architecture Modeling Language) is used to model and simulate software architectures, particularly in IoT-based scenarios. It allows users to define components, interactions, and behaviors in a structured manner. This guide provides a step-by-step approach to using SAML for designing a temperature sensor-based automation system.

CAPS Environment in Eclipse



1. Folder Viewer (on the left)
2. Diagram Canvas (in the center)
3. Pallate(on the right)
4. Connections(on the right)
5. Properties Panel(at the bottom)

FOLDER VIEWER

This displays the file structure. It contains your SAML project, and there is a folder called ‘model’ where you can create your SAML models.

SAML for Motion Based Lighting System

Project Description

This project implements an automated motion detection and response system using a sensor-controller-actuator architecture. A **motion sensor** continuously monitors for movement within its detection range. Upon detecting motion, the sensor transmits a signal to a **controller** (or router) responsible for processing the incoming data. The controller analyzes the signal and, based on predefined logic, sends a command to a **light actuator** to turn the light on. This system enables responsive, energy-efficient lighting by automating illumination based on occupancy or movement detection, making it ideal for smart home or smart building applications.

Motion Sensor

1. Detects Motion every second
2. Sends the data to the controller/router

Controller/Router

1. Waits for the message to receive
2. Look for the available nodes/lights
3. Sends the message

Sensor/Light

1. Receives the message.
2. checks if it's 1 or 0.
3. Turn on/off the light based on the condition.

Steps to Create a SAML Project

1. Create a New SAML Project

1. Open Eclipse and go to `File` → `New` → `Project`.
2. Select `EMF` → `Empty EMF Project`.
3. Name the project and select a location (keeping the default workspace location is recommended).
4. Click `Finish`.

SAML for Temperature based window control

Project Description

We will model a simple scenario: a room equipped with a temperature sensor, an actuator connected to a window, and a server for data storage. The system functions as follows:

- The sensor detects room temperature periodically.
- If the temperature exceeds a threshold, the actuator opens the window.
- If the temperature is too low, the actuator closes the window.
- The temperature data is stored on a remote server.

Temperature Sensor

- Starts a timer to sense temperature every 10 seconds.
- Reads temperature and sends it to the server.

Server

- Receives temperature data from the sensor.
- Stores it in a database.

Controller

- Receives temperature values.
- Decides whether to open or close the window based on thresholds (25°C and 18°C).
- Sends appropriate command (`Open` or `Close`) to the actuator.

Window Actuator

- Receives commands from the controller.
- Opens/closes the window accordingly.

Steps to Create a SAML Project

1. Create a New SAML Project

1. Open Eclipse and go to `File` → `New` → `Project` .
2. Select `EMF` → `Empty EMF Project` .

Hardware Modeling Language (HWML) Guide

Introduction to HWML

The Hardware Modeling Language (HWML) is a framework designed to model the hardware components of a system. It allows users to define deployment nodes, microcontrollers, processors, memory, energy sources, and various sensors. This guide provides a detailed step-by-step process for modeling hardware using HWML.

Step 1: Creating a New HWML Model

1. Open Eclipse and navigate to `Project Explorer`.
2. Right-click on the `model` folder.
3. Select `New` → `Other`.
4. In the dialog box, search for **CAPShwml Model**.
5. Click `Next`, then provide a name for the model (ensure it ends with `.capshwml`).
6. Click `Next` again and choose **Node Specification**.
7. Click `Finish` to create the HWML model file.

Step 2: Initializing the HWML Diagram

1. Locate the newly created `.capshwml` file in the `Project Explorer`.
2. Right-click the file and select `Initialize Filesystem Diagram`.
3. Enter a name for the diagram or keep the default name.
4. Click `Finish` to generate the diagram workspace.

Defining Hardware Components

Step 3: Creating a Deployment Node

1. Open the `Palette` view on the right-hand side of Eclipse.
2. Select the **Node** element and place it onto the canvas.
3. In the `Properties` view, set the following attributes:
    - **Mac Protocol**: ZIGBEE (standard communication protocol)

Essentials

This covers a few of the essentials needed. For complete information, please refer to the **CupCarbon® User Guide Version U-One 5.1.**

CupCarbon Environment

To execute CupCarbon (jar file), use the command window and go to the directory where the jar file is located.

Then execute the following command:

```
java -jar CupCarbon.jar
```

In the case of the existence of a proxy, use the following command:

```
java -jar CupCarbon.jar proxy_host_name proxy_number_of_port
```

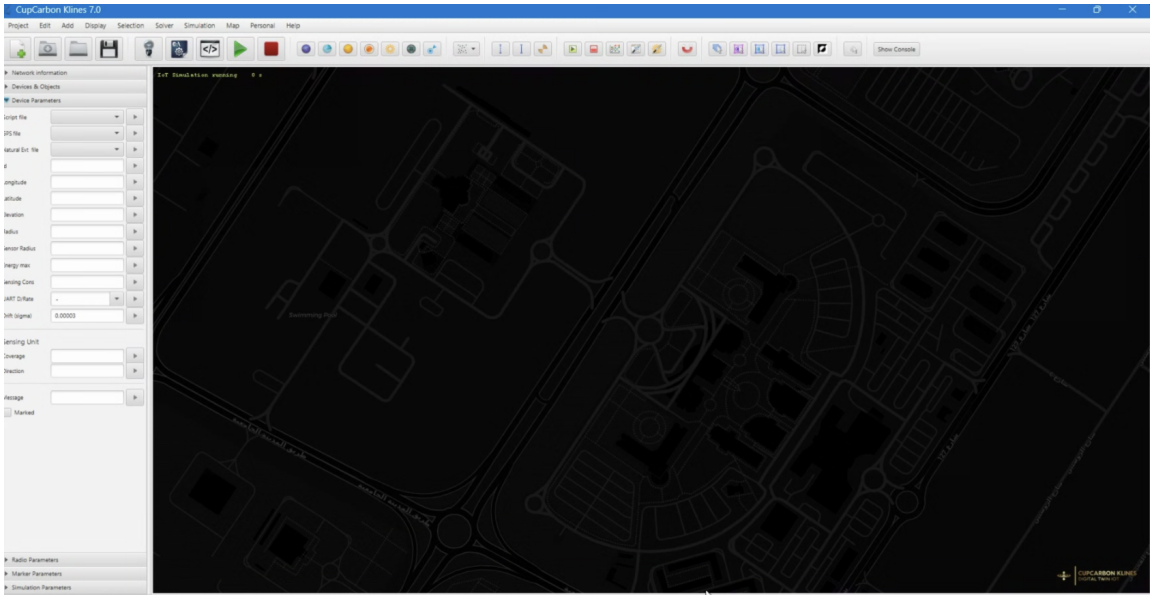
As shown in the Figure, the CupCarbon Graphical User Interface (GUI) is composed of the following five main

parts:

1. The map (in the center)
2. The menu bar (on the top)
3. The Toolbar (below the menu)
4. The parameter menu (on the left)
5. The state bar (at the bottom)
6. The console (in version 5, the console is separated from the main interface)

Simulate Motion Based Lighting System in Cup Carbon

Open the CupCarbon and create the project:



Now, go to the map and select the Notebook view:

Publications

Below are publications related to our work.

Featured Publications

1. Architecting Digital Twin for Smart City Systems: A Case Study - Likhith Kanigolla; Gaurav Pal; Karthik Vaidhyanathan; Deepak Gangadharan; Anuradha Vattem,21st International Conference on Software Architecture Workshops (ICSAW), 2024
2. Modeling and Simulating IoT Infrastructures - Philipp Zech, Karthik Vaidhyanathan, Likhith Kanigolla, Luca Rahm, and Ruth Breu, 15th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), 2025

PDF Viewer

Alternative Download Link

You can also download the PDF directly.