# *A MODULAR CO-SIMULATION APPROACH FOR URBAN ENERGY SYSTEMS*

**KUNPENG WANG, MEng, MSc**

**Thesis submitted to The University of Nottingham**

**for the degree of Doctor of Philosophy**

**October 2022**

# ABSTRACT

Cities are the main site of energy consumption, which result in approximately 71% of global $CO_2$ emissions. Therefore, energy planning in cities can play a critical role in climate change mitigation by improving the efficiency of urban energy usage. The energy characteristics of cities are complex as they involve interactions of multiple domains, such as energy resources, distribution networks, storage and demands from various consumers. Such complexity makes urban energy planning a challenging task, which requires an accurate simulation of the interactions and flows between different urban energy subsystems. Co-simulation has been adopted by a number of researchers to simulate dynamic interactions between subsystems. However, the research has been domain specific and could only be used in limited areas. There was no generic approach to tackle the interoperability challenge of a comprehensive simulation for urban energy systems.

To address such a gap, the aim of this thesis is to develop a generic and scalable urban energy co-simulation approach to comprehensively model the dynamic, complex and interactive nature of urban energy systems. This was achieved through the development of a generic and scalable urban energy co-simulation architecture and approach for the integration and orchestration of urban energy simulation tools, also called simulators, from different domains.

Nine requirements were identified through a literature review of co-simulation, its approaches, standards, middleware and simulation tools. A conceptual co-simulation architecture was proposed that can address the requirements. The architecture has a modular design with four layers. The simulator layer wraps the simulation tools; the interconnection layer enables the communication between tools programmed in different programming languages; the interoperability layer provides a mechanism for the tool composition and orchestration; and the control layer controls the overall simulation sequence and how data is exchanged.

Based on the architecture, a Co-simulation Platform for Ecological-urban (COPE) was developed. Suitable co-simulation software libraries were adopted and mapped together to fulfil the requirements of each layer of COPE to achieve the research objectives. For different simulation purposes, subsystem simulation tools from different domains could be selected and integrated into the platform. A master algorithm could then be developed

to orchestrate and synchronise the tools by controlling how the tools are run and how data are exchanged among the tools.

In order to evaluate COPE's fundamental functionality and demonstrate its application, two case studies are presented in the thesis: simulating multiple application domains for a single building and multiple (interacting) buildings respectively. From the case studies, it was observed that COPE can successfully synchronise and manage interactions between the co-simulation platform and integrated simulation tools. The simulation results are validated by comparing the results obtained from the direct coupling approach. The applicability of COPE is demonstrated by simulating energy flows in urban energy systems in a neighbourhood context. Computing performance diagnostics also showed that this functionality is achieved with modest overhead.

The layered modular co-simulation approach and COPE presented in this thesis provide a generic and scalable approach to simulating urban energy systems. It could be used for decision making to improve urban energy efficiency.

# ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisors Professor Darren Robinson, Professor Peer-olaf Siebers and Professor Yong Mao for their constant guidance, encouragement and tremendous support throughout my PhD. This has led me to establish a fundamental framework for my PhD and future research.

My thanks are due to European Commission FP7 funded CI-NERGY (Smart Cities with Sustainable Energy Systems) project for sponsoring the first three years of my research. The Marie Curie workshops around Europe and beyond helped access relevant insights and perspectives related to energy systems through lectures and informative presentations on many aspects. There have been some very interesting discussions that helped me along the way.

In the final stage of my PhD, I had to start a long and sometimes problematic recovery journey after my aneurysm rupture operation. During the most difficult time of my life, the support of my family and my mentors Prof. Yong Mao and Dr Neal Wade saved me from a jungle in which I blamed and complained about myself quite often. Without their support and encouragement, I could never overcome the hardship and depression and complete the case studies and thesis writing. I am grateful for Dr Neal Wade offered me the job at Newcastle University. The job not only solved my financial and visa pressure, more importantly, it helped me rebuild my confidence, recover physically, mentally and emotionally. To my mom and sisters, thank you for your encouragement, help and prayers. To my wife and my son, I owe you for your caring, patience, gentleness and forbearance. It was love saved me and pulled me out of the darkness and I truly thank you all. Over this journey, I especially thank God for the unconditional love, for the experience I had, which helped me see my weakness and value of life.

This thesis is specially dedicated to my beloved father, Chuncai WANG, who has always been my mental model.

# CONTENTS

# LIST OF TABLES

# List of Figures

# List of Abbreviations and Acronyms

| | |
|---|---|
| ADE | Application Domain Extension |
| API | Application Programming Interface |
| DSM | Demand-Side-Management |
| DLL | Dynamic-Link Library |
| FMI | Functional Mock-Up Interface |
| FMU | Functional Mock-Up Unit |
| HLA | High Level Architecture |
| HPC | High Performance Computing |
| IEA | International Energy Agency |
| RTI | Run-Time Infrastructure |
| No-MASS | Nottingham Multi-Agent Stochastic Simulation |
| PV | Photovoltiac |
| XML | Extensible Markup Language |

# LIST OF APPENDICES

# 1 INTRODUCTION

## 1.1 Motivation and research context

### 1.1.1 Motivation

Compared to 1950 when only 30% of the world's population lived in urban areas, nowadays, 55% of the world's population live in urban areas. This is predicted to increase to over 68% by 2050 (UNDESA 2018). Cities are the main site of energy consumption in many industrial and high population countries. The International Energy Agency (IEA) estimates that more than two-thirds of primary energy in the world is consumed in cities (IEA 2021). This energy consumption correspondingly results in approximately 71% of global $CO_2$ emissions (IEA 2021). Therefore, by improving the efficiency of urban energy usage and using more low-carbon energy resources, cities can play a critical role in climate change mitigation.

To improve the urban energy usage efficiency and promote low-carbon energy resources requires effective planning and decision making. The energy characteristics of cities are complex as they involve interactions of multiple domains. For example, the dynamic balance of energy supply and demand in urban area is influenced by many factors involving the availability and capacity of diverse energy resources (fossil fuel, hydropower, biomass, solar, wind, and other renewable resources), combination of multiple energy distribution networks (electricity, gas, and heating), and a variety of types of energy storage (Pump Hydro Storage (PHS), Thermal Energy Storage (TES), batteries, and Adiabatic Compressed Air Energy Storage (A-CAES)). This is further

complicated by distinctive energy demand behaviours from various consumers (citizens, private and public companies, local authority, government bodies, and manufacturers/traders). In addition, different evaluation criteria (economic, technical, political, environmental and social) are also adopted by various stakeholders, which might create conflicts of interest in urban energy planning and decision-making process.

Such complexity makes urban energy planning a challenging task, which requires not only comprehensive understanding of different energy resources, distribution networks, energy storage systems and various consumers, but also an accurate simulation that captures the complex interaction among different urban energy subsystems. To do this, urban energy modelling tools that can analyse urban energy usage quantitatively are needed to assist decision making either individually or collectively for improving urban energy efficiency and sustainability.

### 1.1.2 Research context

Over the last couple of decades, a number of high quality simulation tools have been developed to simulate dynamic behaviour of urban energy systems. These address different aspects of urban energy planning with different levels of granularity (Robinson, Haldi et al. 2009, Connolly, Lund et al. 2010, Manfren, Caputo et al. 2011, Keirstead, Jennings et al. 2012, Mohammadi, de Vries et al. 2013, Allegrini, Orehounig et al. 2015, Reinhart and Cerezo Davila 2016, Abbasabadi and Ashayeri 2019, Li, Wang et al. 2020, Deng, Chen et al. 2022).

Top-down and bottom-up modelling techniques are two types of approaches used in developing these tools. The top-down modelling approach works at a macro level, which simulates urban energy systems in an aggregated way with large spatial and temporal resolutions based on historical data of energy use patterns. Top-down models describe the general characteristics of energy systems, rather than the explicit energy profile of each individual component (Swan and Ugursal 2009, Ali, Shamsi et al. 2021). They are suitable for a broad and aggregated level large-scale analysis (Ali, Shamsi et al. 2021). However, these models are less suitable to deal with the high degree of technological detail for energy analysis of a specific neighbourhood and it is less suitable for examining potential impacts of more technology-specific policies influencing energy profile (Kavgic 2013, Ali, Shamsi et al. 2021). The bottom-up

approach uses micro-simulation models, which are based on extensive databases related to empirical data describing the dynamic behaviour of each component. It could be used to calculate the energy profile of individual components of the system and then extrapolate these results to represent energy profile at an urban and regional level (Shorrock and Dunster 1997, Robinson, Haldi et al. 2009, Mohammadi, de Vries et al. 2013, Hedegaard, Kristensen et al. 2019). Simulation tools using the bottom-up approach are also termed micro-simulation tools. In comparison to the top-down approach, the bottom-up modelling approach can provide a detailed energy profile of end-use individually, which makes it more suitable to investigate detailed urban energy design choices.

EnergyPlus and CitySim are two micro-simulation tools using the bottom-up approach. They use micro-simulation models focusing on energy modelling of buildings at different scales. EnergyPlus is a building energy simulation tool used by engineers, architects, and researchers to model not only energy consumption for heating, cooling, ventilation, lighting and plug loads, but also water use at single-building level. Some researchers used EnergyPlus as an urban modelling interface for simulating groups of buildings where multiple EnergyPlus instances were created to simulate individual buildings one by one (Reinhart, Dogan et al. 2013, Cerezo Davila, Reinhart et al. 2016, Natanian, Aleksandrowicz et al. 2019, Li, Wang et al. 2020, Buckley, Mills et al. 2021, Deng, Chen et al. 2022). CitySim simulates building energy demand considering the stochastic nature of occupants' presence and behaviour, as well as a range of commonly used heating, ventilation and air conditioning systems. Unlike EnergyPlus, CitySim can simulate a cluster of buildings simultaneously within one instance. The analysis scale of CitySim may vary from a neighbourhood of just a few buildings and a district of several hundred to an entire city with tens of thousands of buildings (Robinson 2011, Thomas, Miller et al. 2014).

Apart from these specialised tools for building energy simulation, there are micro-simulation tools that are used to simulate electrical grids or thermal networks. Examples of the tools for electrical grids simulation are PowerWorld Simulator (DIgSILENT 2019), IPSA (TNEI 2019), PowerFactory (DIgSILENT 2019), and PSCAD (PSCAD 2019), which are widely used to simulate power generation, transmission and distribution (Zhou and Bialek 2005, Wade, Taylor et al. 2010, Palensky, Widl et al. 2014, Neaimeh, Wardle et al. 2015, Aprilia, Meng et al. 2019).

For district thermal networks, tools like NetSim (Brange, Englund et al. 2016) and Termis (Vesterlund, Toffolo et al. 2016) can be used to evaluate the network performances and provide design, operation and optimisation suggestions based on accurate thermal network calculations considering parameters like pressure, velocity and temperature (Dalla Rosa, Boulter et al. 2012, Tol and Svendsen 2012, Brand, Calvén et al. 2014, Tunzi, Boukhanouf et al. 2018).

In addition, there are also specialised tools providing availability and economic analysis of renewable energy resources. For example, the r.sun module within the Geographical Resources Analysis Support System (GRASS) can be used to compute solar radiation on building roofs in a complex landscape area and therefore to identify where and whether a roof is suitable for installing solar photovoltaic (PV) panels (Agugiaro, Nex et al. 2012). RETScreen is a decision support tool which can be used to evaluate the technical and economic feasibility of renewable technologies, such as PV systems (Mirzahosseini and Taheri 2012).

The energy simulation tools described above were developed for specific and distinct purposes. Hence, they are able to model, simulate and help to understand specific problems of particular domains, such as building, electricity network, district heating network, etc. However, they are not able to model and simulate holistic behaviour of urban energy systems with subsystems across the borders of their specific domain. Therefore, integrated modelling for urban energy systems has been proposed (Grubler, Bai et al. 2012, Keirstead, Jennings et al. 2012, Koppelaar, Kunz et al. 2013). The objective is to achieve a comprehensive simulation of urban energy systems not only providing detailed simulation of individual subsystems for specific domains, such as building and distribution network, but also their dynamic interactions, which could address different aspects of urban energy planning with different levels of granularity. However, simulating such complex interactions of urban energy systems faces a number of challenges. Of those, data quality and uncertainty, subsystem model complexity and simulation interoperability, e.g. insight into the internal process of models of a complex system, integration and orchestration of diverse urban energy subsystem simulation tools with different granularities, are the key challenges that need to be tackled.

In terms of data processing, data models representing urban concepts and their interactions have been adopted by some researchers, like van Dam and Keirstead,

(2010), Nouvel et al., (2015) and (Agugiaro, Benner et al. 2018), to solve the technological challenges related to data processing between different simulation tools involved in urban energy simulations.

A data model is an abstraction of a system in the real world. It employs the grammar, vocabulary and content that describe all kinds of information of the system stored in one format or another. Within the model, the grammar defines the relationships between individual objects in the system; the vocabulary defines the terminology to be used to attribute these objects; and content defines what is to be included in the system (Peng and Law 2010). In essence, a data model is a representation of the data and their relationship and provides a conceptual or implementation view of the data (Hamilton, Wang et al. 2005, Diba, Batoulis et al. 2020).

In simulating urban energy systems, multiple domains with different data sources and models need to be considered, which increases the systemic complexity and heterogeneity leading to potential data interoperability issues. It is crucial that the system developers and users share a common understanding of the complex concepts in the domains in terms of common vocabularies and data models in order to be able to communicate effectively (Howell, Rezgui et al. 2017). Obviously, developers could benefit from the interfaces provided by a common data model in order to access disparate data sources. SynCity UES (Keirstead, Samsatli et al. 2010) and CityGML (Kruger and Kolbe 2012) are two data models that can be used to assist the development and sharing of urban energy models, and thus facilitate model integration.

The SynCity UES data model was initially developed for the SynCity project aiming to provide consistent class definitions of major objects within an urban energy system (van Dam and Keirstead 2010). It was designed as a library of domain-specific components consisting of a number of object classes that describe the main elements of an urban energy system and specific instances of these classes (Keirstead and Van Dam 2010). This data model was only used to develop the SynCity toolkit and not released publicly.

In contrast, CityGML is an international standard that defines the physical layout of a city according to its semantics, geometry, topology and appearance. It’s an information and data model for semantic city models, which is intended to support

simulation, urban data mining, facility management and thematic inquiries (CityGML 2015).

However, currently CityGML does not define all energy-related attributes and features of buildings in a systematic and standard way. To address the requirement for building energy simulation based on CityGML, a growing international consortium of urban energy simulation developers has started to develop an Energy Application Domain Extension (ADE) for CityGML (Nouvel, Kaden et al. 2015). ADE is the mechanism that can be used to extend CityGML classes with application-specific objects and their attributes that are not explicitly represented in CityGML. The current version of Energy ADE is 2.0 (Schildt, Behm et al. 2021). With Energy ADE support, CityGML is extended in a standardised way by representing, storing and exchanging energy-related features and attributes that are important and necessary for urban energy models.

The CityGML standard together with its Utility Network ADE and Energy ADE has the potential to be a common data model for urban energy systems. However, CityGML was designed as a data model for 3D city modelling initially and Utility Network ADE and Energy ADE were newly developed. Therefore, researchers now often use CityGML as a 3D data model; potentially also applying this to evaluate solar energy or for urban heating energy consumption estimation from building level to city level through calculations by using the concept of indicators and indexes (Bahu, Koch et al. 2013, Nouvel, Zirak et al. 2014, Wieland, Nichersu et al. 2015, Agugiaro 2016, Ledoux, Arroyo Ohori et al. 2019).

The most popular urban energy simulation tools, like EnergyPlus, CitySim, IPSA, PowerWorld Simulator, PowerFactory, and PSCAD etc., were not developed based on a common data model. Therefore, to make these tools support a specific data representation standard such as CityGML, together with Energy ADE, tool developers have to either redesign their data model or write translators. It means the fundamental data structure has to be changed, which is a complex task. As a result, developers of these mature domain specific tools lack motivation on supporting it. Therefore, on the ground of feasibility, data processing has been dismissed in the research presented in this thesis. Thus, the focus of the research presented in this thesis is on tackling the simulation interoperability challenge. The difficulty of it is to orchestrate the execution of all integrated subsystems and make sure the on-time data exchange

between subsystems is achieved (RJ, A et al. 2014, Gomes, Thule et al. 2017). Currently, holistic approach and co-simulation (cooperative simulation) approach are the two main strategies to tackle the simulation interoperability challenge of the comprehensive simulation for urban energy systems.

The holistic approach aims to develop one multi-disciplinary tool to offer a complete understanding of energy demand, supply and distribution in cities. Holistic simulation tools of urban energy systems typically originate from a particular domain and later have been extended to include other parts of urban energy systems to provide a holistic view of the systems. SynCity (Keirstead et al., 2010) belongs to this category. It is a hierarchical modelling framework for integrated assessment and optimisation of urban energy systems. It aims to bring together different city representations such as layout, transport, resource flows and energy networks, so that urban energy use at different stages of a city's design can be examined within a single platform (Van Dam 2009, Gea 2012). Currently, SynCity links four submodels, i.e. the urban layout model, the urban agent-based transport-land use (ABMS) model, the urban resource-technology network (RTN) optimisation model and the energy service network model, into one toolkit. However, it does not integrate urban energy supply network models, energy consumption and supply models. As an initial effort to develop urban energy holistic simulation tools, SynCity is promising but it was a prototype built with a limited number of proof-of-concept models (Van Dam 2009, Keirstead, Samsatli et al. 2010, Grubler, Bai et al. 2012). More importantly, when more models need to be integrated, the solvers of SynCity need to be modified significantly to adapt to the extension. This makes it hard to integrate more proof-of-concept models or fully developed urban energy simulation models developed by other researchers.

A comprehensive simulation of urban energy systems requires the simulation of multiple urban subsystems, i.e. onsite energy production, buildings and distribution networks, etc (Koppelaar, Kunz et al. 2013). Without detailed onsite energy production, building and network models, the energy flows in urban energy system cannot be captured accurately. The holistic approach is hardly a good choice to fulfil such requirement. This is because a single holistic tool, such as SynCity, is unlikely to provide all functionalities and models required for a comprehensive simulation of urban energy systems, since the developers of the holistic tools are not necessarily experts in the domains that their tools seek to address. In view of the limitations of the

holistic approach, the co-simulation approach provides a solution to this dilemma by enabling the best available simulation tools from their respective domains and integrating them into one system.

The co-simulation approach tackles the challenge of comprehensive modelling of urban energy systems, not by providing a single simulation that addresses a number of issues, but by coupling existing domain-specific energy simulation tools, each with its own speciality and was developed by corresponding domain experts with accumulated years of research and practical experience. In this approach, intermediate results (variables, status information) can be exchanged between coupled modelling tools during simulation where data exchange is restricted to discrete communication points. Between these communication points the subsystems are solved independently (Bastian, Clauß et al. 2011). Such approach enables a dynamic and comprehensive simulation of urban energy systems.

The hard-coded co-simulation and standard based co-simulation are two possible ways to provide functionality like aggregation and enable tools with different granularities to be coupled. The hard-coded co-simulation approach develops a monolithic mega tool, which integrates a group of simulation tools directly with each other without using any co-simulation standard/middleware. Such an approach is straightforward and requires less development effort. However, the co-simulation system developed by using this approach is typically only fit for one simulation purpose, hence not reusable and hard to maintain. In contrast, standard based co-simulation has no such limitation. It uses a co-simulation standard/middleware to develop a co-simulation system, which is reusable and extendable. Although this approach requires more effort to develop, nowadays most researchers prefer this approach of co-simulation system due to its extendibility and flexibility. A number of researchers adopted the co-simulation approach in their work. Some researchers focused on smart grid simulation (Rohjans, Lehnhoff et al. 2013, Stifter, Widl et al. 2013, Kosek, Lünsdorf et al. 2014, Lévesque, Béchet et al. 2014, Neema, Gohl et al. 2014, Palensky, Widl et al. 2014, Strasser, Stifter et al. 2014, Lehnhoff, Nannen et al. 2015, Rotger-Griful, Chatzivasileiadis et al. 2016, Falcone and Garro 2019). Some of them, like Lévesque, Béchet et al. (2014), Neema, Gohl et al. (2014) and Falcone and Garro (2019), used both co-simulation standards such as Functional Mockup Interface (FMI) and High Level Architecture (HLA). FMI is a tool independent industry standard to support both

model exchange and co-simulation of dynamic models (Blochwitz, Otter et al. 2011). HLA is an industry standard for distributed modelling and simulation (Dahmann, Fujimoto et al. 1997). There are also some researchers didn't use either FMI or HLA, but chose to use mosaik (Schutte, Scherfke et al. 2011), a smart grid co-simulation middleware (Rohjans, Lehnhoff et al. 2013, Kosek, Lünsdorf et al. 2014, Lehnhoff, Nannen et al. 2015, Steinbrink, Blank-Babazadeh et al. 2019), or Ptolemy II (Ptolemaeus 2014), a middleware for design and analysis of heterogeneous systems (Palensky, Widl et al. 2014, Rotger-Griful, Chatzivasileiadis et al. 2016), or just use one tool in a tool-sets (Stifter, Widl et al. 2013, Strasser, Stifter et al. 2014) as the master to couple other simulation tools in the tool-sets in a hard-coded way. In addition to the smart grid simulation, there are some researchers, Wetter (2011), Pang, Wetter et al. (2012), Heinzl, Kastner et al. (2014), Nouidui (2014), Thomas, Miller et al. (2014), Yao (2014) Raad, Reinbold et al. (2015), Pang, Nouidui et al. (2016) and Chapman, Siebers et al. (2018), that used co-simulation approach to integrate simulation tools in the building simulation domain. Many of them used the Building Controls Virtual Test Bed (BCVTB) as co-simulation middleware in their research (Wetter 2011, Pang, Wetter et al. 2012, Heinzl, Kastner et al. 2014, Yao 2014). Others either combined BCVTB together with FMI (Nouidui 2014, Pang, Nouidui et al. 2016) or only used FMI (Thomas, Miller et al. 2014, Raad, Reinbold et al. 2015, Chapman, Siebers et al. 2018).

The co-simulation approach adopted by the above mentioned researchers does offer solutions to couple existing domain-specific tools to provide a detailed simulation of not only individual subsystems but also their dynamic interactions. However, it is notable that the approaches adopted by above researchers have their own limitations from different perspectives. For example, some of them, Stifter, Widl et al. (2013) and Strasser, Stifter et al. (2014), presented the simulation environment by using hard-coded co-simulation to solve the specific problem they are facing. Some of them, use standard-based co-simulation but with technique like FMI import which only supports coupling of two simulation tools (Thomas, Miller et al. 2014, Raad, Reinbold et al. 2015, Chapman, Siebers et al. 2018). Furthermore, they all are very domain focused. It is observed that none of the above mentioned researchers aimed to develop a generic model integration approach. Their modelling approaches targeted only a subset of relevant domains, such as smart grids, buildings, etc. and can only solve particular

problems with limited functions in the targeted domain. Therefore, they are not able to model and simulate urban energy systems with subsystems across the borders of traditional engineering domains.

To overcome the shortcomings of existing urban energy simulation tools and integrated modelling methodologies, a generic model integration approach is needed to tackle the simulation interoperability challenge of a comprehensive simulation for urban energy systems. The approach needs to provide functionality for integrating and managing the interaction of diverse urban energy subsystem simulation tools with different granularities in order to comprehensively simulate multiple domains of urban energy systems.

The proposed approach could be used by urban energy planners and stakeholders who collaborate with architects, engineers and researchers closely to propose appropriate and confident solutions to improve the efficiency of urban energy usage and use more low-carbon energy resources. The collaboration requires urban energy planners and stakeholders to propose feasible energy efficiency targets from political, economic, and technological perspectives. Correspondingly, the engineers, architects, and researchers could be actively involved in all stages of the proposal, evaluation, validation and implementation procedure.

## 1.2 Research aim and objectives

Following the gap identified in the previous section, the aim of the research presented in this thesis is to develop a generic and scalable urban energy co-simulation approach for the integration of urban energy simulation tools, in order to comprehensively model the dynamic, complex and interactive nature of urban energy systems.

By utilising well-known domain specific urban energy simulation tools, the approach will make the interoperability and reuse of existing implementations possible, and it can also be easily used by other researchers to integrate simulation tools for their own purposes. As a result, a flexible and sustainable approach to meet various simulation requirements can be achieved. The objectives of the research are summarised as follows:

Obj. I: Identify requirements for the generic and scalable urban energy co-simulation.

Obj. II: Design a conceptual co-simulation architecture that will be able to integrate urban energy simulation tools from different domains. The approach will address the identified requirements.

Obj. III: Develop an urban energy co-simulation platform based on the conceptual architecture. Explicit process to integrate well known domain specific urban energy simulation tools with different levels of granularities will also be presented.

Obj. IV: Evaluate the approach and the platform through use cases to demonstrate synchronisation and interaction between the urban energy co-simulation platform and coupled co-simulation components.

## 1.3 Research methods

In order to achieve the above-mentioned research objectives, a more in-depth review of co-simulation needs to be conducted. The purpose of the review is not only to identify the co-simulation requirements, but also the technologies that could be adopted for the co-simulation approach.

Following the review, a conceptual urban energy co-simulation architecture will be designed through a comprehensive study of co-simulation, related standards and middleware, and urban energy simulation tools. The architecture will need to address all requirements identified from the review.

Based on the architecture, a platform will be developed in order to implement the co-simulation approach. To evaluate the approach and the platform, it is planned to run the platform with two use cases from single-building level to neighbourhood scale.

The research approach and research methods that will be adopted in the research are presented in Figure 1.1.

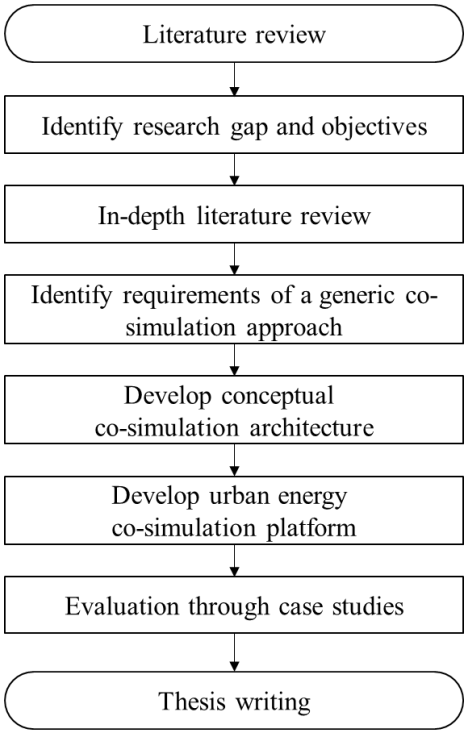



**Figure 1.1: Research approach**

## 1.4 Thesis overview

The thesis consists of six chapters that are corresponding to the research approach. Following the Introduction (**Chapter 1**), which gives an overview of the research background, research gap, and research aim and objectives, the remainder of the thesis is structured as below:

**Chapter 2**: Provides a more detailed review of urban energy simulation tools, introduction of co-simulation, co-simulation standards, co-simulation middleware and co-simulation software architecture requirement. Through the review, requirements of a generic co-simulation approach will be identified in this chapter. (Obj. I)

**Chapter 3**: The proof-of-concept urban energy conceptual co-simulation architecture is proposed to address the requirements identified in Chapter 2. (Obj. II)

**Chapter 4**: Presents the development of each layer of the standard based urban energy co-simulation platform based on the conceptual architecture and demonstrates steps involved in the execution of urban energy simulation using the platform. (Obj. III)

**Chapter 5**: Evaluate the research through case studies from a single building to neighbourhood scale co-simulation, which tackles simulation interoperability challenge of the comprehensive simulation for urban energy systems. Thorough result

analysis of the use cases to demonstrate the platform is capable of capturing the dynamic interaction between various aspects of urban energy subsystems. (Obj. IV)

**Chapter 6**: Concludes the key findings of the research and recommends how the work could be extended in the future.

How the thesis structure correlates with the research approach and the research objectives are illustrated in Figure 1.2.

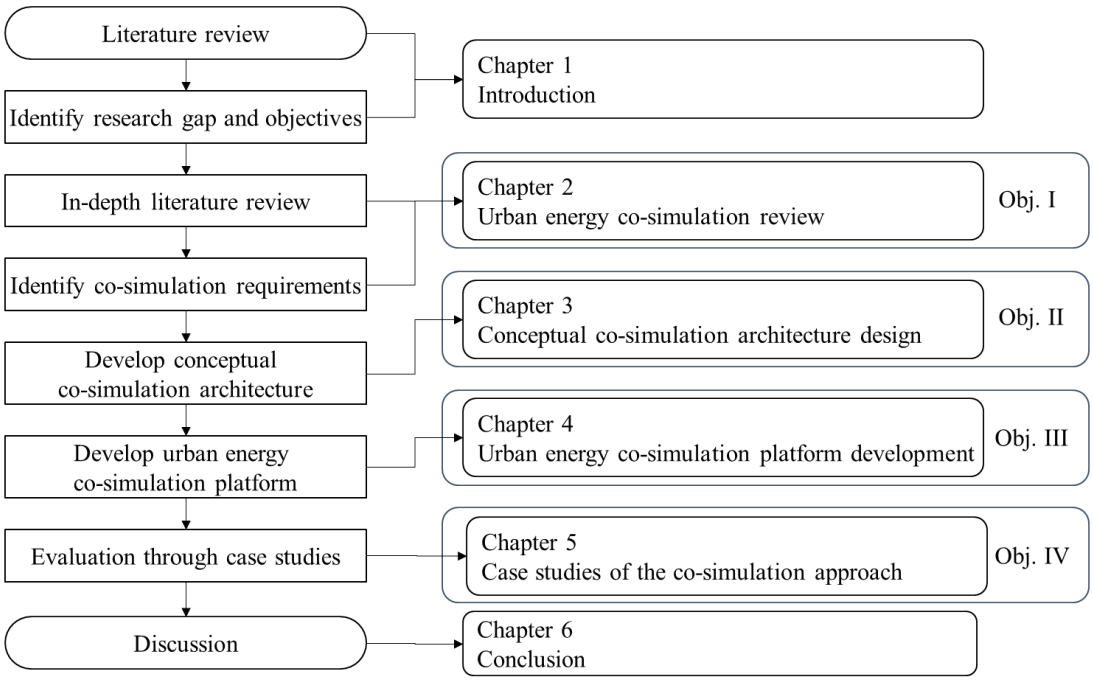



**Figure 1.2: Thesis structure**

# 2 URBAN ENERGY CO-SIMULATION REVIEW

In this chapter, a number of simulation tools with different functionalities for urban energy simulation are presented and analysed. Co-simulation as well as its approaches, standards and middleware are reviewed and presented. How the approach, standard and middleware were chosen in order to build a conceptual architecture for the urban energy co-simulation is described. Through the review, nine requirements of generic urban energy systems co-simulation are identified.

## 2.1 Urban energy simulation tools

The energy characteristics of cities are complex as they involve interactions of multiple domains, e.g. energy supplies, energy distribution networks, energy storage and energy demand from various consumers. Each domain has their tools to simulate the subsystem behaviours.

To achieve a specific purpose of urban energy systems simulation, it is required to integrate the most suitable existing simulation tools with different granularities from different domains. There are a number of simulation tools with different functionalities in the field of urban energy modelling. To simulate urban energy systems, suitable simulation tools could be chosen to suit specific simulation purpose. Different combinations of tools could serve different simulation purposes. Therefore, it is essential to focus on co-simulation of some domain specific simulation tools to make sure this co-simulation expertise and experience be capable of being leveraged into a wide range of simulations of more complex urban energy systems eventually.

In this section, a few available simulation tools that have been developed for building simulation and stochastic simulation are presented and analysed. The attribute analysis of the simulation tools helps to define requirements for simulation tools integration in the urban energy co-simulation approach.

### 2.1.1 Building energy modelling and simulation

Building energy simulation tools are increasingly used for analysis of energy consumption of one or multiple buildings. They are often used for design and optimise buildings. A number of tools have been developed for energy simulation of buildings, such as BLAST, CitySim, DOE-2, EnergyPlus, and ESP-r.

*EnergyPlus* was funded by the U.S. Department of Energy Building Technologies Office. It's a prominent detailed whole-building energy simulation tool that combines the best features of the BLAST and DOE–2 programs along with many new capabilities, which was converted from Fortran to C++ since version 8.0 (Crawley, Lawrie et al. 2004, New and Adams 2018). It is well-documented, open-source and cross-platform, and has been extensively tested by engineers and architects (Crawley, Lawrie et al. 2001, Management Association 2016, Vadiee, Dodoo et al. 2018). It can simulate energy consumption for heating, cooling, ventilation and lighting. It can also calculate thermal zone conditions, heat and mass transfer and illumination in buildings at sub-hourly intervals, accepting user-definable time steps (a minimum time step of 1 minute), taking the ambient weather conditions into account (Crawley, Lawrie et al. 2001, Gervásio, Santos et al. 2010). EnergyPlus determines zone thermal loads and the energy required from the Heating, Ventilation and Air-Conditioning (HVAC) systems to maintain thermal control set points (Zhu 2012, Dols, Emmerich et al. 2016).

The ESP-r tool is another building simulation tool for the simulation of heat, inter-zone air flow, intra-zone air movement, electrical power flow, HVAC systems and lighting performance of buildings (Clarke 2013). The time simulation of the building with ESP-r simulation tool can vary in a range from one minute to one hour. It is quite powerful (Sousa 2012). However, the program is quite complicated and relatively poorly documented which requires great knowledge and expertise from its users and requires a long learning process (Sousa 2012).*CitySim* is an urban energy simulation tool developed at the Solar Energy and Building Physics Laboratory of EPFL. It was programmed in C++ and it focuses on simulation of energy demand of buildings ranging from a few to thousands (Robinson, Haldi et al. 2009, Robinson 2011). CitySim includes a radiation model based on the simplified radiosity algorithm (SRA) to compute the irradiation incident on each surface of the building, direct from the sun, diffuse from the sky and reflected by other surfaces (Robinson and Stone 2004). The thermal model of CitySim is a simplified resistor-capacitor network as an analogy for

the thermal representation of building behaviour (Kampf and Robinson 2007, Robinson 2011). When using CitySim to simulate a building, HVAC systems are modelled using a set of equations that calculates the energy demands of the HVAC systems to reach the comfort zone due to changes in the enthalpy of the supplied air from its intake to its room supply. Considering the interactions within the built environment, inter-reflections between building surfaces and shading from other obstacles, CitySim can simulate the heating or cooling energy required to maintain predefined indoor temperature conditions within an hourly time step (Robinson 2011, Koppelaar, Kunz et al. 2013, Page, Basciotti et al. 2013, Perez 2014, Mauree, Coccolo et al. 2017).

### 2.1.2 Occupant simulation

Occupants interact actively with their built environment. Their behaviours affect building performance, at the same time, are affected by building design and indoor environmental conditions (Laaroussi, Bahrar et al. 2020). To improve building design processes and operating strategies, tools simulating human-building interactions and occupant behaviour have gained significant interest in recent years (Azar, O'Brien et al. 2020). Among them, a few examples are Buildings.Occupants obFMU, No-MASS and Occupancy Simulator (Hong, Sun et al. 2016, Luo, Lam et al. 2017, Chapman, Siebers et al. 2018, Wang, Hong et al. 2019).

Programmed in C++, *Nottingham Multi Agent Stochastic Simulation (No-MASS)* is a multi-agent simulation tool that generates synthetic populations of buildings' occupants and their energy-related behaviours (e.g. interactions with the building envelope, lights and electrical appliances). The stochastic models of No-MASS can be set as a seed value for controlling the randomness of the simulations. No-MASS can be coupled with a building simulation tool, such as EnergyPlus or CitySim. This coupling allows for simulated occupants to make changes within the simulated building environment and to receive responses arising from the effects of their interactions. Due to the window, shading and location/presence models, a sub-hourly time step is recommended for using No-MASS. Longer time steps may overestimate the implication of the occupant interactions (Chapman, Siebers et al. 2018).

### 2.1.3 Simulation tools summary

The above simulation tools from different subdomains of urban energy systems have different characteristics. The features of these simulation tools are summarised in Table 2.1.

**Table 2.1: A summary of urban energy simulation tools**

| Simulation tool | Function | Simulation time progression | Temporal resolution | Implementation programming language |
|---|---|---|---|---|
| EnergyPlus | building energy simulation | discrete time | sub hourly (5 minute, 15 minute, etc.) | FORTRAN / C++ |
| CitySim | building energy simulation | discrete time | hourly | C++ |
| No-MASS | occupant simulation | discrete time | sub hourly (5 minute, 15 minute, etc.) | C++ |

As shown in the above table, the available simulation tools are all based on discrete-time simulation. This requires the architecture of the urban energy co-simulation to provide functionality to integrate simulation tools in a discrete-time simulation manner with a fixed step size. Hence, requirement ***R1. Discrete-time simulator integration support*** is identified. In addition, the temporal resolutions of these simulation tools are different. Therefore, another requirement is that simulation tools with different temporal resolution need to be composed together (Requirement ***R2. Different temporal resolution support***). As the analysed simulation tools are developed in different programming languages, e.g. C++ and FORTRAN, the urban energy co-simulation architecture needs to support the integration of simulators implemented in different programming languages. This becomes requirement ***R3. Simulation tools in various programming language support***. If the urban energy co-

simulation approach can meet the above mentioned requirements, then a broader range of simulation tools with similar attributes can be integrated.

## 2.2 Co-simulation

Co-simulation is an approach for joint simulation of a system or a coupled problem by composing multiple subsystem simulation tools, also called simulators, to achieve a global simulation of multiple subsystems that have interactions among them (Arnold 2004, Trčka, Hensen et al. 2010, Steinbrink, Lehnhoff et al. 2017, Gomes, Thule et al. 2018, Taveres-Cachat, Favoino et al. 2021). In this approach, coupled tools have their own simulation model and runtime environment and the simulation is conducted on the subsystem level, with communications among involved subsystems.

Subsystem simulators coupled in a co-simulation environment are software tools in various domains that are capable of simulating dynamical system behaviour under specified conditions. The subsystem could be a building system, an energy transmission or an energy storage system. As shown in Figure 2.1, a subsystem simulator normally contains a model of the system and a solver. The model is an idealised and simplified representation of the system and the solver performs computations based on the model and the input variables with sufficient numerical accuracy (Siegfried 2014, Palensky, Meer et al. 2017).

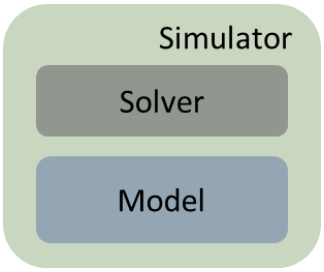



**Figure 2.1: Composition of a simulator**

A co-simulation environment contains at least two coupled subsystem simulators with data exchange between them to achieve simulation results. In most cases, it has a master algorithm that orchestrates the entire co-simulation. During a co-simulation process, the master algorithm controls the data exchange, coordination and synchronisation between the coupled simulators.

The data exchange between the coupled simulators is accomplished at user-defined communication points. The time interval between two communication points is a

global time step (e.g. a macro time step). Within each global time step, each simulator experiences a transition of the system model, which can be carried out at a series of local time steps (e.g. micro time steps).

As shown in Figure 2.2, there are two types of co-simulation from the coupling point of view. One is strong, also called tight coupling co-simulation, where the coupled subsystem simulators are solved by one solver. The other is weak, also called loose coupling co-simulation, where each of the coupled subsystem simulators has its own solver and model. Each simulator is solved independently from each other by its own solver. A co-simulation can be implemented by using either type (Hensen 1995, De Sturler, Hoeflinger et al. 2001, Vaculin, Kruger et al. 2004).

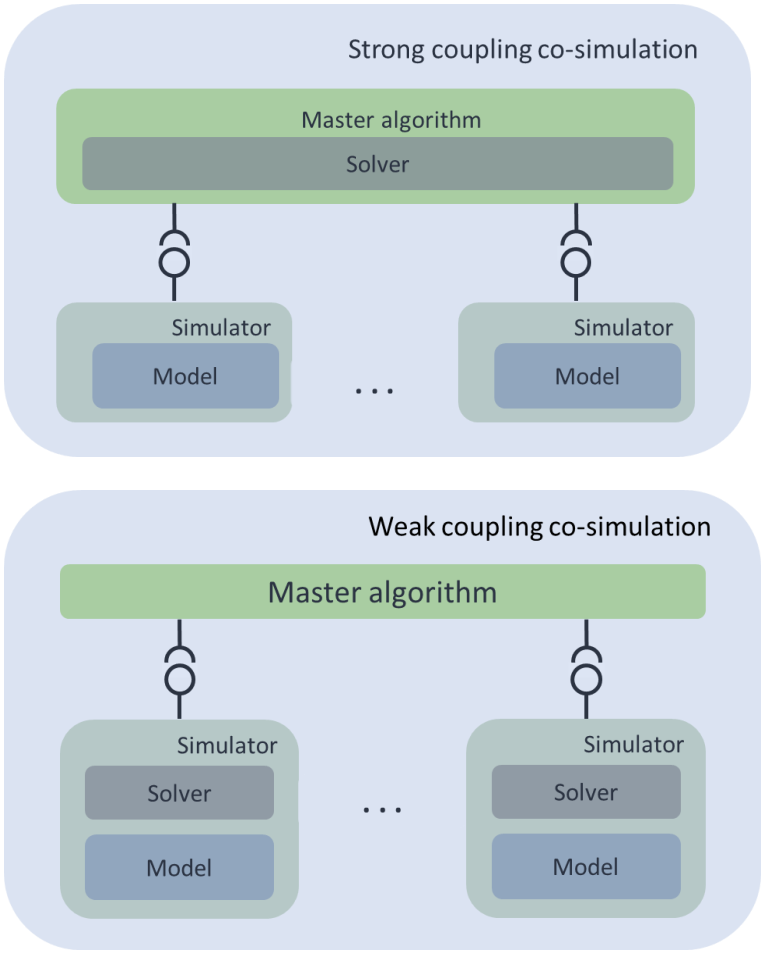



**Figure 2.2: Co-simulation environment**

The weak coupling scheme can be further categorised into Jacobi pattern and Gauss-Seidel pattern based on communication patterns between subsystems (Busch and Schweizer 2010, Andersson 2013). Figure 2.3 presents an overview of the co-simulation strategy classification.

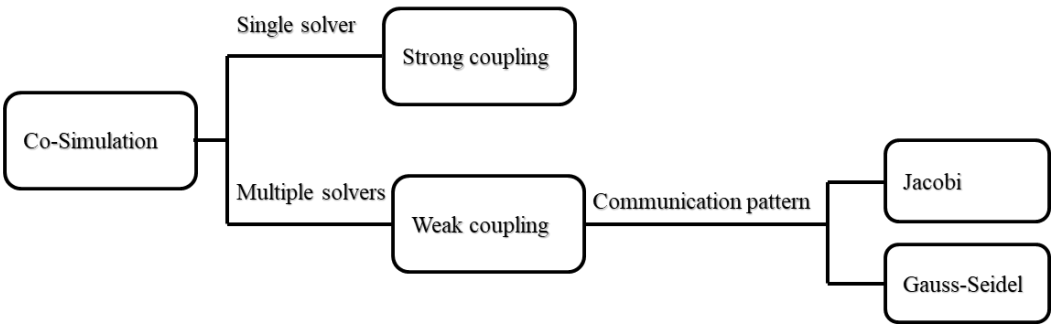



**Figure 2.3: Classification of co-simulation strategies**

In a strong coupling co-simulation, the subsystem models are solved by one single solver. For this approach, model equations of the involved subsystems are exposed to a co-simulation manager. It's required an iterative solution among subsystem models to satisfy a predefined convergence criterion within each global time step of the co-simulation process (Valasek 2008, Andersson 2013). As illustrated in Figure 2.4, the strong coupling results in nesting iteration loops, which involves an internal iteration within individual simulator, and an external iteration to achieve convergence of the coupled simulators (Trčka, Hensen et al. 2010).

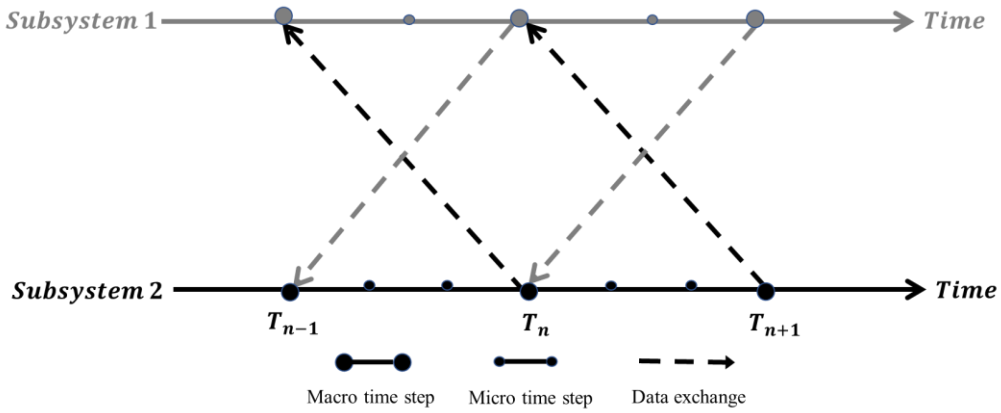
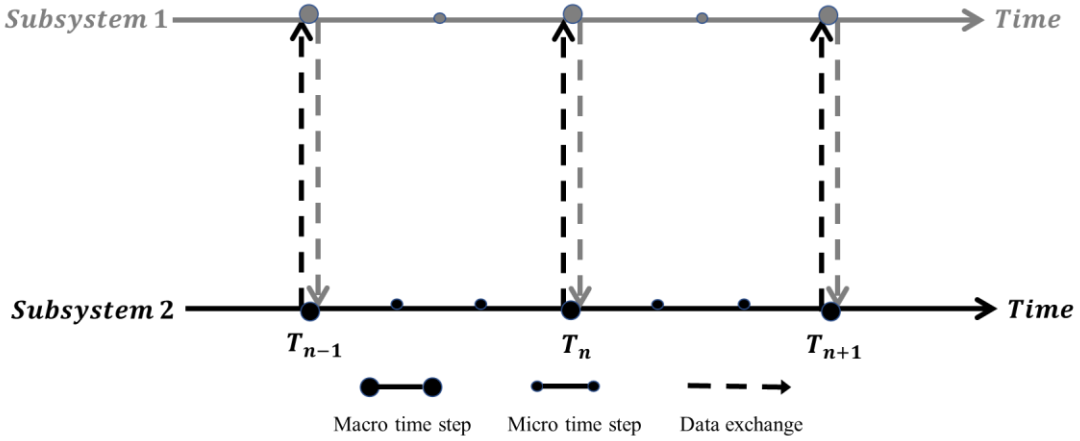


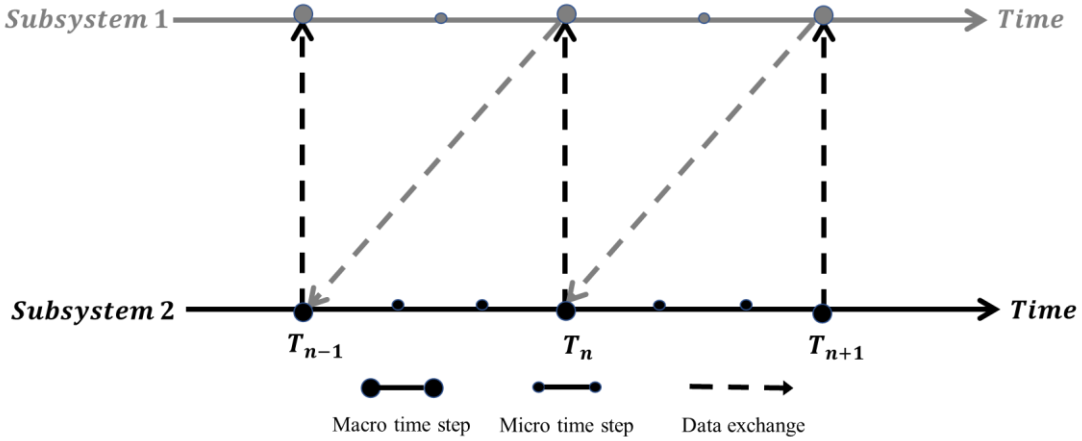

**Figure 2.4: Communication and data exchange sequences for strong coupling co-simulation**

In a weak coupling co-simulation, each subsystem is solved by its own solver. Simulators only exchange data from a preceding global time step. Each simulator can have their own iterations, and there is no iteration required between the coupled simulators at each step (Trčka, Hensen et al. 2010, Andersson 2013). Because the solvers of the coupled subsystem models are independent, the communication among the coupled subsystems can be either in parallel or sequential. For the parallel (Jacobi) type, coupled subsystems could run and exchange coupling data simultaneously in

parallel (Arnold 2010, Schierz and Arnold 2012). Alternatively, the sequential (Gauss-Seidel) type adopts coupled subsystems sequentially and the data is exchanged in a serial manner (Arnold 2010, Schierz and Arnold 2012). An overview of the computation and data exchange sequences of the weak coupling co-simulation is shown in Figure 2.5.




a) Weak coupling – parallel Jacobi type




b) Weak coupling – sequential Gauss-Seidel type

**Figure 2.5: Communication and data exchange sequences for weak coupling co-simulation**

Strong coupling requires communication tightly integrated into the numerical solver, which requires the solver to reverse back in time (including re-initialisation) (Trčka, Hensen et al. 2010). In contrast, iterations are not necessary for a weak coupling (Trčka, Hensen et al. 2010, Andersson 2013). This makes it much easier to implement the co-simulation than strong coupling. Therefore, it became the most commonly used coupling approach in the co-simulation community.

The urban energy system is a dynamic and hybrid system. To simulate such a system, a number of subsystem energy simulation tools are required to be coupled in a way that enables a dynamic simulation. Each of the tools is domain-specific and has its own speciality. The solvers of the subsystem simulation tools are more often kept separate. Therefore, it is more suitable for urban energy co-simulation to adopt the weak coupling approach. Hence it is chosen in this research to simulate urban energy system. In the remaining of the thesis, only weak coupling co-simulation will be discussed. When co-simulation is mentioned, unless otherwise specified, it represents weak coupling co-simulation.

In a co-simulation environment, a coupled subsystem can be represented in a mathematical model by ordinary differential equations as follows:

$$\begin{aligned} \dot{x}\,(\mathrm{t}) &= f(x(\mathrm{t}), \mathrm{u}(\mathrm{t})) \\ y(t) &= g(x(t), u(t)) \end{aligned} \qquad \text{2-1}$$

Where x(t) is the state variable, u(t) is the input and y(t) is the output variable of the subsystem, f is the differential portion, and g is the algebraic portion of the subsystem. In the co-simulation environment, there are at least two subsystems being involved. Assuming the total number of the subsystems is N (N ≥ 2), the $i^{th}$ subsystem can be represented by equation 2-2 where *i* is a positive natural number and $1 \leq i \leq \mathrm{N}$.

$$\begin{aligned} \dot{x}_i(\mathrm{t}) &= f_i(x_i(t), u_i(\mathrm{t})) \\ y_i(t) &= g_i(x_i(t), u_i(\mathrm{t})) \end{aligned} \qquad \text{2-2}$$

To have a better understanding of the two commonly used communication types of weak co-simulation, e.g. Jacobi and Gauss-Seidel, it is essential to represent their mathematical models to show similarities and differences between them. To keep it simple, a co-simulation environment which contains two subsystems is used here. Each of the two subsystems are calculated using a separate numerical algorithm.

Subsystem 1

$$\begin{aligned} \dot{x}_1(\mathrm{t}) &= f_1(x_1(t), u_1(\mathrm{t})) \\ y_1(t) &= g_1(x_1(t), u_1(\mathrm{t})) \end{aligned} \qquad \text{2-3}$$

Subsystem 2

$$\begin{aligned} \dot{x}_2(\mathrm{t}) &= f_2(x_2(t), u_2(\mathrm{t})) \\ y_2(t) &= g_2(x_2(t), u_2(\mathrm{t})) \end{aligned} \qquad \text{2-4}$$

In equation 2-3 and 2-4, $u_1(\mathrm{t})$ and $u_2(\mathrm{t})$ are the input variables and $y_1(t)$ and $y_2(t)$ are the output variables of the subsystem 1 and subsystem 2 respectively. The interactions between the two subsystems are handled by passing $y_1(t)$ and $y_2(t)$ to $u_2(\mathrm{t})$ and $u_1(\mathrm{t})$, which take place at macro time step points from $T_0$, $T_1$, . . ., $T_n$, . . ., to $T_N$. During one macro time step, the two subsystems are solved separately from each other in parallel.

Micro time steps of the two subsystems can be different. We assume the total micro time steps between one macro time step is J and K for the subsystem 1 and 2. $t_{n,j}$ and $t_{n,k}$ are the time step taken by the internal solver of each subsystem from $T_n$ to $T_{n+1}$, which is from $T_{n,0}$, $T_{n,1}$, . . ., $T_{n,j}$, . . ., $T_{n,J} = T_{n+1}$ and $T_{n,0}$, $T_{n,1}$, . . ., $T_{n,k}$, . . ., $T_{n,K} = T_{n+1}$ respectively. $\Phi_1$and $\Phi_2$ are used to represent the individual update functions that compute the values of the state variables $x_1(t_{n,j})$ and $x_2(t_{n,k})$ respectively. At the macro time step the output variables are $y_1(t_{n,j})$ and $y_2(t_{n,k})$.

At the micro step points, the update of the inputs of the two subsystems has to be approximated by extrapolation or interpolation. Using $\Phi_1$ and $\Phi_2$ to represent the individual update functions that compute the values of the state variables, equations 2-3 and 2-4 could be updated to equations 2-5 and 2-6.

$$\begin{aligned} \dot{\tilde{x}}_1(t_{n,j}) &= \Phi_1(\tilde{x}_1(t_{n,j}),\, \tilde{u}_1(t_{n,j})) \\ \tilde{y}_1(t_{n,j}) &= g_1(\tilde{x}_1(t_{n,j}),\, \tilde{u}_1(t_{n,j})) \end{aligned} \qquad 2\text{-}5$$

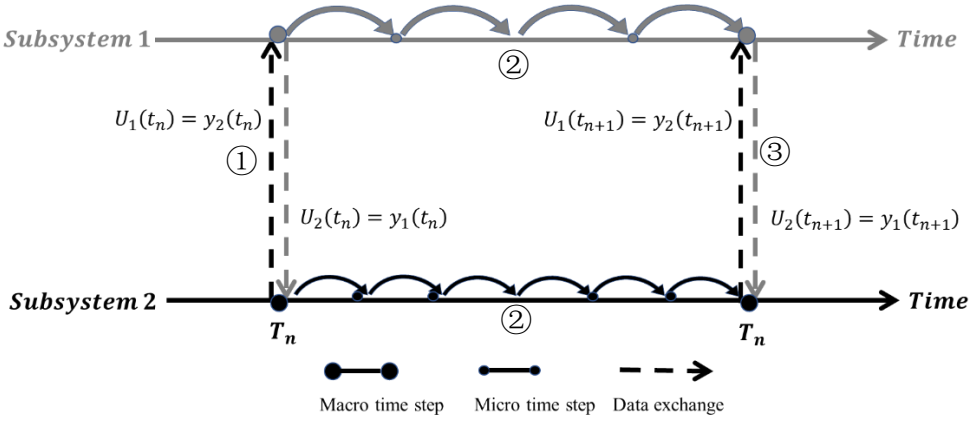
$$\begin{aligned} \dot{\tilde{x}}_2(t_{n,k}) &= \Phi_2(\tilde{x}_2(t_{n,\mathrm{k}}),\, \tilde{u}_2(t_{n,\mathrm{k}})) \\ \tilde{y}_2(t_{n,\mathrm{k}}) &= g_2(\tilde{x}_2(t_{n,\mathrm{k}}),\, \tilde{u}_2(t_{n,\mathrm{k}})) \end{aligned} \qquad 2\text{-}6$$

In the Jacobi type of weak coupling, the two subsystems are executed in parallel. Between two macro communication points, each simulator updates their state variables and output variables within defined micro time steps according to equation 2-5 and 2-6. At the end of each macro time step, the simulators exchange their new outputs with each other as follow.

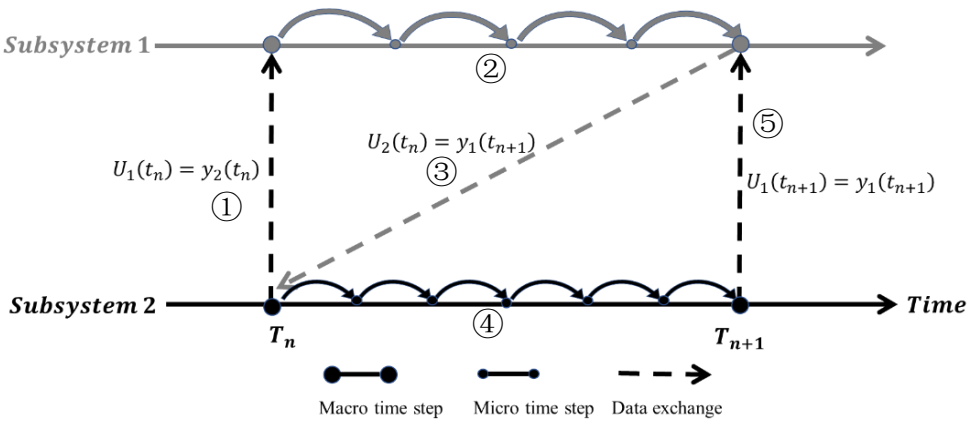
$$\begin{aligned} u_1(t_n) &= y_2(t_n) \\ u_2(t_n) &= y_1(t_n) \end{aligned} \qquad 2\text{-}7$$

As shown in Figure 2.6 a), after data exchange at time point $T_n$ (①), each subsystem is executed in parallel and proceed to the next time point $T_{n+1}$ (②), and start another data exchange at $T_{n+1}$ (③).

In the Gauss-Seidel type, the two subsystems are executed sequentially. At time point $T_n$, firstly the output value of subsystem 2 ($y_2(t_n)$) is passed to subsystem 1 to be assigned to $u_1(t_n)$ (①). Subsystem 1 then advances from $T_n$ to $T_{n+1}$ based on equations 2-5 and 2-6 (②). Once it's done, subsystem 1 feeds its output value $y_1(t_{n+1})$ back to subsystem 2 (③), and subsystem 2 then advances from $T_n$ to $T_{n+1}$(④). At the end of this macro step, it then passes the output value ($y_2(t_{n+1})$) to subsystem 1 (⑤). The more detailed illustration of the data exchange of Gauss-Seidel type of weak coupling could be seen in Figure 2.6 b).




a) Jacobi type of weak coupling




b) Gauss-Seidel type of weak coupling

**Figure 2.6: Data exchange of Jacobi and Gauss-Seidel type of weak coupling**

In the Jacobi type of coupling, all subsystems could run simultaneously. However, because each coupled simulation tool does extrapolation or interpolation in one macro step, extrapolation errors could be increased especially for those systems that have high dependency on each other's input. In comparison to Jacobi type of coupling, Gauss-Seidel type of coupling reduces extrapolation errors during co-simulation because only one simulator runs and it will not run until it receives latest output from the other simulator. This makes the Gauss-Seidel type of coupling more stable (Busch and Schweizer 2010, Bastian, Clauß et al. 2011).

Urban energy systems are complex in that it not only involves multiple domains but also interactions among different domains (Pantaleo, keirstead et al. 2013). Therefore, developing a single simulation tool that could simulate all aspects of urban energy systems is not realistic considering its complexity as well as time and human resources involved. Fortunately, there are a number of specialised domain-specific simulation tools exist already in the urban energy domain. If they could be integrated through a weak coupling scheme, this will be more effective. Using a co-simulation approach to integrate available simulation tools, there are several advantages. Firstly, it provides more flexibility. Tools could be chosen based on specific simulation purpose. Secondly, we can reuse existing simulators, middleware and libraries, which minimise implementation efforts. Thirdly, this can also provide the flexibility to substitute tools targeting similar domains, to balance computational cost and accuracy depending on the task in hand, e.g. more approximate methods for early or high uncertainty decisions and more sophisticated solvers for more complex and later design stage (lower uncertainty) tasks. Last but not the least, it provides scalability as more simulation tools could be integrated where needed.

To conduct urban energy systems co-simulation, we can either choose Jacobi or Gauss-Seidel type of coupling or a combination of both. Domain-specific energy simulation tools could be coupled based on the communication sequences among the composed subsystems using fixed macro time steps. Such communication sequences need to be controlled by a master algorithm regardless coupling approach to use.

Therefore, to achieve overall co-simulation of urban energy systems, an important requirement is to provide a master algorithm in the urban co-simulation architecture that orchestrates all the urban energy subsystem simulation tools coupled either in parallel (Jacobi) type or sequential (Gauss-Seidel) type or a combination of both. This

introduces requirement 4, ***R4. Master algorithm orchestrates simulators coupled in Jacobi or Gauss-Seidel***.

## 2.3 Co-simulation standards

To achieve a generic and scalable urban energy co-simulation by coupling simulators in a co-simulation platform for the analysis of complex urban energy systems, it is essential for the integrated urban energy subsystem simulators to be based on standardised interfaces. The standardised interfaces cover all stages of a co-simulation process including instantiation, initialisation, configuration, simulation, data exchange and termination.

Therefore, another important requirement of urban energy systems simulation is that urban energy subsystem simulators are integrated based on the most suitable co-simulation standards with well-defined co-simulation interfaces ***(***Requirement ***R5. Generic simulator integration based on a co-simulation standard)***. Urban energy subsystem simulators that are compliant with the co-simulation standard can be easily integrated in such a platform.

Currently, Functional Mockup Interface (FMI) and High Level Architecture (HLA) are the two most popular standards that provide standardised API supporting simulation interoperability, which helps to reuse existing implementations.

### 2.3.1 Functional Mockup Interface (FMI)

FMI, developed by Daimler AG within the ITEA2 project MODELISAR, is a tool independent standard to support both model exchange and co-simulation of dynamic models using a combination of XML files, binaries and C-code (either compiled in DLL/shared libraries or in source code) (Blochwitz, Otter et al. 2012). The standard has been widely adopted by more than 170 software vendors and is broadly used in a number of domains, including automotive, thermal, fluid and energy etc. (MODELISAR 2018, Association 2022).

According to the FMI specification, FMI for co-simulation provides support to couple two or more subsystem simulation tools, which focuses on discrete-time based simulation in a co-simulation environment. The data exchange between subsystems is restricted to discrete communication points. In the period between two communication points, the subsystems are run independently from each other by their individual

solver. Master algorithms control the data exchange between the slave subsystems and the synchronisation of all subsystems (Blochwitz, Otter et al. 2012).

The FMI specification defines essential interfaces for co-simulation. The interfaces enable diverse simulation tools to interoperate, covering all stages of the co-simulation process including instantiation, initialisation, configuration, access, modification, manipulation and termination in the form of C functions (FMI APIs). The calling sequences of these functions must follow the state chart shown in Figure 2.7.

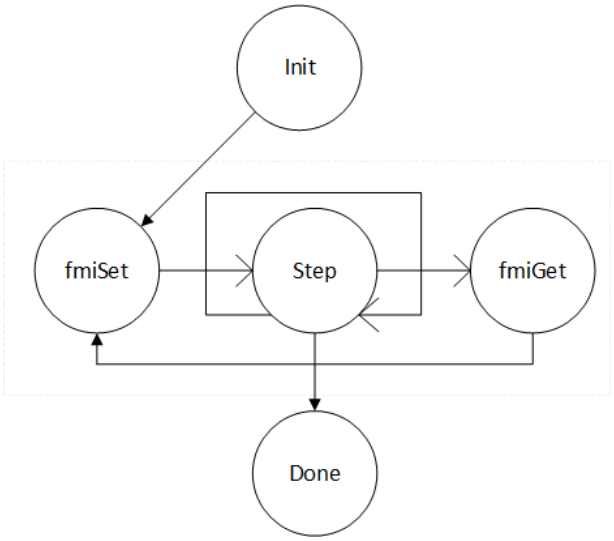



**Figure 2.7: State Machine of Calling Sequence**

In an FMI co-simulation environment, subsystem models, exported by their simulators together with solvers as runnable code, need to be distributed in one zip file with the extension ".fmu" called Functional Mockup Unit (FMU). A FMU contains the following components:

- An FMI model description file in XML format which contains all exposed variables.
- A model contains physical parameters or geometrical dimensions of a system (e.g. building system or power network).
- Run-time library of needed functions of a solver and FMI APIs library, which is implemented as Windows dynamic link libraries (.dll) or Linux shared object libraries (.so). An FMU can be used by master algorithms to create instances of the simulation models and perform co-simulation based on the model and input and output variables.
- Additional FMU data (like tables, maps) in FMU specific file formats.

In an FMI compliant simulation, a master slave scheme normally is adopted, which is illustrated in Figure 2.8. The master algorithm is essential for the co-simulation to provide orchestration of the entire co-simulation though it is not part of the FMI standard.

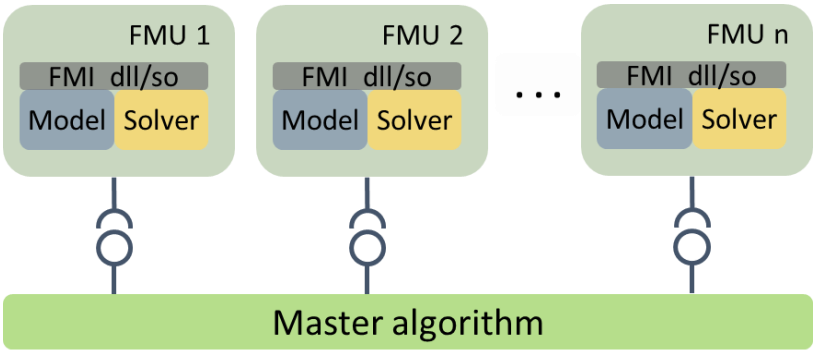



**Figure 2.8: FMI compliant co-simulation**

The strengths and weaknesses of the FMI standard are summarised as below.

**Strengths:** As FMI is a standard, simulation tools that follow its guidelines have built-in interoperability. The FMI specification defines interfaces for co-simulation in the form of C functions. This is a low-level approach. Such an approach makes FMI a platform independent standard, since C compilers are available for almost any platform. Although in the FMI standard, all the interfaces are defined in C, there are many researchers made contribution by defining FMI interfaces in other programming languages such as Python, C++ and Java. This facilitates coupling simulation tools programmed in different programming languages. In addition, the FMI specification does not include simulator-specific functionalities, which also makes it tool independent. It keeps the intellectual property of the simulation models protected, while collaborating and sharing such models in the form of FMUs.

**Weaknesses:** Despite the above-mentioned strengths, FMI does have some limitations. 1) There is increased robustness and stability issue of co-simulation in comparison to the monolithic simulation (Taveres-Cachat, Favoino et al. 2021). 2) The computational performance of co-simulation is degraded in comparison to the monolithic simulation (Schweiger, Gomes et al. 2018). 3) For large scale systems, stability and complexity constitute barriers for real-world complex co-simulation realisation (Schweiger, Gomes et al. 2019).

It needs to be mentioned that there needs development effort to implement a co-simulation system due to following features of the FMI standard. 1) FMI for Co-Simulation does not define master algorithms to control the data exchange between subsystems and the synchronisation of all coupled subsystem simulation tools. 2) FMI for Co-Simulation does not define communication technology for distributed scenarios, which means master algorithms need to implement communication layer to handle data exchange between simulation tools running on different platforms. 3) Co-simulation APIs need to be developed for simulation tools that do not support FMI standard.

### 2.3.2 High Level Architecture (HLA)

HLA (IEEE 2010) was developed by the U.S. Modelling and Simulation Coordination Office (M&S CO) in the mid-nineties (Dahmann, Salisbury et al. 1999). It is a standard for distributed simulation with the goal to provide a solution to solve simulation interoperability problems.

In an HLA based simulation, an interoperable simulation component called federate can be combined to create a simulation called federation. Federates do not communicate directly with each other. The communication is conducted through Run-Time-Infrastructure (RTI) in an event-based manner. The RTI provides various services for the HLA based simulation, including communication, synchronization, event passing and data exchange of federates (Dahmann, Fujimoto et al. 1997). Only with the permission of the RTI, a federate taking an active role in an HLA federation is able to advance in a simulation process.

HLA standard requests a machine-readable specification of Simulation Object Models (SOM) for each federate and a Federation Object Model (FOM) for a federation. While a SOM describes the shared objects, attributes and interactions of a particular federate, a FOM specifies all shared objects, attributes and interactions that may be used for data exchange within the federation (Tolk and Rainey 2014). The RTI keeps track of the subscriptions and publications of participating federates. This enables participating federates to send and receive data based on declared object attributes and interactions (Möller 2013). The structure of an HLA compliant simulation is illustrated in Figure 2.9. The publish/subscribe scheme of HLA results in weak coupling between

simulators, which may be distributed over a local or distributed cluster of computing nodes.

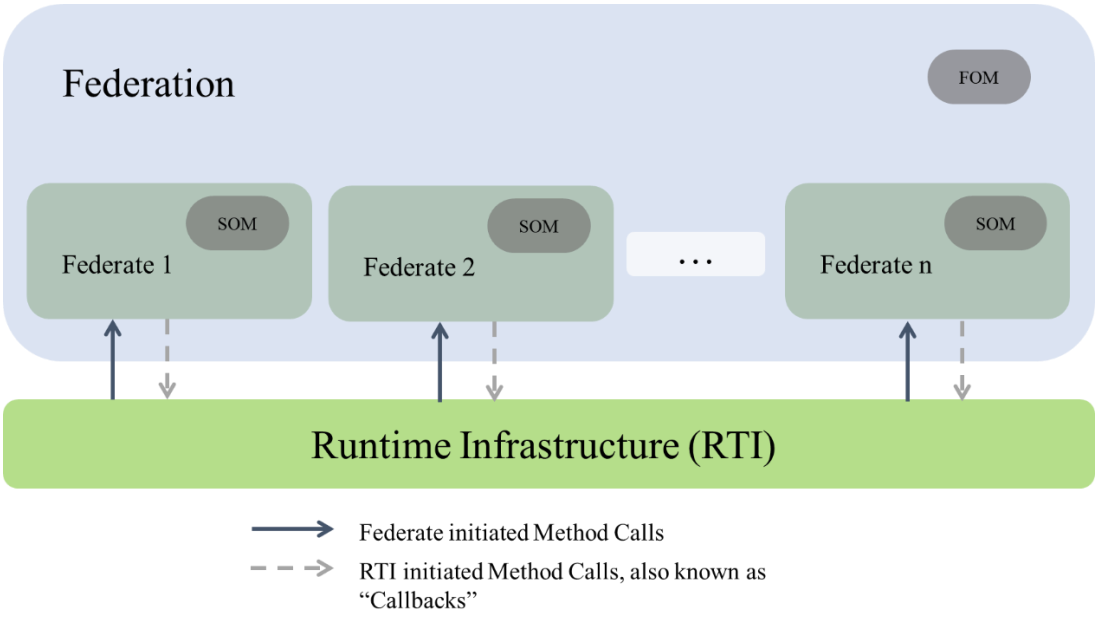



**Figure 2.9: HLA compliant simulation**

The strengths and weaknesses of the HLA standard are summarised as below.

**Strengths***:* As HLA is a standard, simulators that follow its guidelines will have a built-in interoperability. In addition, HLA is programming language- and platform-independent. Therefore, in an HLA compliant simulation, each federate can be developed independently and implemented using different programming languages and different hardware platforms (Fowler and Rose 2004). As a result, it allows the integration of existing simulators without imposing severe modifications on their internal structures, provided that they support some basic features (De Mello and Wagner 2002).

**Weaknesses***:* Similar to FMI, HLA does have some drawbacks. 1) Most RTIs adopt a central structure. As a result, the central mode of RTI can easily be the bottleneck of the system, making it difficult to expand the system (Kang 2010, Wang, Zhang et al. 2013). However, for the RTIs adopt a distributed structure, the harmonising algorithm is very complex, and the running efficiency is low in most situations (Kang 2010). 2) The RTI only supports distributed applications based on local area network. This typically yields message latency times on the order of tens, hundreds of microseconds resulting in performance issue for large simulations (Fujimoto and Hoare 1998). 3)

Federates forward every update to its subscriber. Such communicate mechanism brings additional overheads to the system (Hopkinson, Xiaoru et al. 2006)**.** 4) The HLA is very military specific and very complex, which makes it difficult to modify existing simulation tools to conform to its specification if they are not HLA compliant (Hopkinson, Xiaoru et al. 2006). 5) In terms of scalability, HLA based software packages provide the ability of highly parallel simulations of large-scale systems, but this introduces additional time-synchronisation issues (Müller, Georg et al. 2018).

Conclusion:

Requirement ***R5 (Generic simulator integration based on a co-simulation standard)*** requires the co-simulation platform to be based on a co-simulation standard that provides interoperability. By providing standardised interfaces to encapsulate existing simulation tools either in a format such as FMU or federate, both FMI and HLA can serve as the co-simulation standard for the urban energy conceptual co-simulation architecture.

Though HLA can serve as a standard that provides weak coupling for simulation tools, it focuses more on distributed simulation. The network protocol involved typically yields message latency time and the publish/subscribe scheme brings additional overheads because of the way federates communicate. Originally from military, HLA offers too much irrelevant functionality for urban energy systems co-simulation, which makes it unnecessarily complex, introduces a steep learning curve, and also brings performance issue.

As mentioned earlier, FMI is the most recently developed standard focusing on co-simulation. It is platform independent and can support simulation tools developed in a number of programming languages. It also protects the intellectual properties of the simulation models. It has been widely adopted by more than 170 tool vendors and is being broadly used in various domains. As a co-simulation standard, FMI has the potential to be a de facto standard accepted by co-simulation research community. Therefore, FMI is chosen as the co-simulation standard in the research presented in this thesis. Though a master algorithm needs to be developed, this will enable developers to control how the co-simulation is conducted and optimise the simulation if needed. In addition, for simulation tools that do not support FMI standard, APIs could be developed for them as required.

## 2.4 Co-simulation middleware

To simulate urban energy systems with a co-simulation approach, the integrated simulation tools need to communicate with each other in a unified way. In addition, it is required to provide a scheme for subsystem simulation tools composition and orchestration. Therefore, there are three additional essential requirements for the urban energy co-simulation platform to provide such functionality.

1) Variables for data exchange and required parameters of all simulation models to be integrated need to be defined in a unified way, which enables communication between different simulation models ***(R6. Unified scheme for model configuration and data exchange)***;

2) A set of clearly defined simulator composition APIs is required to composite simulators (***R7. Simulators composition***);

3) A set of clearly defined simulator orchestration API is required to orchestrate simulators and manage data exchange between the integrated simulators (***R8. Simulators orchestration*).**

Such scheme could be achieved through a master algorithm. Master algorithm is an essential component of a co-simulation environment, which is used to not only composite and orchestrate the execution of coupled simulation tools, but also manage data exchange of simulation models.

Currently, FMI standard does not specify master algorithms. To develop orchestration master algorithms for co-simulation from scratch is complicated. However, there are co-simulation middleware available that could be used to provide functions for facilitating master algorithms development. This could substantially simplify and speed up simulators composition and orchestration functionality development. The middleware mentioned here refers to reusable software that can significantly increase reuse by providing readily usable, software class libraries to programming tasks (Schmidt and Buschmann 2003). The co-simulation middleware helps the composition of simulation tools and management of data exchange and synchronisation among the integrated simulators. A simulation scenario is defined based on the given variables for data exchange and parameters of simulation tools.

Therefore, in the co-simulation architecture developed in this thesis, it is intended to generate master algorithms based on existing co-simulation middleware, such as Building Controls Virtual Test Bed (BCVTB) and mosaik.

### 2.4.1 Building Controls Virtual Test Bed (BCVTB)

BCVTB is an open-source middleware based on Ptolemy II (Ptolemaeus 2014) with additional executable entities (actors) implemented in Java, which was developed by Lawrence Berkeley National Laboratory (LBNL) at the University of California. It was designed to couple different simulation tools for co-simulation, which are commonly used in building simulation domain e.g. linking EnergyPlus, Modelica, Radiance and MATLAB/Simulink (Wetter 2011).

BCVTB contains its own APIs for co-simulation. These APIs are much more limited than FMI and are not supported by simulation tools that export FMUs (Nouidui 2014). In order to allow users to import and couple simulation tools that have been exported as FMUs, the BCVTB has been extended to support FMUs import by using JFMI (Brooks, Lee et al. 2012). JFMI is a Java wrapper for FMI to interact with the FMUs for co-simulation.

A BCVTB based simulation follows a client/server architecture, which is shown in Figure 2.10. The dotted line in the centre of the figure marks the components of the BCVTB server. The BCVTB server has a director that calls each actor at fixed synchronisation time step and manages data exchange between the actors (Wetter 2011). Each actor is responsible for one simulator client and needs a configuration file that specifies how the simulator can connect to the actor. The data exchange during runtime is carried out between the client and the BCVTB server via a (Berkeley Software Distribution) socket for inter-process communication using TCP/IP network protocol. This enables co-simulation be done either locally on one hosting computer, or remotely over the internet, possibly using different operating systems (Wetter 2011). The BCVTB director is responsible for passing data received from one client on to another according to a routing topology specified by the user. The BCVTB based simulation conducts data exchange between the simulator clients using a fixed synchronisation time step without iteration results in a weak coupling co-simulation strategy (Wetter 2011).

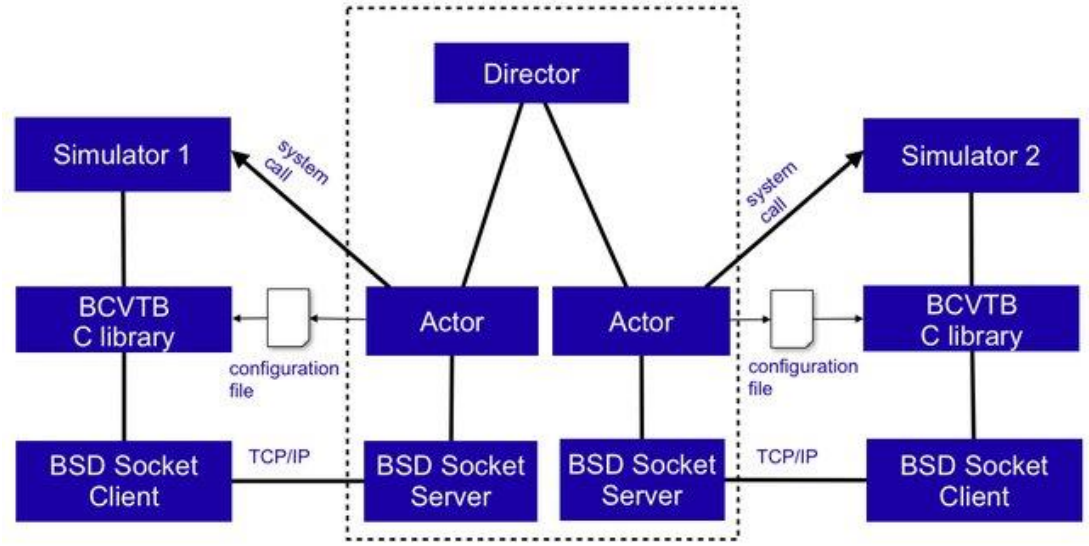



**Figure 2.10: Requirements, software architecture and implementation workflow**

**Strengths:** Using Java as the developing programming language makes the BCVTB middleware platform independent. In addition, as a co-simulation middleware, BCVTB was originated from building simulation domain. It is capable of coupling several simulation tools like EnergyPlus, Modelica, Radiance and MATLAB/Simulink for co-simulation (Wetter 2011). Currently, the BCVTB supports coupling FMUs, which makes it capable of integrating FMI-compliant simulators. Moreover, it is an open-source software, which makes it available for other researchers to modify.

**Weaknesses:** BCVTB introduces an additional socket-based transaction layer into the communication between simulators, which typically yields communication latency. In addition, BCVTB was developed based on Ptolemy II, which was not designed specifically for co-simulation but for research on signal processing and embedded systems (Brooks, Lee et al. 2008). This makes it unnecessarily complex for co-simulation and increases the learning curve for new users. For co-simulation system with a number of subsystem simulators integrated to simulate complex scenarios, the run-time performance of BCVTB can be very poor due to the complex design of Ptolemy II and additional transaction layer between the simulators.

### 2.4.2 Mosaik

Developed by OFFIS, mosaik is another co-simulation middleware, which aims to reuse existing simulation models in a common context to simulate complex smart grid scenarios in order to evaluate control strategies (Schütte 2011).

Basically, mosaik is a Python library that provides a set of APIs, including simulator API and scenario API. Integrated simulators are wrapped by simulator API. Simulator API allows the integration of simulators into a co-simulation environment by providing a model description file that contains variables for data exchange and a set of interface functions. The simulator API is comparable to the FMI API, but is more simplified, specifying only the minimal set of functions a simulator needs to provide in order to participate in a co-simulation. Scenario API facilitates the creation of a scenario script for a specific simulation purpose. Based on the description of the scenario script, mosaik simulator manager and scheduler call scenario API to initialise and configure the integrated simulators. They also connect one simulator with another based on the variables for data exchange, execute the co-simulation in a discrete-time manner and coordinate data exchange between integrated simulators (Rohjans, Lehnhoff et al. 2013).

Mosaik is able to integrate different, existing, technologically heterogeneous simulation tools (Rohjans, Lehnhoff et al. 2013). This allows an easy reuse of available simulation tools from various domains implemented in arbitrary programming language (Lehnhoff, Nannen et al. 2015). In addition, mosaik has its own semantic and control layer that could facilitate implementing scheduling algorithm (Scherfke and Schütte 2012).

**Strengths:** Unlike BCVTB that was developed based on Ptolemy II, and originally not developed for co-simulation purpose, mosaik was developed as a brand new co-simulation middleware focusing on smart grid. It is implemented in Python language, which makes it platform independent and allows easy integration of existing simulation tools, no matter what programming language they are implemented. The set of mosaik API, e.g. scenario API and simulator API, help to develop orchestration master algorithms rapidly. As the mosaik API and scheduler of mosaik were developed for co-simulation purpose, the run-time performance of mosaik based simulation is better than BCVTB based simulation.

**Weaknesses:** Mosaik currently is only able to couple simulation models wrapped by its own programming interfaces simulator API. Lacking FMI support makes it incapable of integrating models exported by FMI-compliant modelling tools. Moreover, the simulator API provides fewer functions than FMI API. This brings limitations for advanced co-simulation.

Conclusion:

Both BCVTB and mosaik provide necessary APIs to implement master algorithms to orchestrate the simulation and data exchange between individual simulation tools. Hence, they both fulfil the research requirements ***R6 (Unified scheme for model configuration and data exchange)***, ***R7 (Simulators composition)***, and ***R8 (Simulators orchestration)***. Both of them are open-source software. Therefore, necessary modification and function enhancement could be done to make it better serve the co-simulation of urban energy environment. However, in comparison to BCVTB, mosaik was designed in a layered manner and offers a more complete solution as a co-simulation middleware. In addition, it has better run-time performance. Given the complexity of urban energy simulation environment, in the research presented in this thesis, mosaik is chosen as the co-simulation middleware to facilitate master algorithm implementation.

## 2.5 Co-simulation software architecture requirement

If a project has not achieved a system architecture, including its rationale, the project should not proceed to full-scale system development. Specifying the architecture as a deliverable enables its use throughout the development and maintenance process.

- Barry Boehm (Boehm 1995)

Software architecture as a concept has its origins in the research of Edsger Dijkstra in 1968 and David Parnas in the early 1970s (Dijkstra 1968, Parnas 1972, Parnas 1976, Parnas 1979). They pointed out that the structure of a software system matters and it is critical to get the structure right at the very beginning. This idea laid the conceptual foundation for what became the study of software architecture today.

In order to develop a platform that meets the requirements raised in the previous sections, a conceptual architecture needs to be designed that links the requirements and implementation of the urban energy co-simulation platform (see Figure 2.10). It

can not only ensure the requirements of the urban energy co-simulation are satisfied, but also help us to study the feasibility of the software systems development. In addition, it can help to determine which requirements are viable.

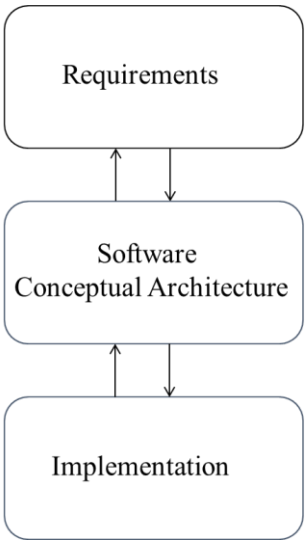



**Figure 2.11: Requirements, software architecture and implementation workflow**

Since the 1990s, the study of software architecture gained in popularity with research work focusing on architectural styles (patterns), architecture description languages, architecture documentation and formal methods (Garlan and Shaw 1993). It is effective to establish a common architectural framework across various domain-related products, which aids in achieving interoperability (Garlan and Perry 1995).

A good software architecture helps to provide modularity across and among various components within the software, which means that the components can be readily combined in different ways to meet different purposes without adding substantial additional code to make these components work together. With this foundation of flexible components, additional modules can be easily added (Meyer and Webb 2005). Therefore, a modular software architecture is a good choice for the design of the urban energy co-simulation software architecture. Such modular design provides flexibility as different modules and functions could be added based on requirements of the simulation.

With a thorough analysis of the requirements mentioned in the above sections and study of software architecture (Garlan and Shaw 1993, Clements and Northrop 1996, Bass, Clements et al. 2003, Fairbanks 2010, Medvidovic and Taylor 2010, Brown 2014), another requirement for a good design of the urban energy co-simulation

software architecture is derived. That is the urban energy co-simulation software architecture should consist of a set of modules facilitating simulation tools integration. Each module of the software architecture has its own functionality and provides disciplined interfaces for other modules to access its functionality. This becomes the final requirement of the co-simulation architecture, ***R9. Modular design***.

## 2.6 Summary of urban energy co-simulation requirements

Through a detailed review of urban energy simulation tools, co-simulation approach, related standards and middleware, requirements of a generic co-simulation approach have been identified in this chapter, i.e. Obj. I (Identify requirements for the generic and scalable urban energy co-simulation) is achieved.

To summarise, the nine requirements for co-simulation of the urban energy systems are listed in Table 2.2.

**Table 2.2: Urban energy systems co-simulation requirements**

| |
|---|
| R1. Discrete-time simulator integration support |
| To integrate simulation tools in discrete-time simulation manner with fixed step size. |
| R2. Different temporal resolution support |
| Simulation tools with different temporal resolution can be composed and integrated. |
| R3. Simulation tools in various programming language support |
| To support integration of simulation tools implemented in different programming languages. |
| R4. Master algorithm orchestrates simulators coupled in Jacobi or Gauss-Seidel |
| To provide master algorithm in the urban co-simulation platform that orchestrates the co-simulation of urban energy subsystem simulators coupled either in parallel (Jacobi) type or sequential (Gauss-Seidel) type or a combination of both. |

| R5. Generic simulator integration based on a co-simulation standard |
| --- |
| To integrate subsystem simulators based on standardised interfaces, so that interoperability among different tools could be achieved. |
| R6. Unified scheme for model configuration and data exchange |
| Parameters and attributes of simulators and their models to be integrated need to be defined in a unified way, which enables communication between simulators and their model integration through unified model configuration and data exchange scheme. |
| R7. Simulators composition |
| A set of clearly defined simulator composition API functions are required to composite simulators. |
| R8. Simulators orchestration |
| A set of clearly defined simulator orchestration API functions are required to orchestrate simulators and manage data exchange between the integrated simulators. |
| R9. Modular design |
| The conceptual architecture of the urban energy co-simulation environment needs to consist of a set of modules facilitating simulation tools integration. Each module of the architecture has its individual functions and provides interfaces for other modules to access its functionalities. |

# 3 CONCEPTUAL CO-SIMULATION ARCHITECTURE DESIGN

The urban energy conceptual co-simulation architecture is presented in this chapter, to address the nine requirements identified through the review presented in Chapter 2.

## 3.1 Conceptual co-simulation architecture

Considering the nine requirements identified during the review, the urban energy co-simulation software architecture is designed in a layered modular way, in order to satisfy requirement ***R9 (Modular design*)**. Each layer of the architecture consists of a modular set of components with its own functions and provides disciplined interfaces for other modules to access its functionalities. The design will ensure that the platform developed based on this architecture could be extended to include alternative standards, middleware and integration tools. In addition, each layer satisfies certain requirements of the urban energy co-simulation presented in Chapter 2.

The architecture is consisted of four layers, i.e. simulator layer, interconnection layer, interoperability layer and control layer. Figure 3.1 illustrates the conceptual architecture of the urban energy co-simulation. Individual urban energy simulation tools could be integrated through the simulator layer and the overall simulation purpose could be implemented through the control and interoperability layers. There are no direct control and data exchange between the integrated simulation tools. Rather they are implemented through the higher layers. The more detailed description of the functionality of each layer is presented in the next section.

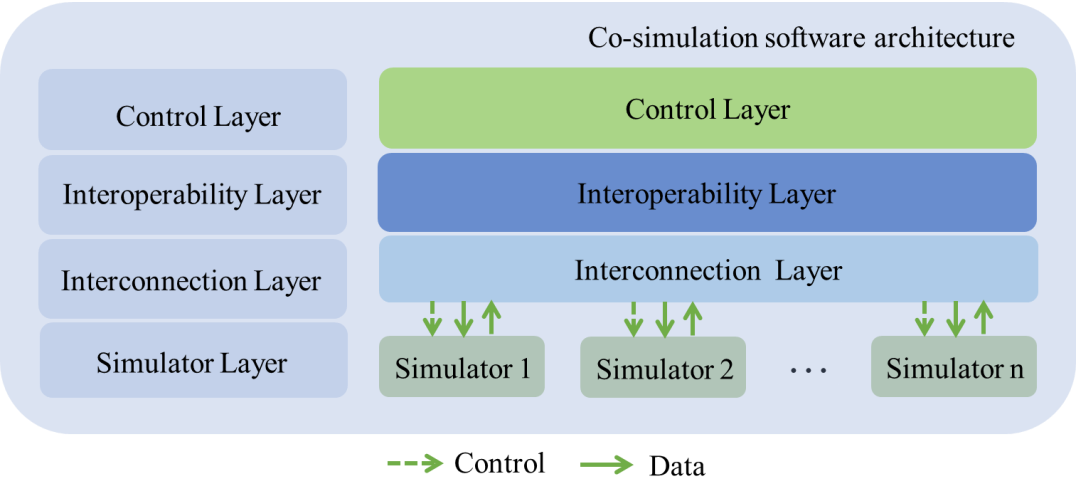



**Figure 3.1: Conceptual architecture of the urban energy systems co-simulation**

## 3.2 Co-simulation architecture layer description

More detailed explanation of each layer is given as follows.

- Simulator Layer

The simulation tools to be integrated are wrapped in this layer. A co-simulation process includes multiple stages such as instantiation, initialisation, configuration, simulation, data exchange and termination. Co-simulation standard provides well defined API functions to manipulate simulators throughout the co-simulation process. Therefore, the relevant requirement for simulator layer of the co-simulation platform to satisfy is ***R5. Generic simulator integration based on a co-simulation standard***.

To integrate into the co-simulation platform in a generic way, the urban energy subsystem simulation tools are required to be based on standardised interfaces. This means individual simulators to be integrated should provide required co-simulation interfaces according to the standard. The integrated simulators do not communicate with each other directly but through the standardised interfaces to the upper layer of Co-simulation Platform for Ecological-urban (COPE).

As discussed in Section 2.3, the tool-independent, open-source standard FMI was chosen as the co-simulation standard for the co-simulation platform. Therefore, this layer will enable simulators to be integrated into the co-simulation platform to be compliant with the FMI co-simulation standard, with full support for FMU export.

- Interconnection Layer

The urban energy simulation tools to be integrated into the co-simulation platform could support different co-simulation standards, and be implemented in different programming languages such as, C, C++, Java, Python, etc. Therefore, the FMUs in the simulator layer could provide FMI interfaces in different programming languages. As a result, it's a challenge for them to communicate and interconnect with each other. The interconnection layer aims to solve such technical challenges, hence satisfying the requirement ***R3 (Simulation tools in various programming language support)***.

To meet the requirement ***R3***, the interconnection layer must enable integration of FMUs with FMI interfaces implemented in different programming languages. This requires the interconnection layer of the co-simulation platform to use necessary FMI libraries to facilitate the handling of FMUs with exposed FMI interfaces implemented in various programming languages.

- Interoperability layer

The interoperability layer provides a mechanism to composite and orchestrate FMUs by calling FMI interfaces provided by the interconnection layer. Composition API and orchestration API will be implemented in this layer to address five requirements identified in Chapter 2, which are ***R1 (Discrete-time simulator integration support)***, ***R2 (Different temporal resolution support)***, ***R6 (Unified scheme for model configuration and data exchange)***, ***R7 (Simulators composition)***, and ***R8 (Simulators orchestration)***.

As mentioned in Section 2.3, the interoperability layer utilises modules and APIs provided by mosaik co-simulation middleware to speed up simulators composition and orchestration functionality development. The core modules of mosaik are scenario, simulator manager and scheduler, which provides essential co-simulation functionality facilitating simulators composition and orchestration.

- Control Layer

The purpose of this layer is to control and coordinate the simulation in order to achieve the overall simulation intentions of the co-simulation scenarios defined by experts, who have knowledge about urban energy systems that are to be simulated and the available simulation tools to be integrated. This requires the control layer to provide a solution to define co-simulation scenarios and execute the co-simulation of urban

energy subsystem simulation tools coupled either in parallel (Jacobi) type or sequential (Gauss-Seidel) type or a combination of both. The requirement ***R4 (Master algorithm orchestrates simulators coupled in Jacobi or Gauss-Seidel)*** will be addressed by this layer.

The master algorithm will include a co-simulation scenario and functions to run the scenario. By calling the composition API and orchestration API provided by the interoperability layer, the simulators can be composed, connected and synchronised as intended and data could be exchanged. In doing so, simulators implemented in different programming languages and modelled with different temporal resolutions could be orchestrated in a discrete-time manner.

Chapter 3 presented the conceptual co-simulation architecture for urban energy systems. Obj. II (Design a conceptual co-simulation architecture that will be able to integrate urban energy simulation tools from different domains) is achieved correspondingly in this chapter. The implementation of each layer of the urban energy co-simulation platform based on the architecture will be explained in the next chapter to achieve Obj. III (Develop an urban energy co-simulation platform based on the conceptual architecture).

# 4 URBAN ENERGY CO-SIMULATION PLATFORM DEVELOPMENT

In the previous chapter, a conceptual co-simulation architecture was developed to address the requirements of urban energy co-simulation. Based on the conceptual architecture, a Co-simulation Platform for Ecological-urban (COPE) was developed for urban energy systems simulation. The platform follows a layered modular design in order to achieve a generic and scalable co-simulation.

The implementation of the four main layers of COPE is explained in detail in this chapter to demonstrate how the requirements are addressed in each layer. In addition, how the modularity of COPE enables it to integrate different urban energy simulation tools conforming to the chosen co-simulation standard with minimum effort and cost will be presented. At the end of the chapter, the process to apply COPE to a specific simulation scenario is given, in order to facilitate other users of the platform.

## 4.1 Simulator layer

As discussed in Section 2.3, the tool-independent, open-source standard FMI was chosen as the co-simulation standard for COPE. Therefore, to meet requirement ***R5 Generic simulator integration based on a co-simulation standard,*** this layer will enable simulators to be integrated into the platform to be compliant with the FMI co-simulation standard, with full support for FMU export.

The FMI standard specification defines essential interfaces for co-simulation. In order to be compliant with the FMI co-simulation standard, FMI co-simulation API

functions need to be implemented for each integrated simulation tool. These API functions provide interfaces to instantiate, initialise, configure, execute simulation tools and conduct data exchange between the integrated tools. Furthermore, FMU export functionality needs to be provided by these FMI-compliant simulation tools. This functionality enables an FMU zip archive to be generated, which includes a model description XML file, a simulation configuration XML file, a simulation model, a model solver and a run-time library containing compiled FMI co-simulation APIs as shown in Figure 4.1. A model solver is a software component of the simulation tool which includes algorithms to solve simulation models. During a co-simulation process, FMUs act as co-simulation slaves. They are called by a master algorithm to create instances of the simulation models and orchestrate them by calling the FMI functions.

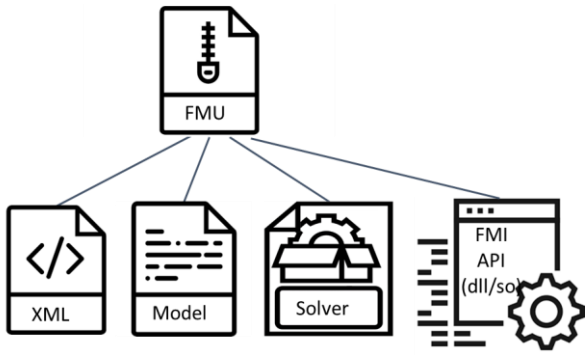



**Figure 4.1: FMU zip archive**

In COPE, the development of the simulator layer is based on integration of simulators that are compliant with the FMI co-simulation standard with FMU export support, which is shown in Figure 4.2.

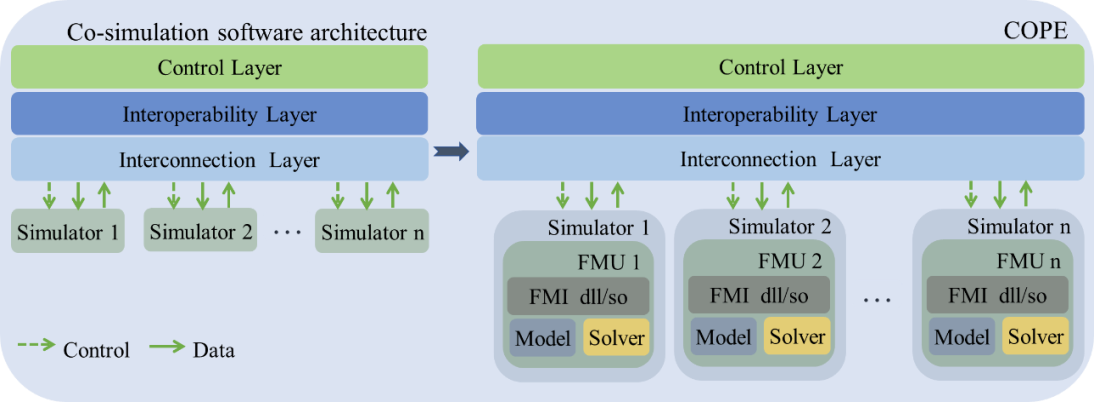



**Figure 4.2: COPE – Detail of the simulator layer**

To participate as a component of the co-simulation platform, simulation tools to be integrated are required to support FMU export. In the research presented in this thesis, we focus on co-simulation by integrating the available time-stepped urban energy simulation tools, such as EnergyPlus and No-MASS, to the co-simulation platform. Correspondingly, simulators in Figure 4.2 are instances of EnergyPlus and No-MASS; meaning for example that we could have multiple instances of EnergyPlus or of No-MASS to model individual buildings.

As mentioned in Section 2.2, some tools already provide functionality to support FMU export. For example, EnergyPlus already supports FMU export by offering a software package named EnergyPlusToFMU to export EnergyPlus together with simulation model as a FMU for co-simulation. However, there are also tools not supporting the FMU export yet. Therefore, to enable them to support FMU export, it is required to implement FMI co-simulation API functions and offer FMU export functionality. It needs to mention that either source code of the simulation tools needs to be accessed or libraries are available.

As the source code of No-MASS is accessible, FMI co-simulation API functions have been implemented in this research in order to integrate it. Depending on the programming language of the tools, FMI co-simulation API functions can be implemented in different programming languages like C, C++, Java, Python, etc. EnergyPlus and No-MASS were developed in C++ and FMI co-simulation API functions were implemented in C for each of them in this research. To implement FMI co-simulation APIs for other urban energy simulation tools in other programming languages, these API functions could be referred to as template in the future.

The approach to make a non-FMI-compliant simulation tool, like No-MASS, with FMU export is demonstrated in this section through the following three steps. The same approach can be adopted to make other urban energy simulation tools to be FMI-compliant with FMU export.

Step 1 – FMI functions development

Firstly, to make No-MASS compliant with the FMI standard, necessary FMI co-simulation API functions need to be developed. In the research presented in this thesis, FMI co-simulation API functions are developed based on FMI standard 2.0.1.

However, as the co-simulation platform is in a layered modular design, FMI 3.0 support can be implemented as additional modules in the future.

In the FMI 2.0.1 release, there are three common functions, eight data exchange functions and 28 functions for co-simulation, which cover all stages of a co-simulation process including instantiation, initialisation, configuration, simulation, data exchange and termination. In this release, wrapper functions are defined for corresponding FMI functions. For example, fmi2_import_get_version() is the wrapper for the FMI function fmiGetVersion(). The FMI common and co-simulation functions are listed in Table 4.1.

**Table 4.1: FMI functions**

| Common Functions | | |
|---|---|---|
| fmiGetTypesPlatform | fmiGetVersion | fmiSetDebugLogging |
| Data Exchange Functions | | |
| fmiSetReal | fmiSetInteger | fmiSetBoolean |
| fmiSetString | fmiGetReal | fmiGetInteger |
| fmiGetBoolean | fmiGetString | |
| Co-Simulation Functions | | |
| fmiInstantiate | fmiTerminate | fmiDoStep |
| fmiReset | fmiFreeInstance | fmiCancelStep |
| fmiGetStatus | fmiGetRealStatus | fmiGetIntegerStatus |
| fmiGetBooleanStatus | fmiGetStringStatus | fmiGetTypesPlatform |
| fmiSetRealInputDerivatives | fmiGetOutputDerivatives | |

In the research presented in this thesis, only key functions were implemented for co-simulation in C programming language. The key functions were selected based on a simplified state-machine for the calling sequence of co-simulation functions shown in Figure 4.3.

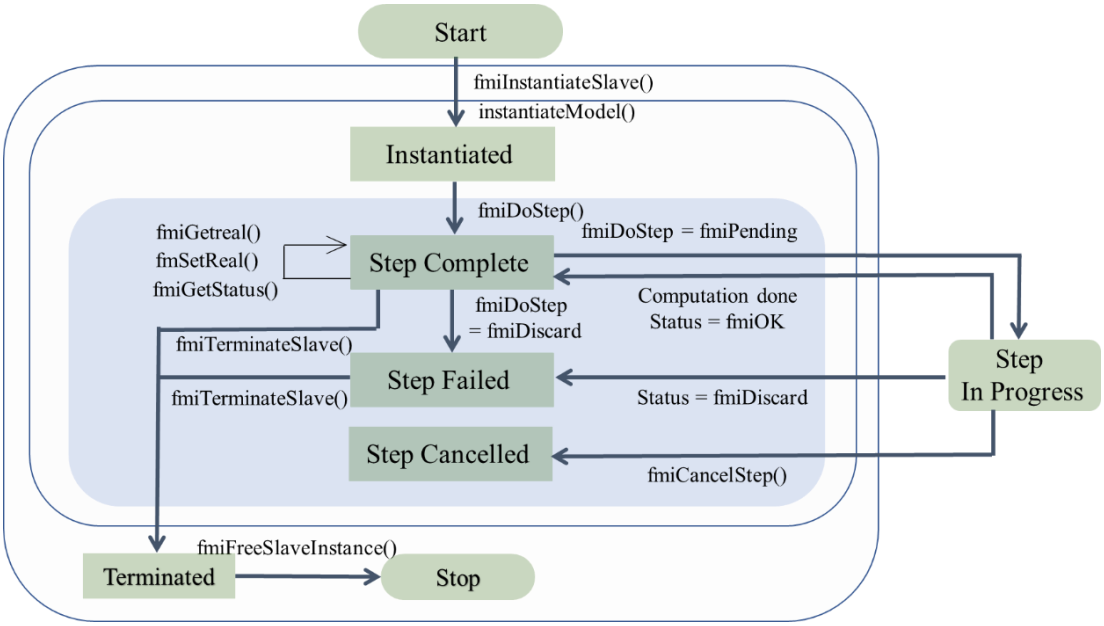



**Figure 4.3: State-machine for the calling sequence of co-simulation functions**

The key co-simulation functions shown in Figure 4.3 are summarised in Table 4.2. A detailed definition of the key functions can be found in Appendix 1.

**Table 4.2: Key FMI functions involved in the calling sequence of co-simulation**

| FMI functions | Description |
|---|---|
| fmiInstantiate | To instantiate the slave FMU |
| fmiFreeInstance | To dispose the given instance, unload the loaded model, and free all the allocated memory |
| instantiateModel | To instantiate the model |
| fmiSetReal | To set values of inputs |
| fmiGetReal | To get values of the variables by providing their variable references |
| fmiDoStep | To run a computation of a time step |
| fmiGetStatus | To inform the master about the actual status of the simulation run. |
| fmiCancelStep | To cancel the current computation. |
| fmiTerminate | To signal the slave the end of the co-simulation. |

The implementation of the key FMI co-simulation functions for a non-FMI-compliant simulation tool requires new functions to be developed to fit for their purposes. The newly developed FMI functions are summarised in Table 4.3. A detailed definition of the new FMI functions can be found in Appendix 2.

**Table 4.3: New FMI co-simulation functions**

| FMI functions | Description |
|---|---|
| DataStore::addVariable | Store input and out variables in an associative container |
| DataStore::addValue | Get input values |
| DataStore::getValue | Provide output values |
| Simulation::preprocess | Necessary pre-processing before simulation |
| Simulation::parseConfiguration | Parse simulation configuration xml file |
| Simulation::setupSimulationModel | Pre-condition building model |
| Simulation::preTimeStep | Process before running timeStep |
| Simulation::timeStep | Run a computation of a time step |
| Simulation::postprocess | Postprocess a computation of a time step |

Step 2: Generate SimulationConfig.xml and ModelDescription.xml

The SimulationConfig.xml file defines the time step, starting and end date of the simulation. ModelDescription.xml defines model name, variables and its implementation.

To support FMU export, the second step is to develop templates of SimulationConfig.xml and ModelDescription.xml. For certain use cases, if simulation period and time step need to be different, they can be modified in the SimulationConfig.xml file. Similarly, exposed variables can be modified in the

ModelDescription.xml. An example of a SimulationConfig.xml and modelDescription.xml is shown as follows.

**SimulationConfig.xml**

```
<simulation>
  <seed>6600</seed>
  <timeStepsPerHour>12</timeStepsPerHour>
  <beginMonth>1</beginMonth>
  <endMonth>12</endMonth>
  <beginDay>1</beginDay>
  <endDay>31</endDay>
  <learn>0</learn>
 <buildings>
...
 </buildings>
<models>
...
</models>
</simulation>
```

**modelDescription.xml**

```
<fmiModelDescription fmiVersion="2.0"
 modelName="FMI"
 modelIdentifier="FMI"
 guid="{fd719ef5-c46e-48c7-ae95-96089a69ee64}"
 generationDateAndTime="2022-04-17T19:12:58Z"
```

```
…
<ModelVariables>
  <ScalarVariable causality="input"
name="Block1:Zone1ZoneAirRelativeHumidity" valueReference="1">
   <Real declaredType="Modelica.Blocks.Interfaces.RealInput"
start="0.0" />
  </ScalarVariable>
…
  <ScalarVariable causality="output"
name="Block1:Zone1LightState" valueReference="12">
   <Real declaredType="Modelica.Blocks.Interfaces.RealInput"
start="0.0" />
  </ScalarVariable>
 </ModelVariables>

<Implementation>
  <CoSimulation_Tool>
   <Capabilities
    canHandleVariableCommunicationStepSize="true"
    canHandleEvents="true"
    canRejectSteps="false"
    canInterpolateInputs="false"
    maxOutputDerivativeOrder="0"
    canRunAsynchronuously="false"
    canSignalEvents="false"
    canBeInstantiatedOnlyOncePerProcess="true"
```

```
      canNotUseMemoryManagementFunctions="true"/>
    </CoSimulation_Tool>
  </Implementation>
</fmiModelDescription>
```

Step 3: Compile No-MASS and generate FMU zip archive

After the FMI functions are implemented and the XML files are generated, the last step is to archive FMU. The makefile of No-MASS was modified to compile FMI-compliant binaries and generate the corresponding FMU zip archive. A schematic view of the directory structure of a zip archive of a No-MASS FMU is shown as follows.

**Table 4.4: Structure of a No-MASS FMU zip-file**

| **Directory structure of a zip archive of a No-MASS FMU** |
|---|
| agentFMU/ |
| \| |
| \|-------binaries/ |
| \| \|-------linux64/ // Target platforms are specified as: <os><architecture> |
| \| \|-------FMI.so // Model code, must export the FMI-specified functions |
| \|-------modelDescription.xml // Detailed model information, e.g. all exposed variables, etc. |
| \|-------SimulationConfig.xml // Detailed simulation information, e.g. simulation period and time step |

For a co-simulation, the FMU export support serves as the starting point, which enables individual urban energy subsystem simulators to be integrated into the simulator layer of COPE.

## 4.2 Interconnection layer

The urban energy simulation tools to be integrated into the co-simulation platform can be programmed in different programming languages such as, C, C++, Java, Python, etc. Therefore, FMUs in the simulator layer could provide FMI interfaces in different programming languages.

To address the requirement ***R3***. ***Simulation tools in various programming language support***, the interconnection layer of COPE is responsible to enable integration of FMUs with FMI interfaces implemented in different programming languages. This requires the interconnection layer to use necessary available FMI libraries to facilitate the handling of FMUs with exposed FMI interfaces implemented in various programming languages. The libraries need to be able to load FMU with FMI API in its own programming language, and expose the API functions in the same programming language with exposed interfaces by its upper layer, the interconnection layer.

Currently, there are several FMI libraries, such as PyFMI, JavaFMI, FMI++, FMI Library and FMU SDK, available to facilitate the interaction of FMUs with FMI interfaces in different programming languages. PyFMI is a python library for loading and interacting with FMUs using Python native calls based on FMILibrary, which is maintained by Modelon AB (Andersson, Åkesson et al. 2016). JavaFMI is a Java library for controlling FMUs, developed by SIANI institute at Las Palmas University (Evora, Hernandez et al. 2014). FMI++ is a C++ library that provides access to handle FMUs (Widl, Müller et al. 2013). The FMI Library and FMU SDK provide basic access to FMI functions, which are written in C and C++ respectively (Modelon 2014, QTronic 2014). The features of these libraries are summarised in Table 4.5.

**Table 4.5: FMI libraries for interacting FMUs**

| Name | FMI co-simulation support | Programming language | Version | License |
|---|---|---|---|---|
| PyFMI | Yes | Python | 2.2 | LGPLv3 |
| JavaFMI | Yes | Java | 2.6.1 | LGPLv3 |
| FMI++ | Yes | C++ | - | BSD |
| FMI Library | Yes | C | 2.0.1 | BSD |
| FMU SDK | Yes | C++ | 2.0.3 | BSD |

As discussed in Section 2.4, mosaik was selected as the co-simulation middleware in this research to speed up the simulators composition and orchestration functionality development in the upper layer of the interconnection layer, the interoperability layer.

Currently, mosaik co-simulation middleware can composite and orchestrate simulators by calling API functions written in Python and Java. The available urban energy simulation tools, like EnergyPlus, CitySim, No-MASS are programmed in C++ and provide FMI API functions written in C. Therefore, the interconnection layer needs to provide a solution to load FMUs with C-language FMI API functions and expose FMI API functions in Python or Java, so that the API interface can be called by mosaik co-simulation middleware.

Based on the simulation tools planned to be integrated in this research, PyFMI together with FMI Library were adopted in the interconnection layer to fulfil this task. The implementation of the interconnection layer of COPE is shown in Figure 4.4.

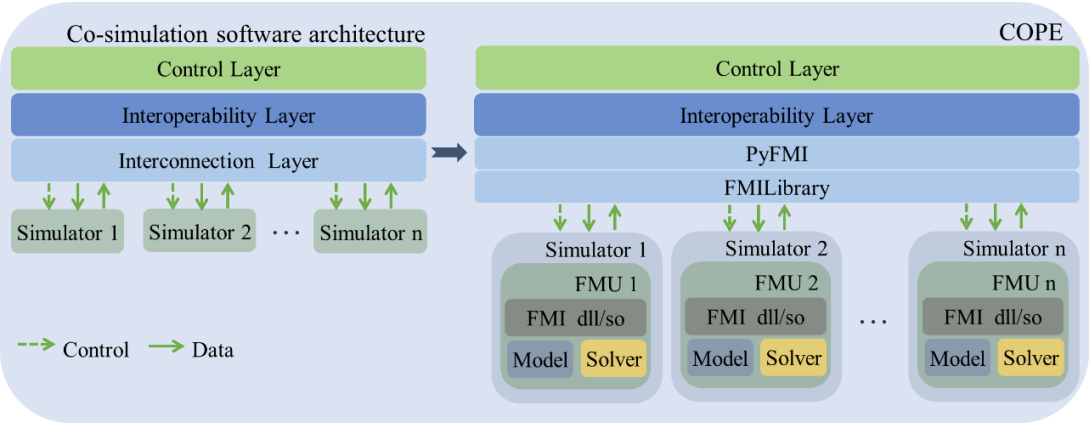



**Figure 4.4: COPE – Detail of the interconnection layer**

Currently, the FMI library (version 2.0.1) and PyFMI (version 2.2) were already integrated in the interconnection layer of the platform, which provides Python native calls to the interoperability layer to load and interact with FMUs within C FMI functions. It needs to be pointed out that the current version of PyFMI does not support co-simulation tool coupling. Therefore, in this research, the fmi.py file of PyFMI was modified by adding FMI model initialisation function to make PyFMI capable of direct and generic coupling co-simulation tools.

Because COPE is designed in a modular architecture, the interconnection layer could be extended easily by adding more FMI libraries. In the future, if there are simulation tools, with FMI interfaces programmed in other programming language such as C++ and Java, need to be integrated into the platform, we can just plug FMI++ and JavaFMI into the interconnection layer. Such approach enables COPE to integrate urban energy simulation tools implemented in different programming languages shown in Appendix 3 into the co-simulation platform.

## 4.3 Interoperability layer

The purpose of the interoperability layer of COPE is to provide a mechanism to composite and orchestrate FMUs by calling FMI interfaces provided by the PyFMI and FMI Library of the interconnection layer. This layer will provide relevant functions to address the following requirements ***R1. Discrete-time simulator integration support***, ***R2. Different temporal resolution support***, ***R6. Unified scheme for model configuration and data exchange***, ***R7. Simulators composition***, and ***R8. Simulators orchestration***.

In the research presented in this thesis, the interoperability layer utilises modules and APIs provided by mosaik co-simulation middleware to speed up simulators composition and orchestration functionality development. The core modules of mosaik are simulator manager, scenario handler and scheduler, which provide essential co-simulation functionality facilitating simulators composition and orchestration. Simulator API and scenario API are two set of APIs offered by core mosaik modules for simulators integration, co-simulation scenarios creation and simulators orchestration (OFFIS 2017).

The simulator manager module of mosaik is responsible for starting simulator processes, managing the data exchange between simulators, and shutting them down. This is achieved by calling mosaik simulator API, which is a set of functions for advancing simulators in a discrete-time stepped fashion and exchanging data that simulators receive and provide.

The mosaik simulator API functions can be classified into three categories: initialisation functions, such as init and create, run-rime functions, such as get_data, set_data and step, and termination function, such as finalize. The initialisation and termination functions are called only once during a co-simulation process. After the initialisation functions are called, the run-rime functions are called repeatedly until the co-simulation reaches its end. The key mosaik simulator API functions are summarised in Table 4.6. A detailed definition of the Mosaik simulator API functions can be found in Appendix 5.

**Table 4.6: Mosaik simulator API**

| Function | Description |
|---|---|
| init | Initialise a simulator by utilising simulator and model initialisation parameters |
| create | Create model instances |
| step | Perform a simulation step based on some input data for a time interval |
| get_data | Get data from a simulator |
| set_data | Set input data for a simulator |

| | |
|---|---|
| finalize | Do some clean-up operations after the simulation finished |

The mosaik scenario module handles simulator configuration and composition, as well as co-simulation execution based on variables specified for a scenario. The specified variables describe how the simulators and their models are configured and interconnected with each other. Scenario API is a set of functions provided by the scenario module to set up and run a co-simulation scenario. The scenario API provides functionalities to start simulators and instantiate simulation models, connect models with each other by establishing dataflows between the simulators, and execute simulation of the connected models. The key mosaik scenario API functions for flexible simulator coupling and execution are summarised in Table 4.7. A detailed definition of the mosaik scenario API functions can be found in Appendix 6.

**Table 4.7: Mosaik scenario API**

| Functions | Description |
|---|---|
| start | Start a simulator based on the configuration |
| connect | Connect source model to destination model. |
| run | Execute simulation of connected models |

The mosaik scenario module provides scenario definition scheme to define variables of coupled simulators and their models, as well as data exchange between simulators, in a unified way. Following the scheme, a co-simulation scenario can be defined for simulators composition and co-simulation execution. Thus, the requirement ***R6 Unified scheme for model configuration and data exchange*** is addressed. The mosaik scenario API provides well defined functions facilitating simulators and their models composition. Therefore, the requirement ***R7 Simulators composition*** is also satisfied.

When the mosaik scenario API is called, the mosaik scenario module composites simulators and their models and establishes interconnections between them. A data flow is automatically generated during this process, which describes the data

dependencies among the simulators involved in the co-simulation. It serves as the basis for simulators orchestration.

The mosaik scheduler module utilises the data flow generated by the scenario module, to progress all coupled simulators in a discrete-time stepped fashion, coordinate data exchange between them during runtime and keep them synchronised. Together with the scenario module and the simulator manager module, mosaik scheduler is able to orchestrate discrete-time simulators with different step sizes. Therefore, the requirements ***R1. Discrete-time simulator integration***, ***R2. Different temporal resolution support***, and ***R8. Simulators orchestration*** are satisfied.

In the research presented in this thesis, simulators to be integrated into the co-simulation platform need to be encapsulated in FMU format. As mentioned in Section 4.2, the interconnection layer provides FMI co-simulation interfaces to be used by the interoperability layer to composite and orchestrate integrated FMUs. However, mosaik co-simulation middleware cannot call FMI interfaces directly to coordinate FMUs. It can only manipulate simulators via mosaik simulator wrappers, which contain implementation of the mosaik simulator API functions. Therefore, the implementation of the interoperability layer requires not only utilising mosaik middleware but also additional development of an FMI interfaces adapter and mosaik simulator wrappers for simulators to be integrated into the co-simulation platform. The adapter allows mosaik simulator API and FMI API to work together in the same platform and the wrapper contains implementation of the mosaik simulator API functions.

FMI adapter development

Conceptually, mosaik simulator API and FMI API both are co-simulation API, and there are overlaps between them. Therefore, technically an interfaces adapter mapping functions of mosaik simulator API and FMI API is required to make these two interfaces compatible with each other in the same platform. In this research, an FMI adapter was developed which allows mosaik co-simulation middleware to interact with FMUs by calling FMI API functions exposed by the interconnection layer. Currently, the implemented FMI adapter supports the key FMI functions involved in the calling sequence of co-simulation that are listed in Table 4.2. The mapping

between the mosaik simulator API and the FMI API via the FMI adapter is illustrated in Figure 4.5.

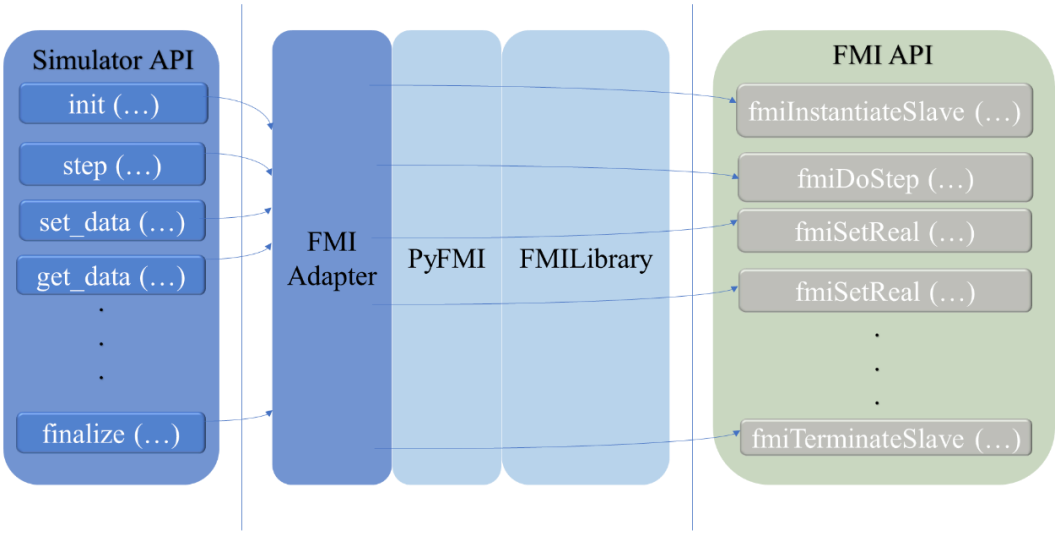



**Figure 4.5: FMI adapter**

Mosaik simulator wrapper development

As mentioned earlier, mosaik simulator wrappers, which contain implementation of simulator APIs are required, to enable the communication between mosaik middleware and FMI API. In the research presented in this thesis, mosaik simulator wrappers were implemented for the integrated time-stepped urban energy simulation tools, including EnergyPlus and No-MASS. These mosaik simulator wrappers were implemented in Python language, which is not restricted by specific programming languages used to develop these urban energy simulation tools.

Each mosaik simulator wrapper includes a formal simulator description of each simulator and its models, as well as implemented mosaik simulator API functions.

Using No-MASS as an example, the approach to develop mosaik simulator wrapper is demonstrated as follows, which includes two steps. The same approach could be adopted to develop the mosaik simulator wrappers for other urban energy simulation tools.

Step 1: Define No-MASS simulator metadata

Describe the models of the simulator by defining a meta data dictionary that tells which model the simulator implements, along with its parameters that can be received by the model and attributes that can be accessed.

**Meta description for the No-MASS simulator**

```
META = {
  'models': {
    'officeAgent': {
      'public': True,
      'params': [
        'model_name', # Name of the idf file
        'model_path', # Path containing idf file
        ],
      'attrs': [
        'Block1:Zone1ZoneAirRelativeHumidity',
        'Block1:Zone1ZoneMeanRadiantTemperature',
        'Block1:Zone1ZoneMeanAirTemperature',
        'Block1:Zone1DaylightingReferencePoint1Illuminance',
        'EnvironmentSiteExteriorHorizontalSkyIlluminance',
        'EnvironmentSiteRainStatus',
        'EnvironmentSiteOutdoorAirDrybulbTemperature',
        'Block1:Zone1BlindFraction',
        'Block1:Zone1NumberOfOccupants',
        'Block1:Zone1LightState',
        'Block1:Zone1WindowState0',
        'Block1:Zone1AverageGains',
      ],
    },
  },
}
```

Step 2: Implement interface functions of mosaik simulator API

Furthermore, essential interface functions of mosaik simulator API have to be implemented, which are used by mosaik simulator manager to control the simulator. In the research presented in this thesis, five mosaik simulator API functions *init()*, *create()*, *set_data()* and *get_data()*, *step()* were implemented for No-MASS simulator wrapper by calling functions exposed by FMI adapter.

In summary, the mosaik core modules and APIs, mosaik simulator wrappers, and FMI adapter are implemented in the interoperability layer, which address the relevant requirements of this layer. An overview of the implemented interoperability layer is illustrated in Figure 4.6.

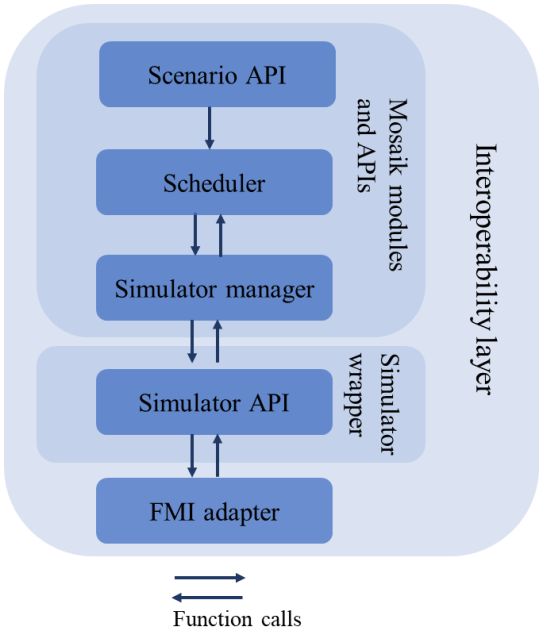



**Figure 4.6: Interoperability layer implementation**

Based on the above discussion of the components in the interoperability layer of COPE, the detail of the interoperability layer developed during the present research is shown in Figure 4.7.

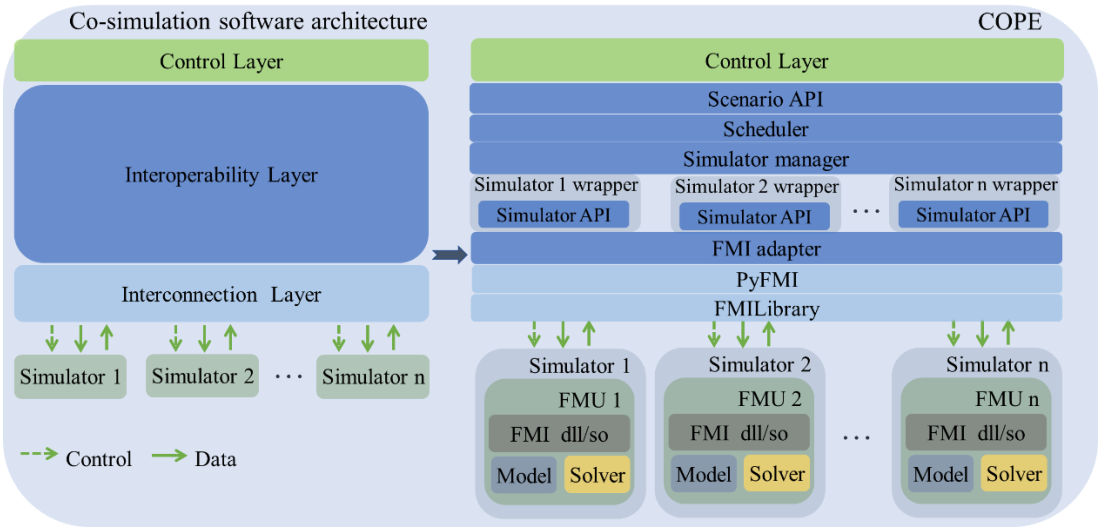



**Figure 4.7: COPE – Detail of the interoperability layer**

In the implemented interoperability layer, the mosaik simulator wrapper together with the FMI adapter facilitate manipulating simulators encapsulated in FMU format by calling python FMI interfaces exposed by the interconnection layer. The mosaik co-simulation middleware provides mosaik simulator API and Scenario API, which serves as the composition API of the co-simulation platform. It facilitates integration of discrete-time simulators via a meta data description of simulators and a set of interface functions. Furthermore, the mosaik scenario API serves as the orchestration API, which provides functions to establish interconnections between simulators and their models. When the scenario API functions are called, the scenario module of mosaik establishes a data flow among models, which is used by the mosaik scheduler and simulator manager to execute the co-simulation of all connected simulators.

## 4.4 Control layer

The control layer of the co-simulation platform is responsible for coordinating integrated simulators to achieve the overall simulation intentions of co-simulation scenarios defined by the users. As mentioned in Chapter 3, requirement to address the control layer is ***R4. Master algorithm orchestrates simulators coupled in Jacobi or Gauss-Seidel***.

To define co-simulation scenario and execute the co-simulation, a master algorithm needs to be implemented to orchestrate the co-simulation and manage the data flow of simulators involved in the co-simulation. The generic scenario definition scheme, provided by mosaik scenario module of the interoperability layer, is used by the master

algorithm to describe which simulators are to be integrated, how they are configured and interconnected with each other, and what data are to be exchanged between them. The orchestration API, e.g. mosaik scenario API, is then called by the master algorithm to establish interconnections between simulators and their models and execute the co-simulation in a discrete-time stepped fashion. To develop a master algorithm, a co-simulation scenario needs to be defined. Functions to run the scenario will need to be included in the master algorithm as well.

To demonstrate how to develop a master algorithm used in COPE, an example of a mater algorithm for a Gauss-Seidel type of weak coupling including two simulators is given below.

Step 1: Define a co-simulation scenario

To define a co-simulation scenario, the topology to connect model instances of the integrated simulators needs to be defined first. For a Gauss-Seidel type of weak coupling, the way to connect two simulators and data to be exchanged at each time step is illustrated in Figure 4.8. The two simulators are executed in sequence. At the beginning of the co-simulation, simulator 1 passes initial data as input to simulator 2. Simulator 2 then starts to run simulation and advances one time step. After the first time step finishes, simulator 2 passes its outputs back to simulator 1 and simulator 1 then starts to run and advances one time step. It then passes output values as input to simulator 2 again. This process will repeat until the end of the co-simulation.

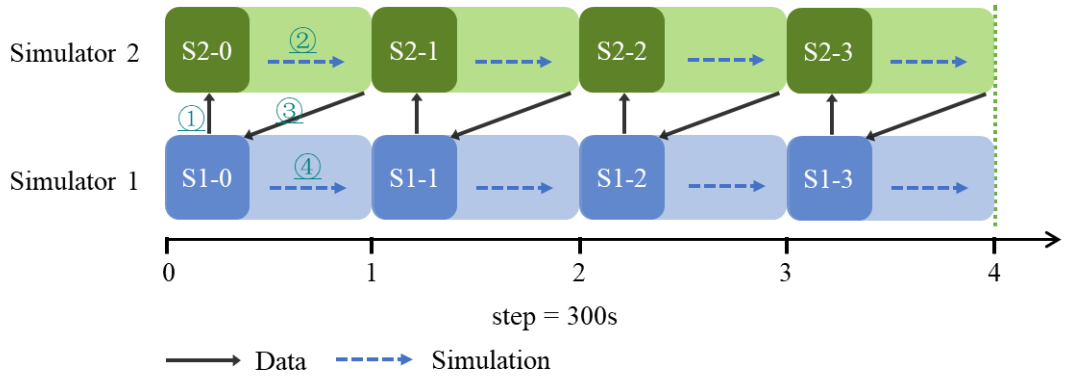



**Figure 4.8: Gauss-Seidel type of weak coupling of two simulators**

Based on the simulation sequence, the configuration information of simulator 1 and simulator 2 and the duration of the simulation are defined in the master algorithm.

The sim_config data structure is used to specify which simulators are available and how to start them. The definition of simulator configuration of this example is shown below.

**sim_config**

```
sim_config = {
    'simulator1': {
        'python': 'mosaik_simulator1.mosaik:Simulator1',
    },
    'simulator2': {
        'python': 'mosaik_simulator2.mosaik:Simulator2',
    },
}
```

In the above example, two simulators are listed. For each simulator, the way to start it is specified. Since the simulators in the example are encapsulated in FMU format and wrapped with mosaik simulator wrapper, they can be manipulated in pure python calls as stated in Section 4.2 and 4.3. The mosaik co-simulation middleware can import the simulators and execute in python processes. This is indicated by the lines 'python': 'mosaik_simulator1.mosaik:Simulator1' and 'python': 'mosaik_simulator2.mosaik: Simulator2'.

The simulation duration of the integrated simulators needs to be specified before the implementation of functions to run the co-simulation of the two simulators through Gauss-Seidel coupling. The duration is specified by starting date and time, and length of the simulation period.

**Simulation duration**

```
START = '2015-01-01 00:00:00'
END = 365 * 24 * 3600  # 1 year
```

After the co-simulation scenario configuration, a world object needs to be instantiated within the provided simulator configuration information.

```
world = mosaik.World(sim_config)
```

The world object holds simulation state of all integrated simulators, which is used by functions in the master algorithm to be defined in the next step.

Step 2: Define functions to run the scenario

In this example, we implemented one function create_scenario(world) in the master algorithm to create executable scenario to conduct co-simulation. This function includes all essential functions to start the two simulators, instantiate models within the simulators, connect the model instances of different simulators to establish the data flow between them, and execute the co-simulation.

**Simulation duration**

```
# Start simulators
sim1= world.start('Simulator1', step_size=600, tStart=0, tStop=END, sim_params={}, model_config=[('cid','sim1model', 1, {'model_path': '/work/fmu_repo/sim1/sim1model/', 'model_name': 'sim1FMU.fmu'})])
sim2= world.start(' Simulator2', step_size=600, tStart=0, tStop=END, sim_params={}, model_config=[('cid','sim2model', 1, {'model_path': '/work/fmu_repo/sim2/sim2model/', 'model_name': 'sim2FMU.fmu'})])
# Instantiate models
 sim1model = sim1.sim1model.create(1)
sim2model = sim2.sim2model.create(1)
# Connect sim1 to sim2
world.connect(sim1model[0], sim2model[0], 'var1', 'var2', async_requests=True)
```

Once the executable scenario is created by referring topology to connect model instances of the two simulators and data flow is established between the model instances in the co-simulation scenario definition stage, the co-simulation could be started by calling world.run(). How long the simulation needs to run could be specified here.

```
world.run(until=END)
```

In this example master algorithm, the two simulators are executed sequentially as the data flow connection diagram illustrated in Figure 4.8. At each time point, the output value of simulator 1 is firstly passed to simulator 2. Then simulator 2 advances one time-step and feeds its output value back to simulator 1. Once it's done simulator 2 can advance one time-step. This process continues until the end of the simulation. For co-simulation with different simulation scenarios, similar master algorithms could be developed by referring to the example master algorithm demonstrated in this section. Master algorithms could be developed to support co-simulation of simulators coupled in either Jacobi type or Gauss-Seidel type.

The implementation of the control layer of the co-simulation platform is shown in Figure 4.9.

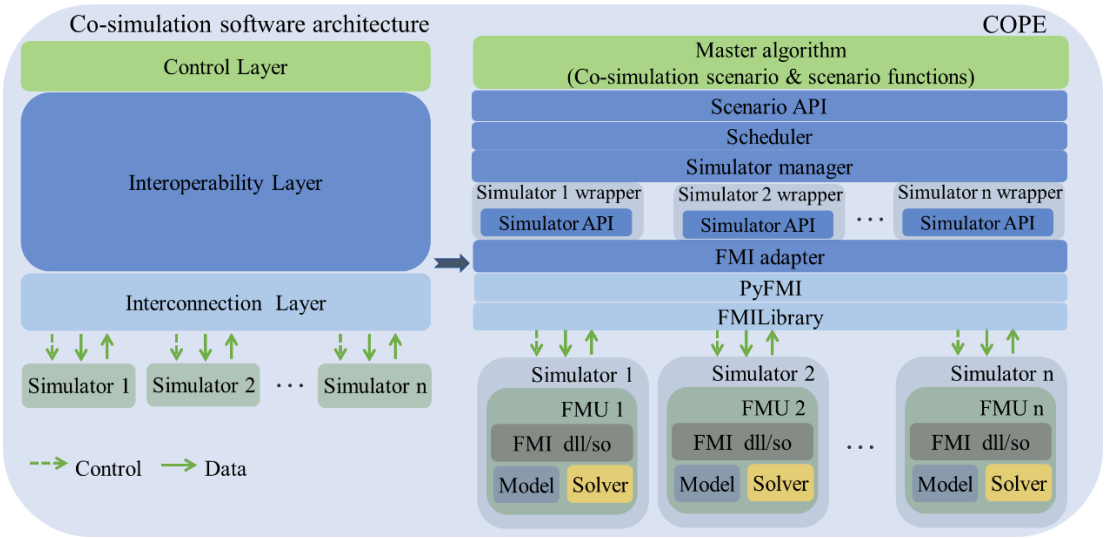



**Figure 4.9: Control layer implementation**

## 4.5 The developed platform - COPE

The development of each layer of COPE has been presented in the above sections. Figure 4.10 shows the detailed internal structure of the platform, which was

implemented in a layered modular software architecture and addresses all the key requirements of the urban energy co-simulation identified in Chapter 2.

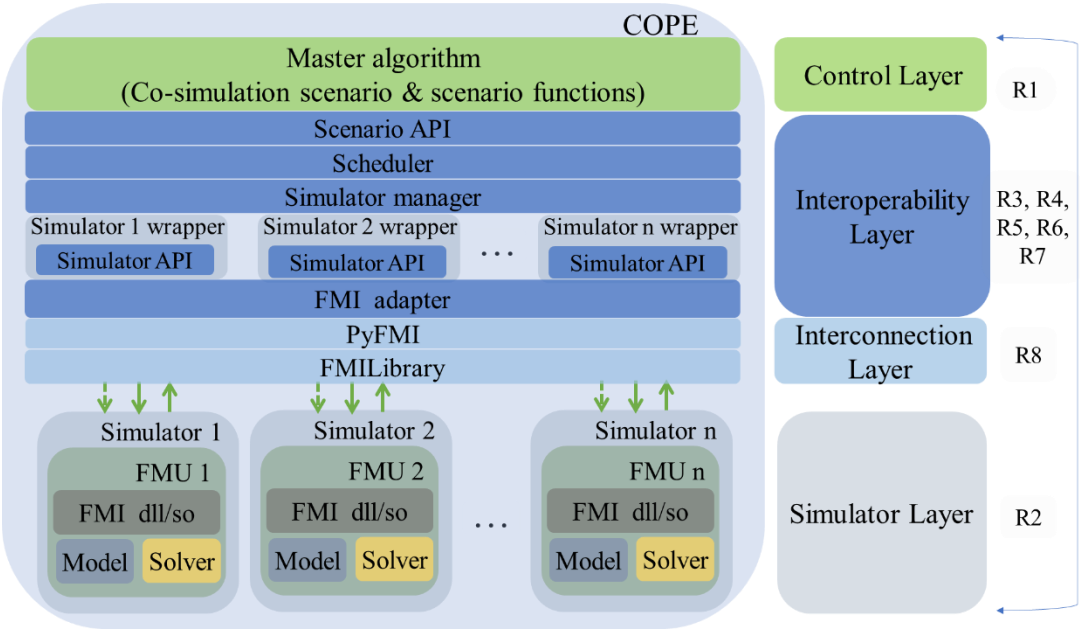



**Figure 4.10: The co-simulation platform for Ecological-Urban - COPE**

COPE, as shown in Figure 4.10**,** requires integrated simulation tools to be compliant with FMI standard and allows simulation in a discrete-time (fixed-step size) manner. With FMI support, simulation tools can be exported in FMU format in the simulator layer.

The interconnection layer of COPE enables it to load and interact FMUs with C FMI interfaces and provides Python native calls to the interoperability layer. The layered modular design of the platform allows to add additional FMI software package to the interconnection layer to support integrating simulation tools with FMI interfaces programmed in programming language other than C.

For each simulator integrated into the co-simulation platform, mosaik simulator API functions must be implemented in a mosaik wrapper. The mosaik simulator wrapper together with FMI adapter in the interoperability layer enable the communication between mosaik co-simulation middleware and the FMI interfaces of the integrated simulators exposed by the interconnection layer. The mosaik middleware composite and orchestrate simulators and build interconnections between simulators and their models. When the scenario API functions are called, the scenario module of mosaik

establishes a data flow among models, which is used by the mosaik scheduler and simulator manager to execute the co-simulation.

By calling mosaik scenario API provided by the interoperability layer, the master algorithm of the control layer is able to establish interconnections between the integrated simulators and their models and execute the co-simulation in a discrete-time stepped fashion. Thus, the overall simulation purpose of a co-simulation scenario designed by urban energy simulation end users could be achieved, and the results could be used for decision making in urban energy planning to improve energy efficiency and urban sustainability.

## 4.6 COPE application process

To facilitate other end users to use COPE to conduct simulation for a specific scenario, a process is explained in this section with step by step guidance. The application process of COPE for a co-simulation scenario is illustrated in Figure 4.11. A brief description of what needs to be done at each step is given below.

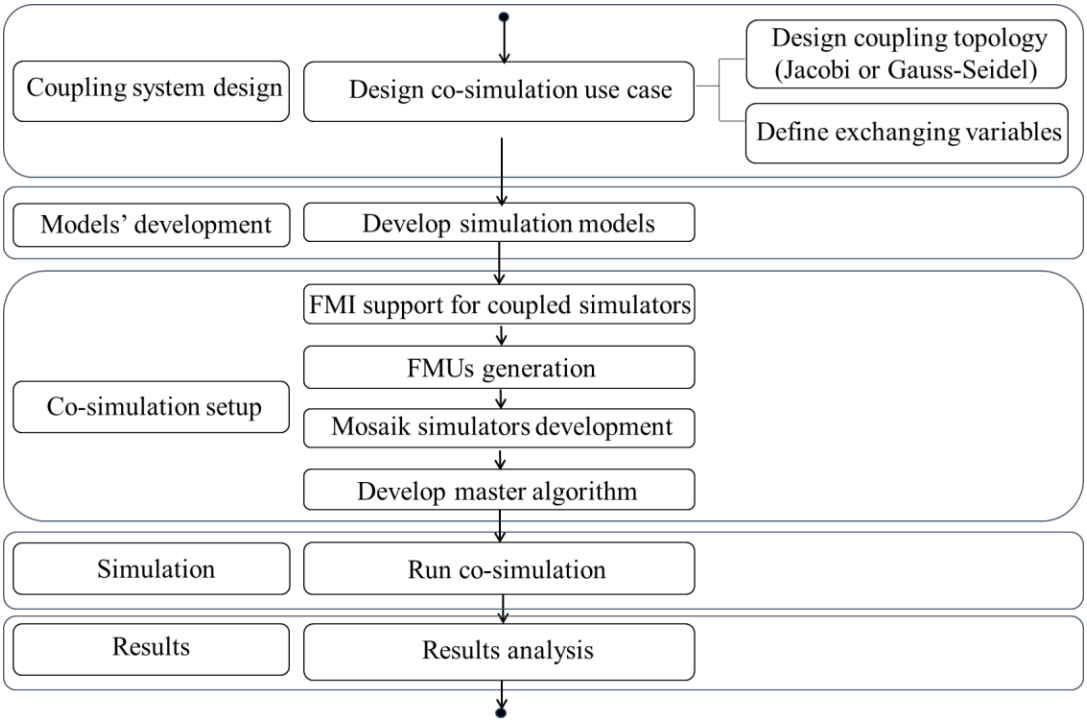



**Figure 4.11: COPE application process for a co-simulation**

- Coupling system design

The first step is to design the co-simulation, which includes defining the co-simulation scenario and simulation purpose. The simulation tools to be integrated into COPE will

be selected based on the simulation purpose. The coupling typology is decided based on relationship between the simulation tools. The coupling scheme is either Jacobi pattern or Gauss-Seidel pattern based on communication patterns between the simulation tools. In addition, variables to be exchanged among the integrated tools are also defined.

- Model development

In this step, simulation models for each tool are collected or developed, for the simulation purpose.

- Co-simulation setup

Most co-simulation development work is conducted in this step.

If the integrated simulators do not support FMI, then they need to be made FMI-compliant by developing FMI co-simulation API functions, generating configuration and model XML files, and generating FMU zip archive. An example was presented in Section 4.1.

After the models FMU are generated, an interfaces adapter and mosaik simulator wrappers will need to be developed. In addition, a master algorithm needs to be developed by defining co-simulation scenario and functions to run the scenario. The coupling type is also reflected in the scenario.

- Simulation

After all setup is completed, the co-simulation could be started by running the master algorithm. Depends on the time resolution and duration, area scale, complexity of the simulation and the computation hardware capability, this process could take up to a few hours or days.

The co-simulation process orchestrated by the master algorithm is illustrated in Figure 4.12.

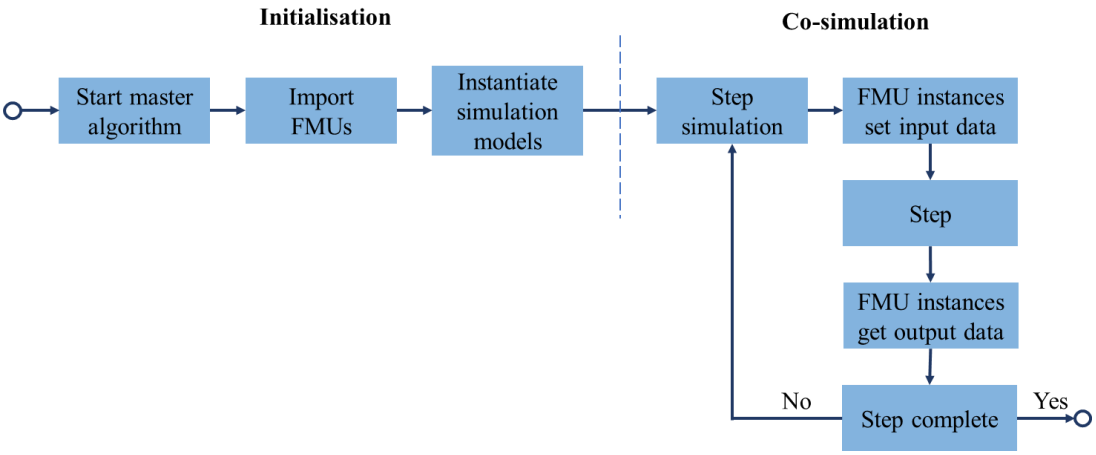



**Figure 4.12: Co-simulation process**

- Results analysis

After the simulation is completed, results could be then analysed and used for urban energy planning or decision making.

The development of COPE, as well as its application for a co-simulation, is presented in this chapter. Correspondingly, Obj. III (Develop an urban energy co-simulation platform based on the conceptual architecture) is achieved in this chapter. In the next chapter, the application of COPE through two case studies will be presented. The purpose is to evaluate the modular co-simulation architecture as well as COPE in order to validate the co-simulation approach developed in the research.

# 5 CASE STUDIES OF THE CO-SIMULATION APPROACH

Chapter 3 introduced a conceptual co-simulation architecture and Chapter 4 presented the implementation of COPE based on such architecture. The COPE approach addressed all the research requirements raised in Chapter 2 through a generic and scalable co-simulation for urban energy modelling.

In this chapter, COPE is applied to two illustrative case studies: single building co-simulation and multiple buildings co-simulation. The aim of the case studies is twofold: functional evaluation and validation of the co-simulation architecture; and application demonstration of the co-simulation platform.

The first case study is presented in Section 5.1 as a proof-of principle demonstration and functionality evaluation of the platform. The fundamental functionalities, such as synchronisation and interaction between the co-simulation platform and coupled simulation tools are evaluated through a single building energy co-simulation. Through the evaluation, the conceptual co-simulation architecture, on which the platform was based, is validated.

Furthermore, Section 5.2 presents the second case study to demonstrate the extendibility and applicability of the co-simulation platform. This case study is used to demonstrate COPE for a more complex multiple buildings energy modelling with multiple simulation models.

Together the two case studies give an overview of the main capabilities of COPE. A conclusion of the functional evaluation and application demonstration of the platform is then given in Section 5.3.

## 5.1 Case 1: Single building co-simulation

To validate the conceptual co-simulation architecture as well as evaluate the fundamental functionalities of COPE, such as integration and orchestration, a simple case with only one shoe box office building was designed. With this building energy system, the building energy consumption is affected by its occupants' stochastic behaviours. Equally, the occupants' behaviours are impacted by the building environment such as temperature, shading, etc.

To simulate its energy behaviour as well as impact of its occupants' stochastic behaviours on the building performance, two simulation tools were selected and integrated. They are the detailed whole-building energy simulation tool EnergyPlus (Crawley, Lawrie et al. 2001) and the multi-agent stochastic simulation tool No-MASS (Chapman, Siebers et al. 2018). In this case, they are orchestrated to simulate energy behaviour of the shoe box office and impact of its occupants' stochastic behaviours on the building performance (Figure 5.1).

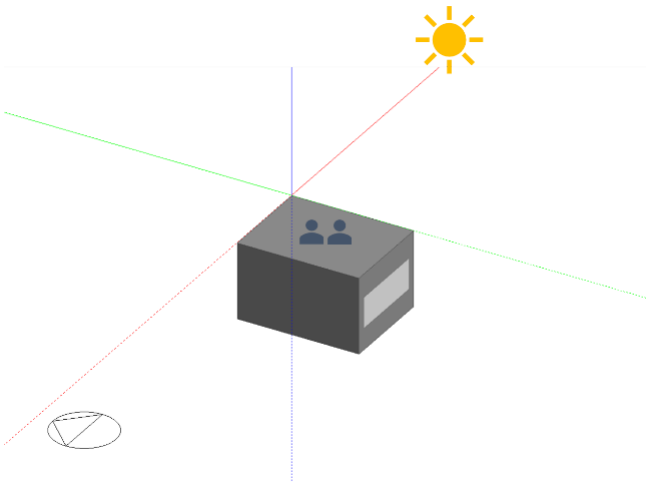


**Figure 5.1: Shoe box office energy system**

EnergyPlus, on its own, performs energy simulation of a building using deterministic rules and schedules that represent occupant interactions with the building, such as window openings, shading, lights and occupants' presence. When coupled with No-MASS, EnergyPlus simulates the building's energy flows and passes environmental

variables to No-MASS. No-MASS then parses the environmental conditions to predict occupants' behaviours and interacts with shading devices, windows and lighting. It then returns the number of occupants in a zone, their metabolic gains, appliance gains, the window status, the blind shading fraction, and the lighting status to EnergyPlus. Thus, the impact of stochastic interactions of occupants on building performance simulation is examined.

In a previous work, Chapman coupled EnergyPlus and No-MASS to conduct co-simulation by using FMU import feature of EnergyPlus for studying the impact of multi-agent stochastic behaviours on building performance simulation (Chapman, Siebers et al. 2018). In such an approach, EnergyPlus acts as the co-simulation master and No-MASS is imported as a salve. Similar to Chapman, some other researchers such as Stifter, Schwalbe et al. (2013), Thomas, Miller et al. (2014) and Miller, Thomas et al. (2017), Gurecky, De Wet et al. (2020) also used FMU import approach to couple two simulators. There was no integration platform, and the simulation structure was limited to only two simulators. However, the FMU import method is a verified coupling technique for conducting co-simulation between two simulators. Therefore, the simulation results of EnergyPlus and No-MASS in Chapman's FMU import approach will be referred to as a baseline simulation in this research.

In this case, the coupling of EnergyPlus and No-MASS to simulate a hypothetical shoe box office and its occupants' stochastic behaviours was implemented in both Chapman's FMU import approach and the COPE approach. The results will be examined and compared.

The major steps involved in this case study include coupling system design, model development, co-simulation setup, simulation and results analysis. They are illustrated in Figure 5.2.

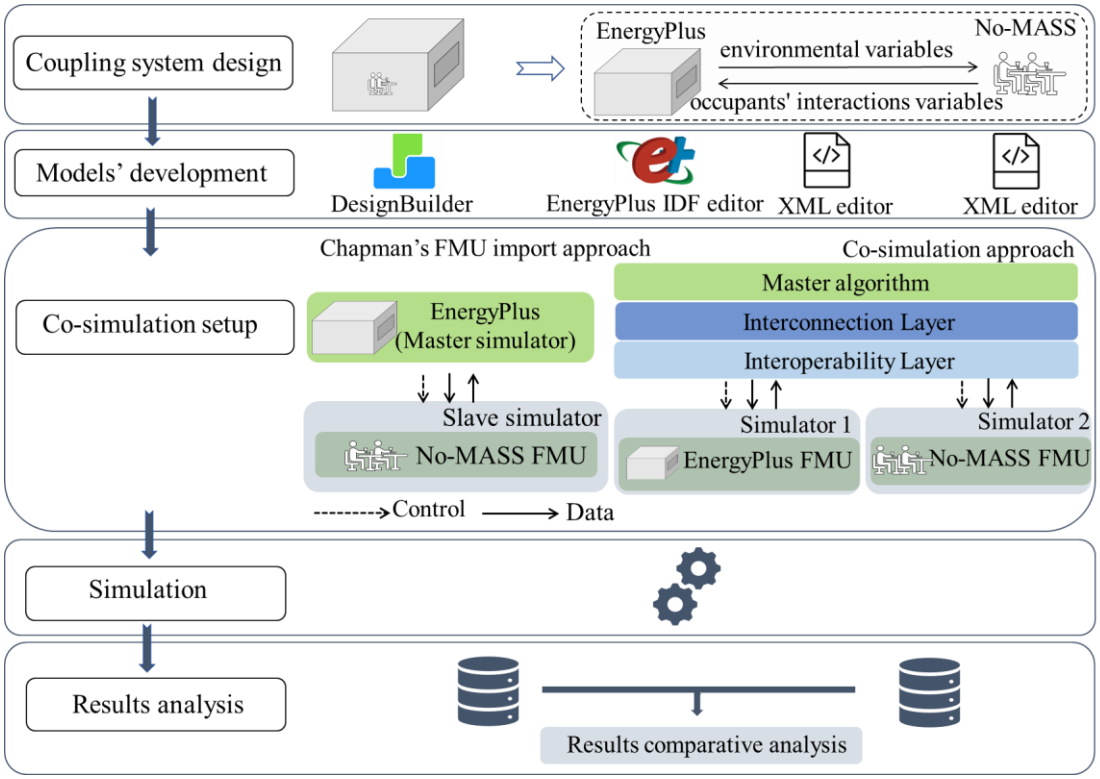



**Figure 5.2: Single shoe box building co-simulation case study process**

The first step of the single shoe box building co-simulation case study process is the coupling system design (Section 5.1.3). In this step, the exchanging variables and the data flow diagram of EnergyPlus and No-MASS, as well as the occupants' stochastic behaviours of the shoe box office, will be defined. After that, the shoe box EnergyPlus model and the corresponding stochastic simulation No-MASS model will be developed (Section 5.1.4). In the co-simulation setup stage, necessary FMUs will be generated and then mosaik simulator wrappers and the master algorithm will be developed (Section 5.1.5). After that, the occupants' stochastic behaviours of the shoe box office were simulated through both the Chapman's FMU import approach and COPE (Section 5.1.6). In the last step, the simulation results of COPE were compared with that from Chapman's FMU import approach to demonstrate that the synchronisation and interaction between the urban energy co-simulation platform and coupled co-simulation components could work as expected (Section 5.1.7).

Before presenting the major steps of the case, the orchestration and co-simulation process of the FMU import approach and COPE will be presented in Sections 5.1.1 and 5.1.2.

### 5.1.1 Chapman's FMU import approach

The connection between EnergyPlus and No-MASS of the FMU import approach adopted by Chapman is shown in Figure 5.3 and simulation input files for EnergyPlus and No-MASS are described in Table 5.1.

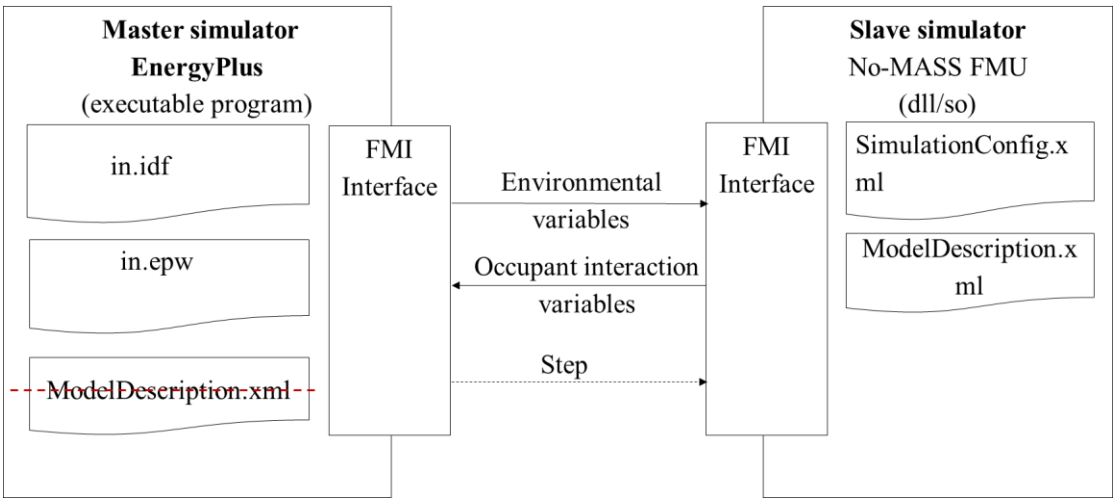



**Figure 5.3: EnergyPlus and No-MASS orchestration diagram (FMU import approach - based on (Chapman, Siebers et al. 2018))**

**Table 5.1: Simulation input files for EnergyPlus and No-MASS (FMU import approach - based on (Chapman, Siebers et al. 2018))**

| Simulation tool | File | Description |
|---|---|---|
| EnergyPlus | in.idf | The building configuration file containing the data describing the building and environmental variables need to be exchanged during the co-simulation process. |
| | in.epw | The weather data file |
| | ModelDescription.xml | In Chapman's approach, for EnergyPlus the building and environmental variables need to be exchanged during the co-simulation process are defined in the in.idf file. ModelDescription.xml file |

| | | |
|---|---|---|
| | | should not be part of simulation input files for EnergyPlus shown in Figure 5.3. |
| No-MASS | SimulationConfig.xml | It contains information about the occupants that is used to build the agent population, and the subsequent processing of an agent activity profile (a series of parameters defining the socio-demographic characteristics of the agent, i.e. gender, age, income level, etc) that is used to calculate the probability of an activity taking place at each time step (Chapman, Siebers et al. 2018). It also defines the window and shading model coefficients for each model, allowing for diversity between occupants and models to be represented as needed (Chapman, Siebers et al. 2018). |
| | ModelDescription.xml | It contains occupants' interactions variables need to be exchanged during the co-simulation process. |

In Chapman's FMU import approach, No-MASS FMU is imported as a slave simulator and EnergyPlus acts as the master simulator, which controls data exchange and synchronisation between the master simulator and the slave simulator. The co-simulation process between EnergyPlus and No-MASS in the FMU import approach is illustrated in Figure 5.4.

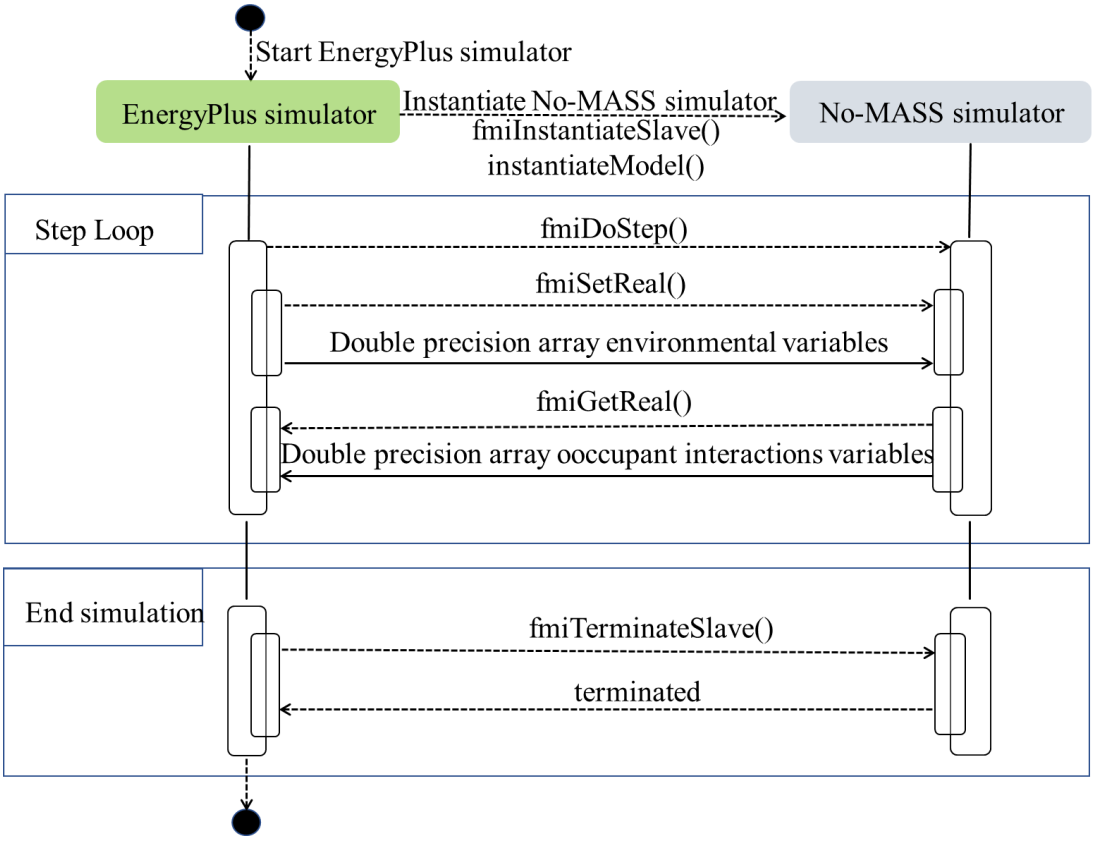



**Figure 5.4: EnergyPlus and No-MASS co-simulation process (FMU import approach)**

In this FMU import approach, the simulation is started by EnergyPlus. After reading its building configuration IDF file and weather data EPW file, EnergyPlus locates No-MASS FMU, extracts it and then instantiates No-MASS to start the co-simulation process. At each time step, EnergyPlus simulates the single building's energy flows and passes its environmental variables as an array of double precision values defined in the ExternalInterface section of the EnergyPlus in.idf file to No-MASS. Then fmiDoStep function of No-MASS is called by EnergyPlus to perform a calculation based on the environmental variables received from EnergyPlus and then returns results of stochastic interactions of occupants to EnergyPlus, as an array of double precision values specified in the No-MASS model description XML file. These values from No-MASS are used by EnergyPlus in the next time step to overwrite corresponding values defined by the predefined deterministic rules. These include the number of occupants in a zone, their metabolic gains, appliance gains, the window status, the blind shading fraction, the lighting status and heating setpoints. Consequently, the energy consequences of occupants' interactions are considered

during simulation of building's energy flows. This process continues until the end of the simulation. Once all time steps are completed the No-MASS FMU model instance is terminated by EnergyPlus and the co-simulation process is ended.

### 5.1.2 COPE approach

In contrast to Chapman's approach, the generic and scalable urban energy co-simulation approach presented in this thesis couples simulators in COPE. In this approach, simulation tools, such as EnergyPlus and No-MASS, are encapsulated in FMU format and are integrated into COPE as shown in Figure 5.5.

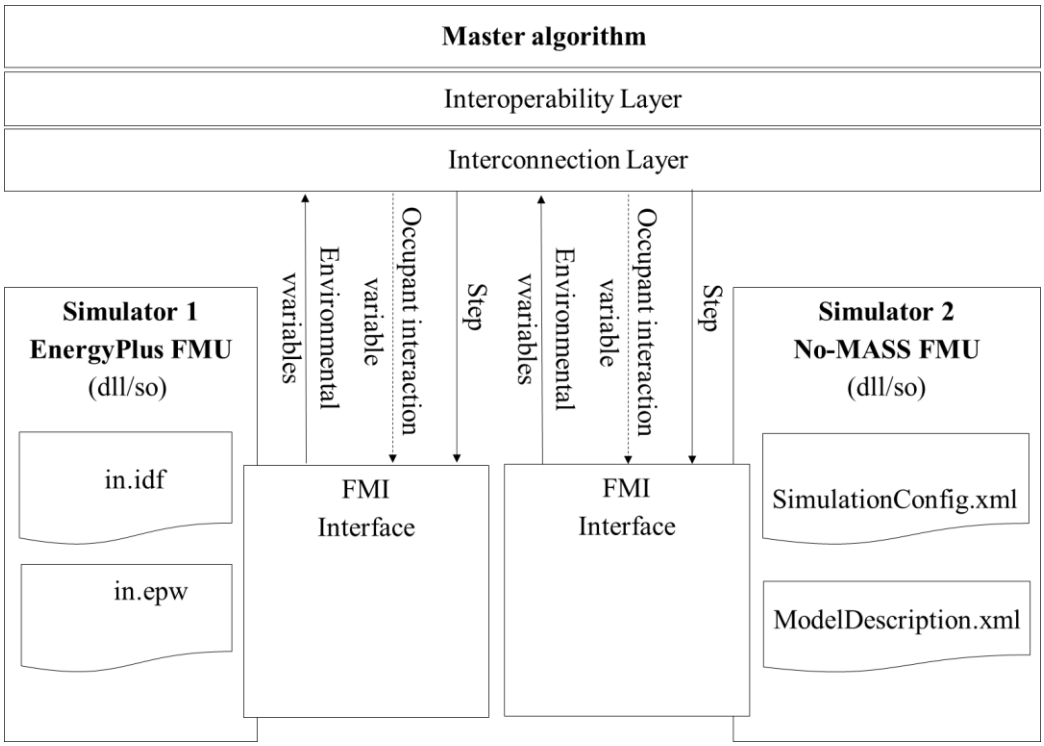



**Figure 5.5: EnergyPlus and No-MASS orchestration diagram (COPE approach)**

In the COPE approach, both EnergyPlus FMU and No-MASS FMU can be integrated as slaves and orchestrated by a master algorithm to control the data exchange between EnergyPlus and No-MASS and the synchronisation of them. The co-simulation process of EnergyPlus and No-MASS of this approach is illustrated in Figure 5.6.

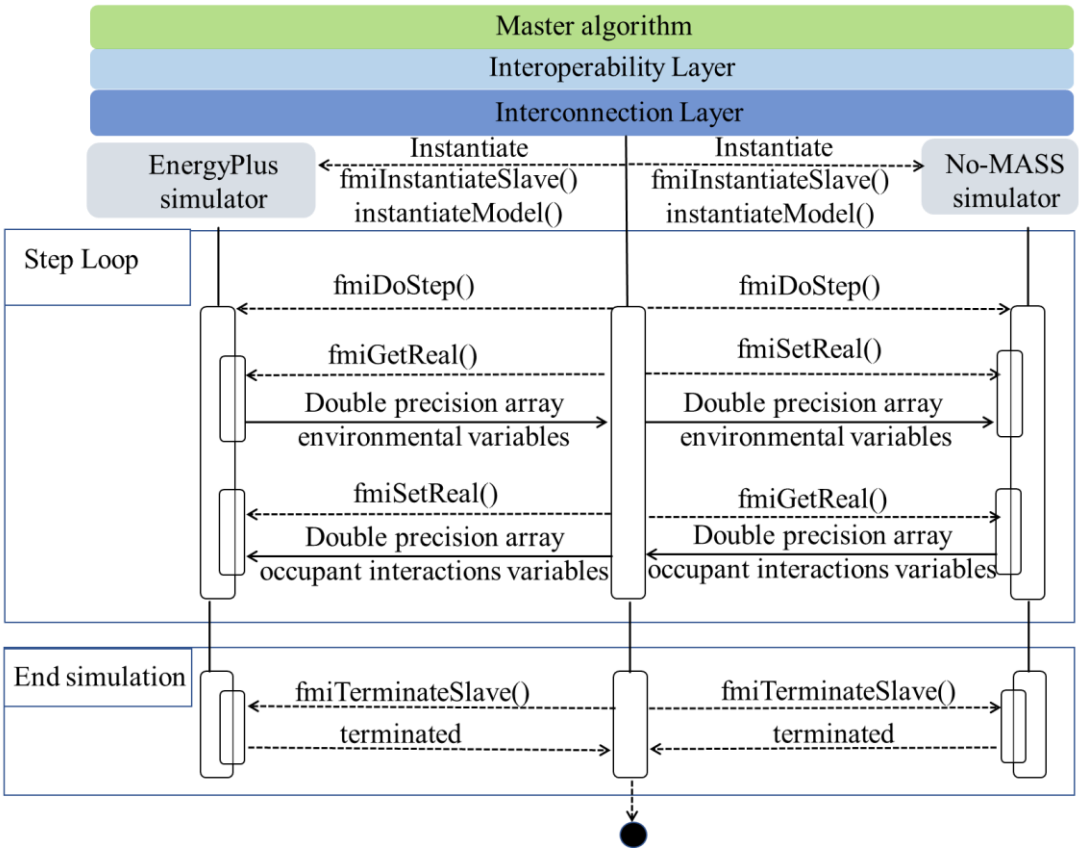



**Figure 5.6: EnergyPlus and No-MASS co-simulation process (COPE approach)**

In the COPE approach, the simulation is started by the master algorithm of the co-simulation platform. Firstly, the EnergyPlus FMU and No-MASS FMU are located, extracted, and then instantiated by the master algorithm. During the co-simulation process, the master algorithm calls fmiDoStep function of EnergyPlus to simulate the building's energy flows at each time step. The values of the environmental variables of EnergyPlus are extracted by the master algorithm and passed to No-MASS. Then the master algorithm calls fmiDoStep function of the No-MASS to perform a calculation of stochastic interactions of occupants based on the environmental variables received from EnergyPlus. These values from No-MASS are then extracted by the master algorithm and passed to EnergyPlus at the next time step to overwrite corresponding values based on predefined deterministic rules then resolve the energy consequences of these interactions when simulating the building's energy flows. This process continues until the end of the simulation. Once all time steps are completed, the FMU model instances of EnergyPlus and No-MASS are terminated by the master algorithm and the co-simulation process is ended.

### 5.1.3 Coupling system design

In the coupling system design step, the environmental variables of the hypothetical shoe box office and its occupants' stochastic behaviours are specified, which constitute the exchanging variables and data flow between the EnergyPlus and No-MASS as shown in Figure 5.7.

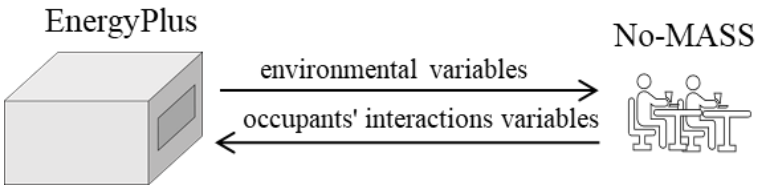



**Figure 5.7: Coupling system design**

In this case, EnergyPlus provides environmental variables of the building, which include zone humidity, indoor radiant temperature, zone air temperature, indoor illuminance, horizontal sky illuminance, rain status, outdoor air temperature. Some of these variables will be used by No-MASS to simulate occupants' behaviour, and it generate variables related to the presence of the occupants, such as the number of occupants in a zone, the window status, the lighting status and heat gains due to the occupants' presence. These variables constitute the input and output variables of the two simulation tools. The input and output variables of EnergyPlus and No-MASS for shoe box office simulation are outlined in Table 5.2 and Table 5.3.

**Table 5.2: EnergyPlus output variables (No-MASS input variables)**

| Variables | Description |
|---|---|
| ZoneAirRelativeHumidity [%] | It represents the air relative humidity after the correct step for each zone. |
| ZoneMeanRadiantTemperature [°C] | The mean radiant temperature is the interior wall temperature by the emittance of the interior surfaces. Quite often the emittance is assumed to be 1. |

| | |
|---|---|
| ZoneMeanAirTemperature [°C] | The zone mean air temperature is the average air temperature of a zone at the system time step. |
| DaylightingReferencePoint1Illuminance [lux] | The total daylight illuminance at the first reference point from all of the windows in a daylight zone. |
| SiteExteriorHorizontalSkyIlluminance [lux] | Illuminance from sky solar radiation on an unobstructed horizontal plane at the earth's surface. |
| SiteRainStatus [1 or 0] | This field indicate whether it is raining outside or not (1=yes, 0=no). |
| SiteOutdoorAirDrybulbTemperature [°C] | This is the outdoor dry-bulb temperature. |

**Table 5.3: No-MASS output variables (EnergyPlus input variables)**

| Variable | Description |
|---|---|
| NumberOfOccupants | The number of occupants in a zone. |
| AverageGains [Watt/m2] | The sensible metabolic heat gains due to the occupants' presence. |
| GainsAppliance | The appliance gains. |
| BlindFraction | The blind shading fraction. |
| WindowState [0 or 1] | The widows are CLOSE (0) or OPEN (1). |
| LightState [0 or 1] | The lights are OFF (0) or ON (1). |

The co-simulation exchanging variables defined in Case 1 are listed in Appendix 4. After the definition of the input and output variables of the shoe box office and its occupants, the data flow diagram of coupling between EnergyPlus and No-MASS on the shoe box office and its occupants' stochastic behaviours are defined as shown in Figure 5.8.

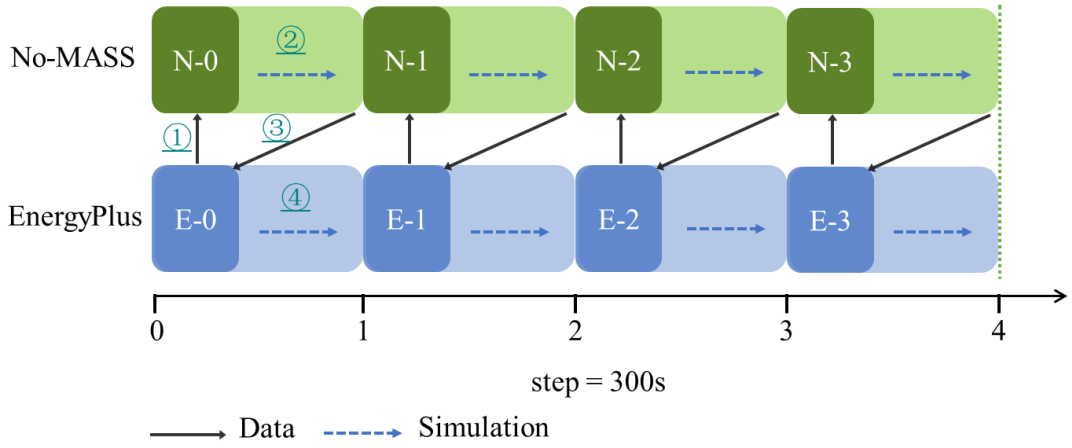



**Figure 5.8: Data flow between EnergyPlus and No-MASS during co-simulation**

During the co-simulation process, EnergyPlus simulates the building's energy flows and passes environmental variables to No-MASS at each time step (①). Consequently, No-MASS parses the environmental conditions to predict agents' behaviours and interactions with shading devices, windows and lighting (②). It then returns the number of occupants in a zone, their metabolic gains, appliance gains, the blind shading fraction, the window status, and the lighting status to EnergyPlus (③). After receiving outputs from No-MASS, EnergyPlus resolves the energy consequences of these interactions when simulating the building's energy flows in the next step (④). This process continues until the end of the simulation. Due to the window, shading and location/presence models used within No-MASS, 5 minutes time step for co-simulation is recommended as longer time step may overestimate the implication of the occupants' window control interactions (e.g. the duration that windows remain open) (Chapman, Siebers et al. 2018).

### 5.1.4 Model development

After coupling system design, the EnergyPlus shoe box office building model was developed. The layout of the shoe box office building modelled by EnergyPlus is shown in Figure 5.9.

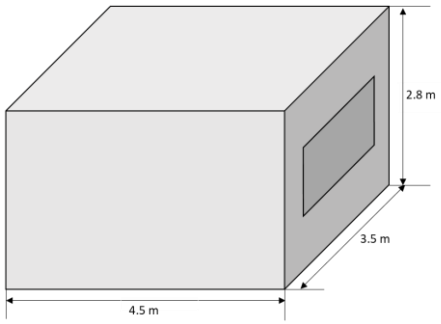



**Figure 5.9: Shoe box building**

In common with a single zone case studied by Chapman, details such as glazing ratios, heating set-points, and constructions of the shoe box office are given in Table 5.4 and Table 5.5. The weather file was taken from EnergyPlus weather data website ((DOE) 2018) giving the location of Geneva, Switzerland (46$^{o}$ 25' North, 6$^{o}$ 13' East).

**Table 5.4: Shoe box building zone details**

| Zone | Area [$m^2$] | Volume [$m^3$] | Gross Wall Area [$m^2$] | Glazing ratio % | Lighting [$W/m^2$] | Setpoint Temp [$^{o}C$] |
|---|---|---|---|---|---|---|
| Office | 11 | 39 | 47 | 6 | 20 | 21 |

**Table 5.5: Shoe box building construction materials**

| Location | Layer | Thickness (m) | Material |
|---|---|---|---|
| External Wall | Outer | 0.1 | Brick |
| External Wall | 2 | 0.07 | XPS extruded |
| External Wall | 3 | 0.1 | Concrete Block |
| External Wall | Inner | 0.01 | Gypsum Plaster |

| | | | |
|---|---|---|---|
| U-Value | | | 0.37 |
| Internal Partition | Outer | 0.02 | Gypsum Plaster |
| Internal Partition | 2 | 0.1 | Air Gap |
| Internal Partition | Inner | 0.02 | Gypsum Plaster |
| U-Value | | | 2.86 |
| Ground Floor | Outer | 0.13 | Urea Formaldehyde Foam |
| Ground Floor | 2 | 0.1 | Cast Concrete |
| Ground Floor | 3 | 0.07 | Floor Screed |
| Ground Floor | Inner | 0.03 | Timber Flooring |
| U-Value | | | 0.26 |
| Floor | Outer | 0.10 | Cast Concrete |
| U-Value | | | 4.7 |
| Pitched Roof | Outer | 0.02 | Clay Tile |
| Pitched Roof | 2 | 0.02 | Air Gap |
| Pitched Roof | Inner | 0.005 | Roofing Felt |
| U-Value | | | 4.97 |

Just as in Chapman's approach, the standard deterministic schedules were referred to as the baseline to judge the stochastic co-simulation of EnergyPlus and No-MASS. For this case study, as shown in Table 5.6, the deterministic rules that govern occupancy for the office on weekdays were defined as 00:00 until 07:00, [0%], until 08:00, [25%], until 09:00, [50%], until 12:00, [100%], until 14:00, [75%], until 17:00, [100%], until 18:00, [50%], until 19:00, [25%], until 24:00, [0%]. The percentage number in the bracket means the chances of occupants in the buildings. It's assumed that the office is vacant during the weekend period. The operation rules of the office equipment and lighting were defined in a way similar like the occupancy schedule.

For the windows operation, it was defined to be opened when occupants are present and the indoor temperature exceeds 24 $^{o}$C.

**Table 5.6: Shoe box office schedules**

| Occupancy | Equipment | Lighting | WindowState |
|---|---|---|---|
| Office_OpenOff_Occ,<br>Fraction,<br>Through: 31 Dec,<br>For: Weekdays<br>SummerDesignDay,<br>Until: 07:00, 0,<br>Until: 08:00, 0.25,<br>Until: 09:00, 0.5,<br>Until: 12:00, 1,<br>Until: 14:00, 0.75,<br>Until: 17:00, 1,<br>Until: 18:00, 0.5,<br>Until: 19:00, 0.25,<br>Until: 24:00, 0,<br>For: Weekends,<br>Until: 24:00, 0,<br>For: Holidays,<br>Until: 24:00, 0,<br>For: WinterDesignDay<br>AllOtherDays,<br>Until: 24:00, 0; | Office_OpenOff_Equip,Fraction,<br>Through: 31 Dec,<br>For: Weekdays SummerDesignDay,<br>Until: 07:00, 0.05,<br>Until: 20:00, 1,<br>Until: 24:00, 0.05,<br>For: Weekends,<br>Until: 24:00, 0.05,<br>For: Holidays,<br>Until: 24:00, 0.05,<br>For: WinterDesignDay AllOtherDays,<br>Until: 24:00, 0; | Office_OpenOff_Light,<br>Fraction,<br>Through: 31 Dec,<br>For: Weekdays SummerDesignDay,<br>Until: 07:00, 0,<br>Until: 19:00, 1,<br>Until: 24:00, 0,<br>For: Weekends,<br>Until: 24:00, 0,<br>For: Holidays,<br>Until: 24:00, 0,<br>For: WinterDesignDay AllOtherDays,<br>Until: 24:00, 0; | Schedule:Compact, 20001,Any Number,<br>Through: 12/31,<br>For: Weekdays Weekends SummerDesign Days,<br>Until: 24:00, 24,<br>For: WinterDesignDays,<br>Until: 24:00, 0,<br>For: AllOtherDays,<br>Until: 24:00, 24; |

DesignBuilder, a comprehensive graphical user interface for EnergyPlus, was used to generate the shoe box office model based on the geometry and construction shoe box properties shown in Table 5.4 and Table 5.5 and the deterministic rules shown in Table 5.6. Using the EnergyPlus IDF file export function of DesignBuilder, the EnergyPlus shoe box model was generated. The simulation configuration and the model description files were defined for the corresponding No-MASS model development. The simulation configuration file contains information such as the seed number for controlling the randomness of the simulations, the time step, the simulation period, buildings, zones, agents and appliances. The model description file was defined to contain the set of exposed variables that need to be passed between the No-MASS and EnergyPlus.

### 5.1.5 Co-simulation setup

Once the simulation models were developed, the co-simulation environment needs to be set up for coupling and simulating building energy of the shoe box office and its occupants' stochastic behaviours. This is shown in Figure 5.10.

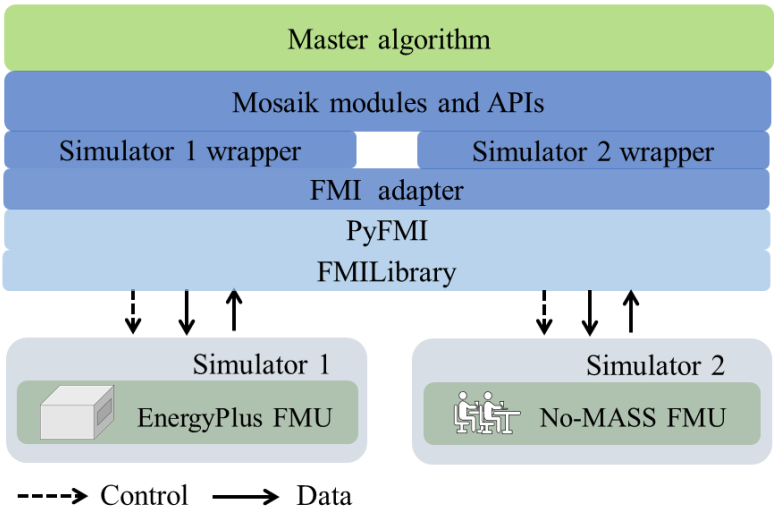



**Figure 5.10: EnergyPlus and No-MASS single building co-simulation**

For this case, EnergyPlusToFMU was used to convert the EnergyPlus shoe box office building model into an FMU. The corresponding No-MASS FMU was generated by archiving the compiled library FMI.so with the model description and simulation configuration files. The layout of the EnergyPlus and No-MASS FMUs are shown as below.

**EnergyPlus shoe box office FMU**

```
in.fmu/
|
|-------binaries/
|         |-------linux64/
|                   |------- in.so
|-------modelDescription.xml
|-------resources/
          |-------Energy+.idd
          |-------idf-to-fmu-export-prep-linux
          |-------in.epw
          |-------in.idf
          |-------variables.cfg
```

**No-MASS FMU**

```
agentFMU.fmu/
|
|-------binaries/
|         |-------linux64/
|                   |-------FMI.so
|-------modelDescription.xml
|-------SimulationConfig.xml
```

Then mosaik simulator wrappers were developed. To orchestrate co-simulation of EnergyPlus and No-MASS models, a master algorithm was implemented in this case according to the data flow diagram defined in Figure 5.8. The master algorithm is shown in Appendix 7.

### 5.1.6 Simulation

After the models were developed and the master algorithm was implemented, the simulation is ready to run. As mentioned earlier, simulation results from Chapman's FMU import approach were planned to be used as a baseline to verify the fundamental functionalities of the co-simulation platform COPE.

In order to do so, firstly EnergyPlus heating demand simulation results of the shoe box office were calculated based on standard deterministic rules and schedules. then from the stochastic interactions of occupants by coupling EnergyPlus and No-MASS in Chapman's FMU import approach. For the stochastic simulation using No-MASS, the stochastic models of No-MASS can be set a seed value for controlling the randomness of the simulations, e.g. window opening, shading, occupancy, etc. 50 replicates of the co-simulation of EnergyPlus and No-MASS with 50 randomly selected seed numbers were performed to show the impact of stochastic interactions of occupants on building performance. All simulations were run on an annual basis. The co-simulation flow of Chapman's approach is shown in Figure 5.11.

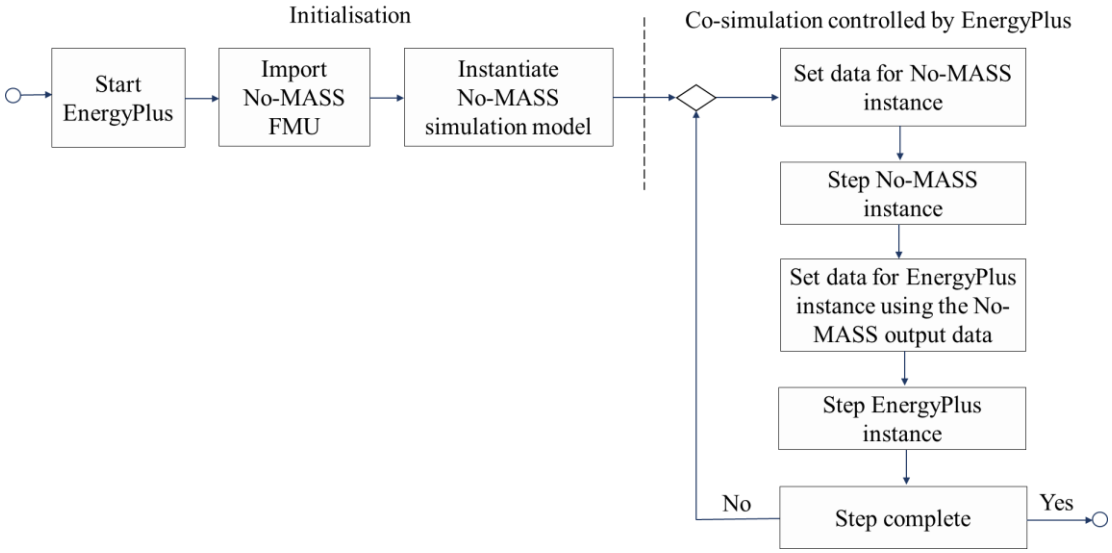



**Figure 5.11: EnergyPlus and No-MASS single building co-simulation flow diagram (Chapman's FMU import approach)**

Following the simulation in Chapman's FMU import approach, the co-simulation of EnergyPlus and No-MASS of the same shoe box office was executed in COPE for 50 replicates as well. Both simulations used exactly the same condition and occupants' behaviours. All simulations were also on an annual basis. The co-simulation flow of the COPE approach is shown in Figure 5.12.

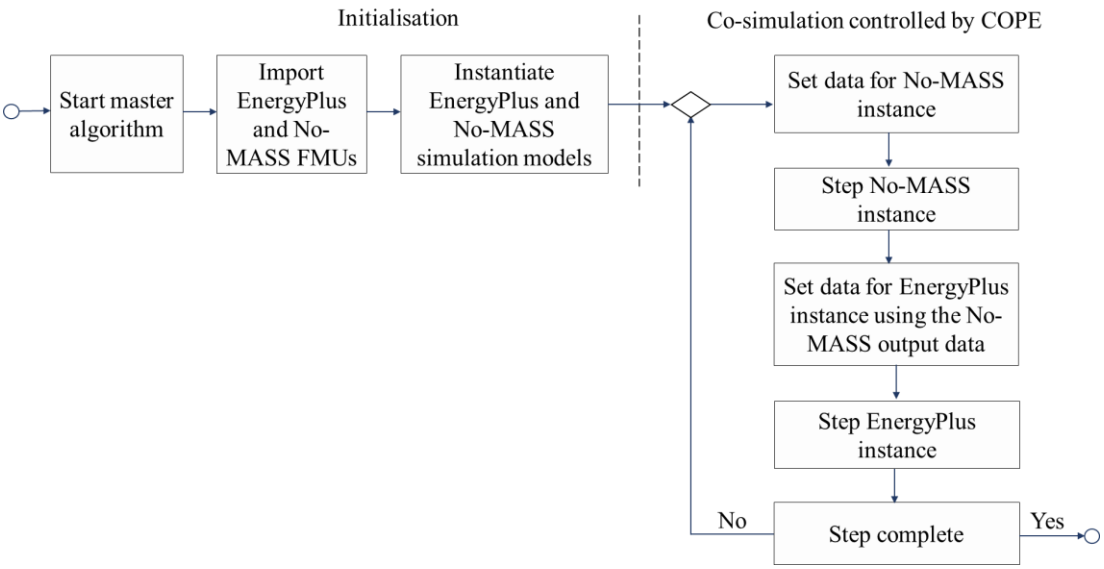



**Figure 5.12: EnergyPlus and No-MASS single building co-simulation flow diagram (COPE approach)**

### 5.1.7 Results analysis

In order to evaluate whether the synchronisation and interaction between COPE and the coupled simulation tools work as intended, the simulation results from the COPE approach were compared with that from Chapman's coupling approach.

The simulation results of the shoe box office building in Chapman's FMU import approach and COPE are shown in monthly box plots in Figure 5.13 and Figure 5.14. In each of the figures below, the mean and standard deviation of heating demand for each month are presented.

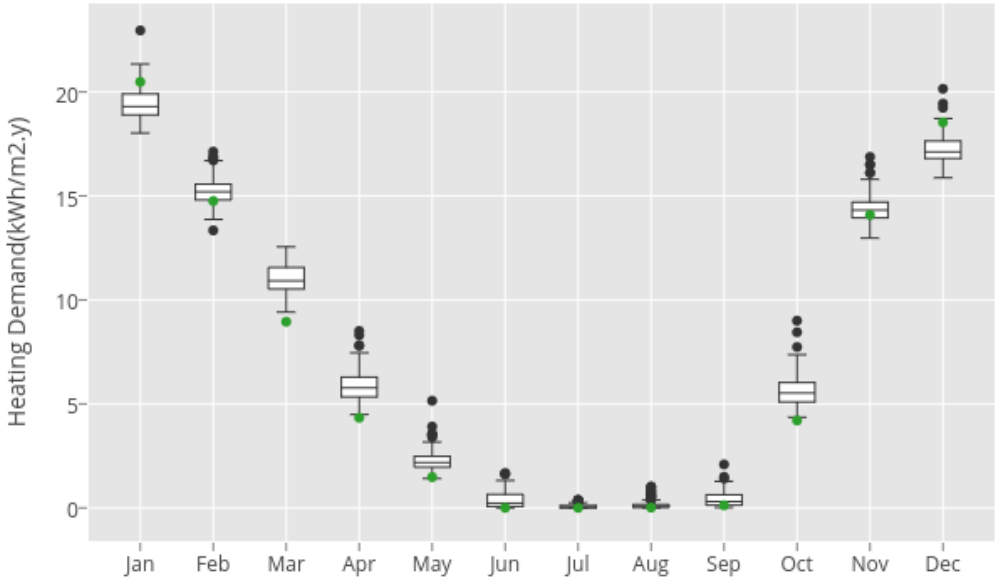



**Figure 5.13: Monthly heating demand of the shoe box office in Geneva (Chapman's approach). (Boxplot) Stochastic agent platform 50 replicates, (Green point) Single deterministic simulation**

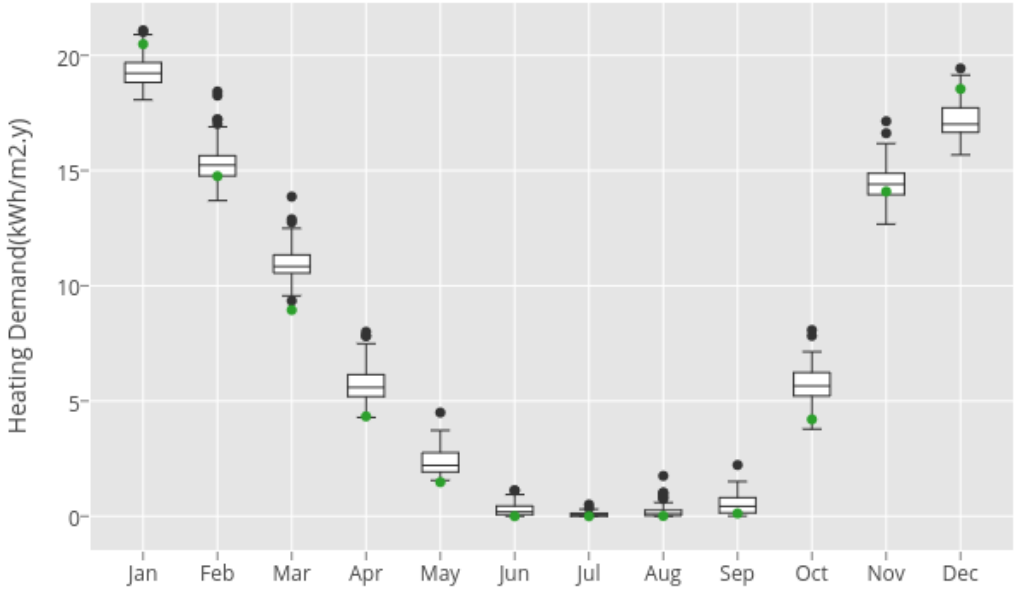



**Figure 5.14: Monthly heating demand of the shoe box office in Geneva (COPE). (Boxplot) Stochastic agent platform 50 replicates, (Green point) Single deterministic simulation**

From the results, it can be observed that COPE achieved the equivalent heating demand of each month with the interaction of occupants, comparing to the results

obtained from the Chapman's FMU import approach. By checking the simulation results step by step, it was verified that the master algorithm and coupled simulation tools synchronised and interacted as intended. Simulation results of Chapman's approach and the COPE approach show that the monthly heating demand results help us understand the variations in energy demands arising from occupants' interactions in one year. The heating demand in months except January and December for the stochastic simulations is higher than the deterministic simulation. This is due to occupant's interaction with the windows to refresh indoor air.

A Debian 10 Linux virtual machine allocated 16.0 GB RAM, which is hosted by a computer operating on Windows 10 with Intel Core i5-6440HQ CPU @ 2.60GHz and 32.0 GB RAM, was used to run the simulation for case 1. Regarding single building co-simulation time, the FMU import approach took about 34 minutes for a 50 replicates simulation. In comparison, it took about 55 minutes for COPE to run 50 replicates. The longer time taken by COPE is due to the orchestration process by the mosaik master algorithm to control the data exchange between the EnergyPlus shoe box FMU and the corresponding No-MASS FMU through interconnection layer and interoperability layer.

Though the co-simulation time of COPE is longer than the direct coupling, it provides a versatile and easy extendable solution. The FMU import approach adopted by Chapman is limited to coupling EnergyPlus with another simulation tool and only two simulators can be coupled at the same time. However, the co-simulation platform presented in this thesis has no such limitation, and can couple multiple simulators acting as slaves in the platform orchestrated by a master algorithm.

In the next section, a more complex energy system with multiple buildings will be presented to demonstrate COPE's capability of coupling multiple simulation models.

## 5.2 Case 2: Multiple buildings co-simulation

To demonstrate the capability of multiple simulators coupling of the co-simulation platform for a more complex energy system, the second case was designed to simulate the energy behaviour of three shoe box office buildings and the impact of their occupants' stochastic behaviours on the building performance. The system is shown in Figure 5.15.

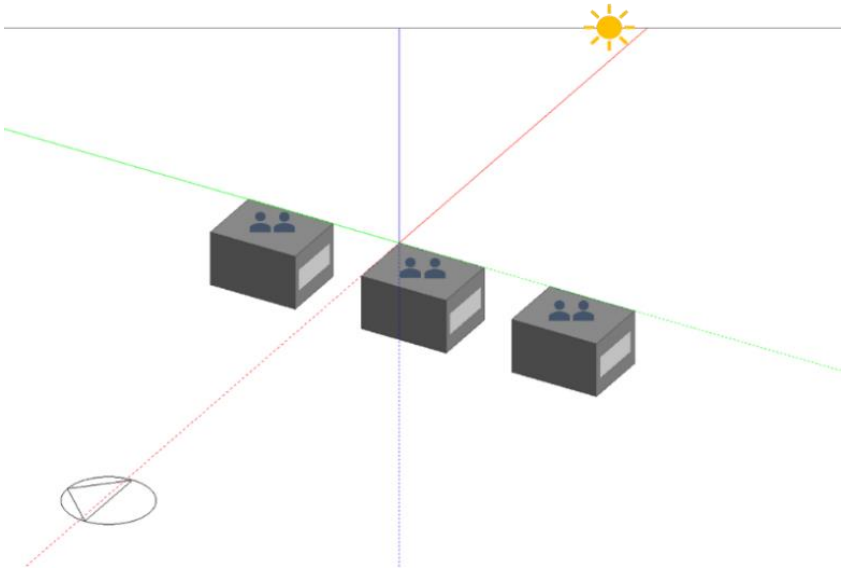


**Figure 5.15: Multiple shoe box office buildings energy system**

Like the first use case, the major steps involved in this case study include coupling system design, models' development, co-simulation setup, simulation and results analysis as shown in Figure 5.16.

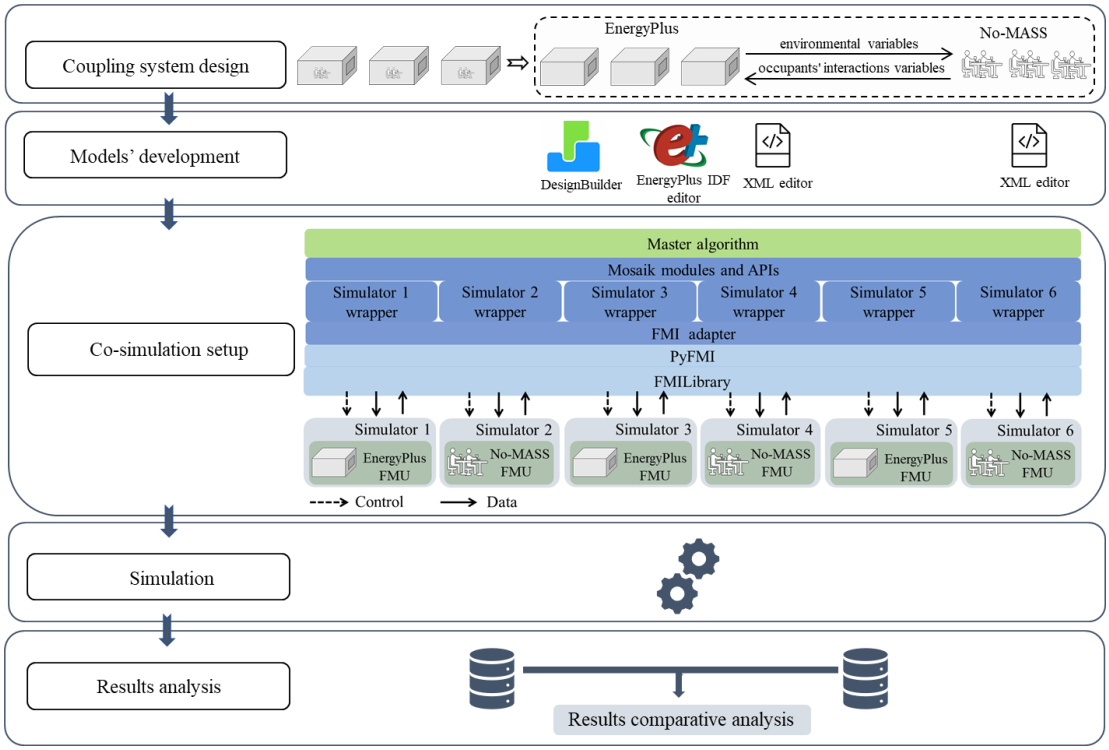



**Figure 5.16: Multiple shoe box buildings co-simulation case study process**

This test case focuses on energy simulation of multiple buildings relates to the stochastic behaviours of their occupants. Coupling system design is the first step of

this case study. In this step, the exchanging variables and coupling topology of the three shoe box offices and corresponding occupants' stochastic behaviours were defined. In the second step, three EnergyPlus shoe box building models and the corresponding No-MASS stochastic simulation model which simulates the presence, activities and related behaviours of occupants of the three shoe box buildings, were developed. In this case study, the SimulationConfig.xml files of the coupled No-MASS models contain a series of parameters, e.g. gender, age, income level, etc., to define agent activity profiles. As a result, there was a discrepancy between the three buildings which was attributed to differences in the occupants' characteristics. After that, necessary FMUs were generated and then mosaik simulator wrappers and the master algorithm were developed in the co-simulation setup stage. Next, the co-simulation of the three EnergyPlus shoe box simulators and their occupants' stochastic behaviours simulator was executed by using COPE. In the last step, the simulation results were analysed to show that COPE is capable of coupling multiple simulators.

### 5.2.1 Coupling system design

The coupling system design is the first phase of the co-simulation case study process. As shown in Figure 5.17, the environmental variables of the three hypothetical shoe box offices and corresponding occupants' stochastic behaviours in each office were specified in this stage.

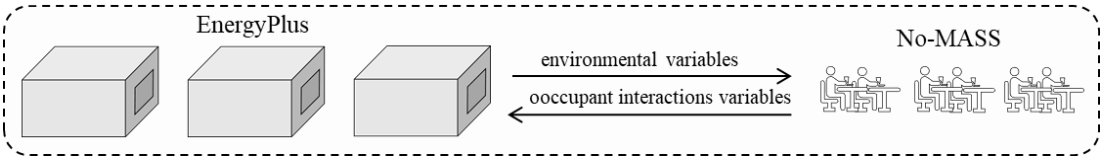



**Figure 5.17: Coupling system design**

For this case study, the environmental variables of the three hypothetical shoe box offices and their occupants' stochastic behaviours were specified same as the first use case. Each EnergyPlus simulator provides environmental variables of the building, like zone humidity, indoor radiant temperature, zone air temperature, indoor illuminance, horizontal sky illuminance, rain status, and outdoor air temperature, to the No-MASS simulator. The No-MASS simulator then simulates occupants' behaviours in each of the shoe box buildings and returns each EnergyPlus simulator variables related to the presence of the occupants in the corresponding shoe box office,

like the number of occupants in a zone, the window status, the lighting status and heat gains due to the occupants' presence. The input and output variables of EnergyPlus simulators and the No-MASS simulator for this case study were set same as the first use case which are illustrated in Table 5.2 and Table 5.3. The co-simulation exchanging variables defined in Case 2 are listed in Appendix 4.

The data flow diagram of the coupled shoe box office buildings and related occupants was then defined as shown in Figure 5.18, which depicts co-simulation sequence for the coupled three EnergyPlus simulators and the No-MASS simulators which simulate occupants' behaviours in the three buildings.

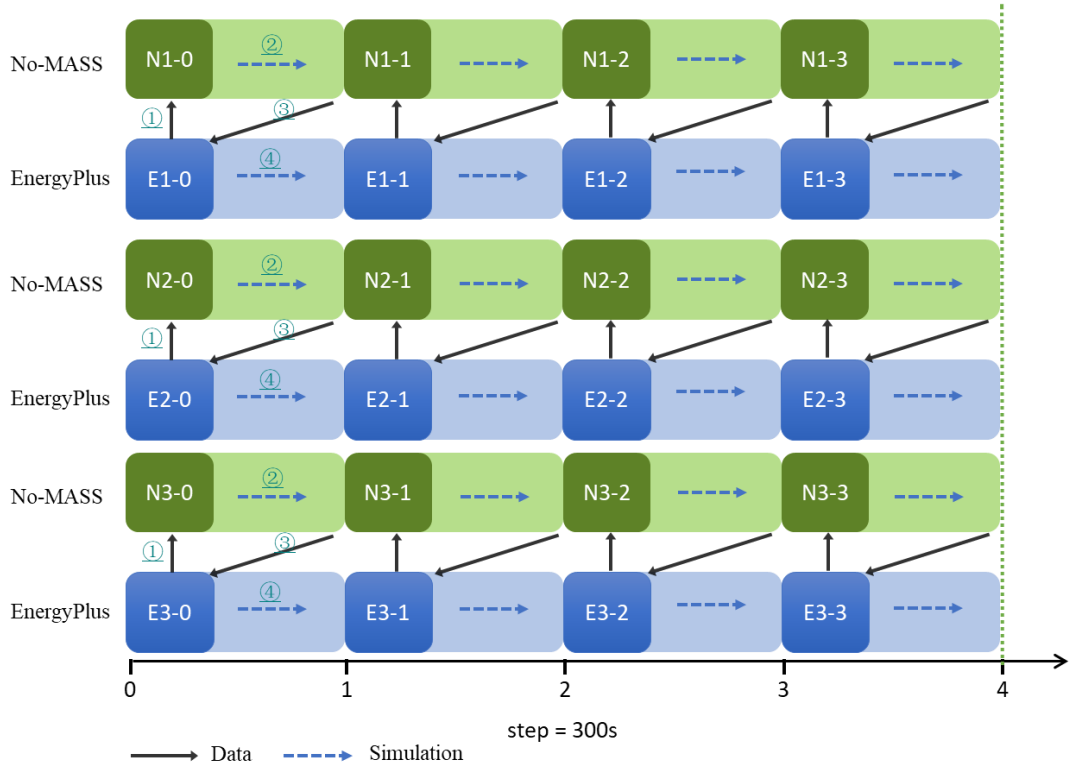



**Figure 5.18: Data flow of multiple EnergyPlus building simulators and corresponding No-MASS simulator associated with the buildings during co-simulation**

For the multiple simulators co-simulation of EnergyPlus and No-MASS, during the simulation process each EnergyPlus shoe box building simulator simulates energy flows within the building then passes environmental variables to the No-MASS simulator at each time step (①). Subsequently, each No-MASS simulator parses the environmental conditions of related shoe box office to predict its agents' behaviours that interact with shading devices, windows and lighting (②). Then the number of

occupants in a zone, their metabolic gains, appliance gains, the blind shading fraction, the window status, and the lighting status of each shoe box office are returned to corresponding EnergyPlus simulator (③). After receiving output from the related No-MASS simulator, each EnergyPlus simulator resolves the energy consequences of these interactions during the building's energy flows simulation in the next step (④). This process carries on till the end of the simulation process. Like the first use case, the timestep for the co-simulation process was set to 5 minutes.

### 5.2.2 Model development

Following the coupling system design, the three EnergyPlus shoe box office building models and corresponding No-MASS models were developed respectively.

For this case study, the distance between shoe box buildings in the 1x3 matrix was set to 3.5 metres considering the influence of the distance between buildings (Long, Alalwany et al. 2015). The layout design of the three shoe box office buildings is shown in Figure 5.19.

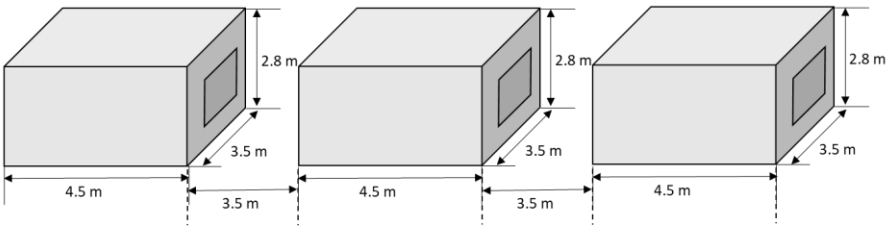



**Figure 5.19: Three shoe box buildings layout**

The shoe box office buildings in the 1x3 matrix were modelled by considering sky view factor and shading geometry, with same glazing ratios, heating set-points and constructions of the shoe box office in the first use case presented in Table 5.4 and Table 5.5. For the three office buildings, the deterministic rules that govern occupancy on weekdays were all defined as 00:00 until 07:00, [0%], until 08:00, [25%], until 09:00, [50%], until 12:00, [100%], until 14:00, [75%], until 17:00, [100%], until 18:00, [50%], until 19:00, [25%], until 24:00, [0%]. The percentage number in the bracket means the chances of occupants in the buildings. It's assumed that all the three office buildings are vacant during the weekend period. The rules to operate the equipment and lighting in the three buildings were defined in a way similar like the occupancy schedule. The windows of the three office buildings were defined to be opened when occupants are present, and the indoor temperature exceeds 24 $^{o}$C. The

weather file for the shoe box buildings was taken from EnergyPlus weather data website ((DOE) 2018) giving the location of Finningley (53º 48’ North, 1º 0’ West), a location about 40 miles from Nottingham, UK.

DesignBuilder was used to create each of the three shoe box office models considering adjacent buildings as component blocks as shown in Figure 5.20. Then corresponding EnergyPlus input files (*.idf) were then generated by using the export function of DesignBuilder to generate EnergyPlus idf files.

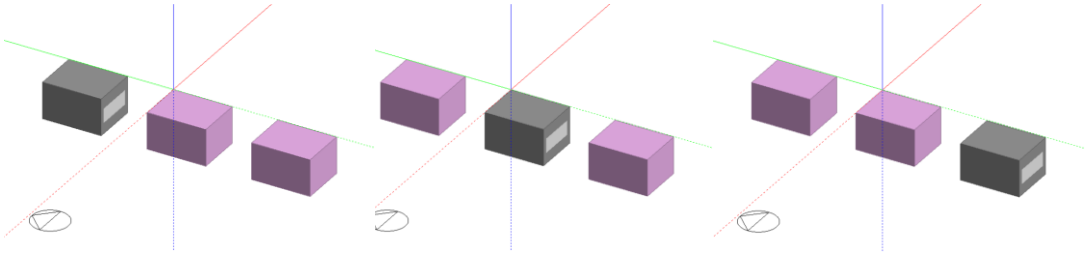


**Figure 5.20: Three shoe box building DesignBuilder models**

After the EnergyPlus shoe box office building models were built, the No-MASS models for the three shoe box offices were developed. The simulation configuration file of the No-MASS models includes data such as the seed numbers for controlling the randomness of the simulations, the time step, the simulation period, buildings, zones, agents and appliances for the three shoe box buildings. The model description file contains the set of exposed variables of the three shoe box buildings that need to be passed to the EnergyPlus models.

### 5.2.3 Co-simulation setup

When all the simulation models were ready, the multiple simulators co-simulation environment was then set up in order to co-simulate the three shoe box offices and their occupants’ stochastic behaviours. The co-simulation environment is illustrated in Figure 5.21.

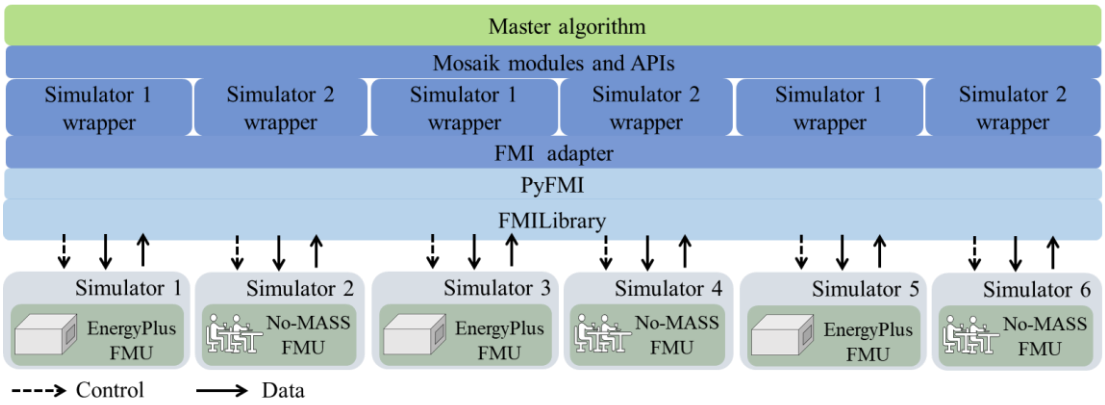



**Figure 5.21: EnergyPlus and No-MASS three buildings co-simulation**

EnergyPlusToFMU was used to convert the three EnergyPlus shoe box office building models into FMUs. The corresponding No-MASS FMUs containing the No-MASS models for the three shoe box offices were then created by archiving the compiled library FMI.so with the model description file and simulation configuration file. The layout of the EnergyPlus and No-MASS FMUs are shown as follows.

| **EnergyPlus shoe box office 1 FMU** | **EnergyPlus shoe box office 2 FMU** | **EnergyPlus shoe box office 3 FMU** |
|---|---|---|
| in.fmu/<br>\|<br>\|----binaries/<br>\|  \|----linux64/<br>\|  \|----in.so<br>\|----modelDescription.xml<br>\|----resources/<br>\|----Energy+.idd<br>\|----idf-to-fmu-export-prep-linux<br>\|----GBR_FINNINGLEY_IWEC.epw | in.fmu/<br>\|<br>\|----binaries/<br>\|  \|----linux64/<br>\|  \|----in.so<br>\|----modelDescription.xml<br>\|----resources/<br>\|----Energy+.idd<br>\|----idf-to-fmu-export-prep-linux<br>\|----GBR_FINNINGLEY_IWEC.epw | in.fmu/<br>\|<br>\|----binaries/<br>\|  \|----linux64/<br>\|  \|----in.so<br>\|----modelDescription.xml<br>\|----resources/<br>\|----Energy+.idd<br>\|----idf-to-fmu-export-prep-linux<br>\|----GBR_FINNINGLEY_IWEC.epw |

| | | |
|---|---|---|
| |----id_bdg_1.idf | |----id_bdg_2.idf | |----id_bdg_3.idf |
| |----variables.cfg | |----variables.cfg | |----variables.cfg |

**No-MASS FMU**

```
agentFMU.fmu/
|
|-------binaries/
|         |-------linux64/
|                   |-------FMI.so
|-------modelDescription.xml
|-------SimulationConfig.xml
```

Mosaik simulator wrappers for each of the EnergyPlus FMUs and the No-MASS FMUs were then developed. After that, a master algorithm was developed to orchestrate the co-simulation of EnergyPlus and No-MASS models by referring to the data flow diagram defined in Figure 5.18. The master algorithm for this case study is similar like the single building co-simulation algorithm shown in Appendix 7 with adjustment to six simulators.

### 5.2.4 Simulation

Following the models and the master algorithm development, the simulation is ready to run and demonstrate how COPE orchestrate more complex ecosystems: e.g. multiple buildings co-simulation.

The co-simulation of multiple EnergyPlus shoe box buildings models and the corresponding No-MASS model was executed in COPE for 20 replicates on an annual basis. For each of the replicates, the stochastic models of the No-MASS, e.g. window opening, shading, occupancy, etc., for the three office buildings were set with a randomly selected seed number for controlling the randomness of each simulation. All simulations were performed on an annual basis.

### 5.2.5 Results analysis

The FMU import approach adopted by Chapman's is limited to coupling EnergyPlus with another simulation tool and only two simulators can be coupled directly in this way. COPE has no such limitation, which can couple as many simulation tools acting as slaves in the platform and orchestrated by a master algorithm. Therefore, a more generic co-simulation solution is achieved.

The co-simulation results of the shoe box office building 1, 2 and 3 are shown in monthly box plots in Figure 5.22, Figure 5.23 and Figure 5.24, which present the mean and standard deviation of heating demand for each month.

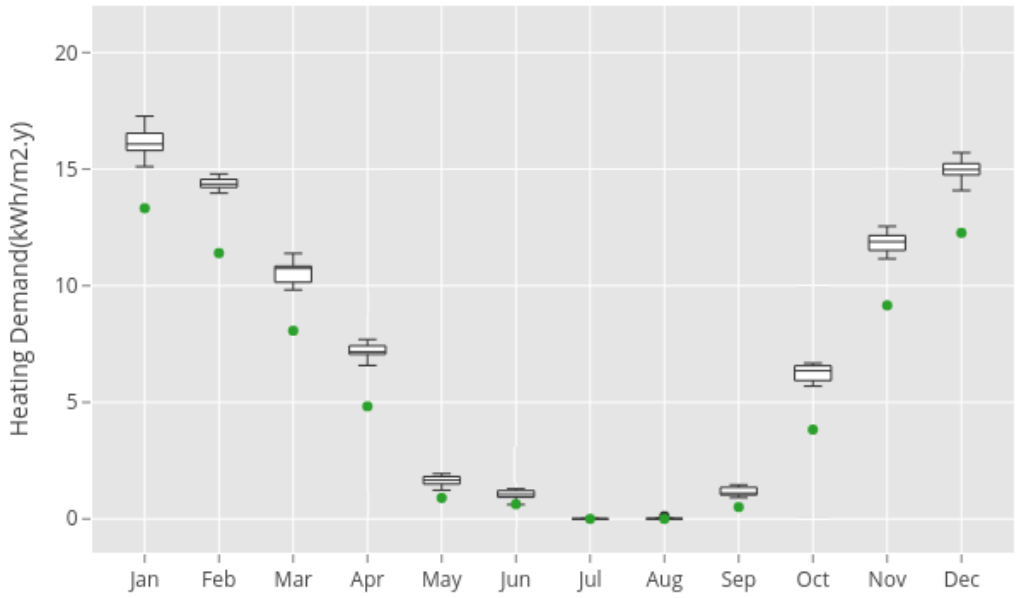



**Figure 5.22: Monthly heating demand of the shoe box office 1 in Finningley (COPE). (Boxplot) Stochastic agent platform 20 replicates, (Green point) Single deterministic simulation**

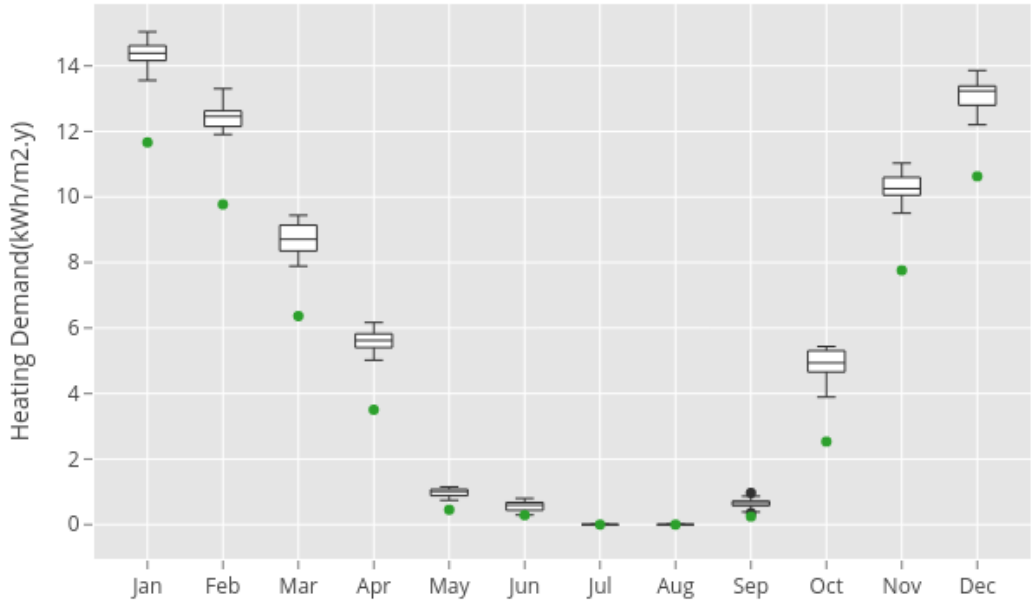



**Figure 5.23: Monthly heating demand of the shoe box office 2 in Finningley (COPE). (Boxplot) Stochastic agent platform 20 replicates, (Green point) Single deterministic simulation**

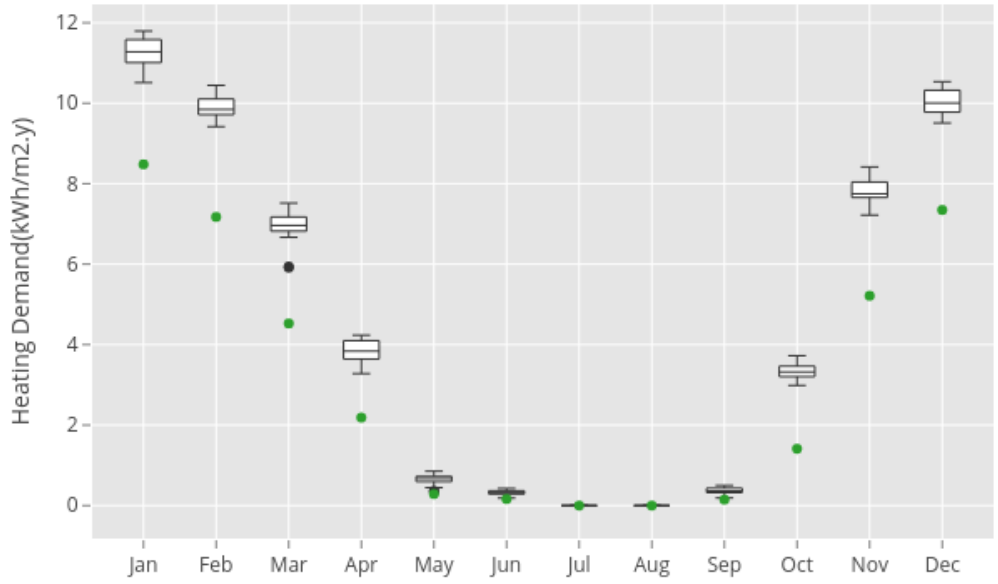



**Figure 5.24: Monthly heating demand of the shoe box office 3 in Finningley (COPE). (Boxplot) Stochastic agent platform 20 replicates, (Green point) Single deterministic simulation**

These monthly box plots illustrate the possible range of energy demands arising from occupants' behaviour and existence in these three shoe box buildings over one year

period. The range of different seed numbers enable us to simulate likely range of possible energy demands arising from occupants' behaviour and existence in these three shoe box buildings, such as window interactions, lighting switching behaviour and heat gains due to the occupants' presence.

During winter months, the heating demand is higher for the stochastic simulations, as occupants can interact with the windows at a range of temperatures (e.g. to refresh the indoor air). For example, the deterministic heating demand in December is 12.27 kWh/m$^2$, 10.63 kWh/m$^2$, and 7.37 kWh/m$^2$ for shoe box office 1, shoe box office 2 and shoe box office 3 respectively. Correspondingly, the heating demand of shoe box office 1 considering stochastic behaviours is in the range of 14.76 kWh/m$^2$ and 19.56 kWh/m$^2$, and the median value is 14.98 kWh/m$^2$. The heating demand of shoe box office 2 is in the range of 12.2 kWh/m$^2$ and 13.86 kWh/m$^2$ considering stochastic behaviours, and the median value is 12.23 kWh/m$^2$. Furthermore, for shoe box office 3, the heating demand is in the range of 9.51 kWh/m$^2$ and 10.54 kWh/m$^2$, and the median value is 10 kWh/m$^2$.

During summer time, such as July and August, basically there are no heating both for the deterministic and stochastic simulations. For June, there is only a short range of span. The deterministic heating demand is 12.27 kWh/m$^2$, 10.63 kWh/m$^2$, and 7.37 kWh/m$^2$ for shoe box office 1, shoe box office 2 and shoe box office 3 respectively. In comparison, the heating demand considering stochastic behaviours for shoe box office 1 is in the range of 1.23 kWh/m$^2$ and 3.51 kWh/m$^2$, and the median value is 0.9 kWh/m$^2$. For shoe box office 2, the heating demand is in a the range of 0.29 kWh/m$^2$ and 0.8 kWh/m$^2$ considering stochastic behaviours, and the median value is 0.6 kWh/m$^2$. Moreover, for shoe box office 3 the heating demand is in the range of 0.19 kWh/m$^2$ and 0.43 kWh/m$^2$, and the median value is 0.33 kWh/m$^2$.

The same Debian virtual machine with the same configuration was used to run case 2. Regarding co-simulation time, it took about 74 minutes for COPE to run co-simulation of the three EnergyPlus shoe box buildings models and the corresponding No-MASS models for 20 replicates on an annual basis. In comparison, it's about 22 minutes to run one shoe box co-simulation for 20 replicates on an annual basis. The co-simulation duration is more than three times greater than the single building co-simulation. This means the mosaik master algorithm is not designed to support running co-simulation

in parallel using multiple threads on multiple processors. This makes the co-simulation a relatively inexpensive undertaking.

If mosaik provides multithreading functionality, the simulation time for multiple FMUs co-simulation is expected to be much faster. That will enable more complex urban scale co-simulation scenarios for an affordable time duration.

## 5.3 Outcome

The development of use cases follows a verification and validation process. First, Case 1 focuses on the verification of the platform through a single building energy co-simulation. Through the result analysis of the case study by comparing results between COPE and the direct coupling approach, the fundamental integration and orchestration functionality of COPE were verified.

The applicability of COPE is demonstrated in a more complex scenario in Case 2. This use case focuses on the simulation of energy flows in urban energy systems in a neighbourhood context by coupling and orchestrating the co-simulation of three EnergyPlus shoebox models and corresponding No-MASS models. By illustrating a neighbourhood multiple-buildings co-simulation, the applicability of COPE was demonstrated. Thus, Obj. IV (Evaluate the approach and the platform through use cases to demonstrate synchronisation and interaction between the urban energy co-simulation platform and coupled co-simulation components) of the research is addressed.

Through the result analysis of the demonstrated cases, it shows the integrated energy simulation systems based on the platform are capable of capturing the dynamic interaction between various aspects of urban energy systems and providing an integrated representation of urban energy use. The modular layered co-simulation platform is able to integrate and manage the interactions of diverse energy subsystem simulation tools with different granularities to comprehensively simulate multiple domains of urban energy systems. Simulation tools which support the FMI co-simulation standard can be easily attached to COPE to comprehensively model the complexity of urban energy systems.

By utilising well-known domain specific urban energy simulation tools, the approach will make the interoperability and reuse of existing implementations possible and it

can also be easily used by other researchers to integrate simulation tools for their own purposes. As a result, a flexible and sustainable approach to meeting various simulation requirements can be achieved.

The current use case focuses on neighbourhood simulation, but the approach can be applied to the city level using suitable simulation tools. In theory, the platform is capable of coupling and orchestrating simulation models as many as possible, from single building to urban scale level. The process to conduct COPE based co-simulation is illustrated in Figure 5.25.

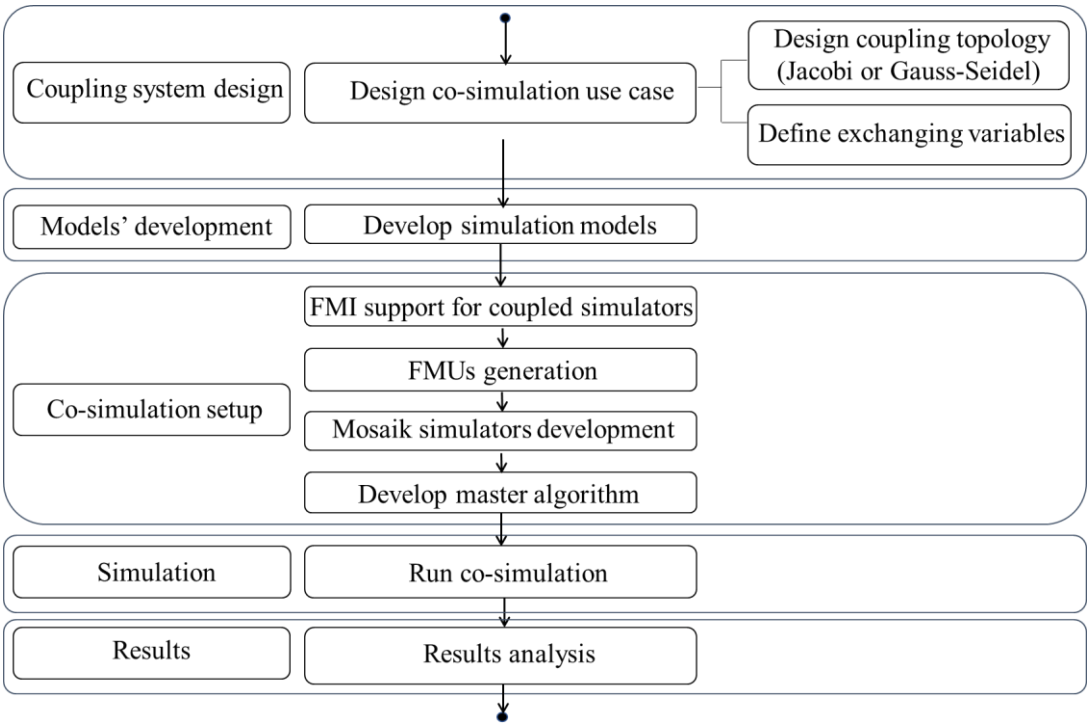



**Figure 5.25: Process to conduct co-simulation based on the urban energy co-simulation platform**

# 6 CONCLUSION

The principal aim of the research presented in this thesis was to develop a generic integration approach to tackle the simulation interoperability challenge of a comprehensive simulation for urban energy systems. This was achieved through the development of a generic and scalable urban energy co-simulation architecture for the integration of urban energy simulation tools presented in Chapter 3. The co-simulation platform, COPE, to validate the conceptual architecture was presented in Chapter 4, and then the evaluation was presented in Chapter 5 through two case studies following the verification and validation process.

The work presented in this thesis is discussed in this chapter in terms of the strengths and weaknesses of the conceptual co-simulation architecture as well as its implementation platform COPE (Section 6.1) and the research methods (Section 6.2). Section 6.3 outlines a selection of future work based upon the weaknesses and potential research directions.

## 6.1 Generic and scalable urban energy co-simulation approach

In this section, the main results of the work are discussed regarding the strengths and weaknesses of the adopted generic and scalable urban energy co-simulation approach in this thesis.

Overall, the work presented in this thesis addressed the research gap identified in Chapter 1 and the requirements raised in Chapter 2. Specifically, the major contribution of the research presented in this thesis is the generic and scalable urban

energy co-simulation approach providing a solution for urban energy simulation tools integration, which was achieved by identifying research requirements for the generic and scalable urban energy co-simulation, designing a conceptual co-simulation architecture, developing COPE and verifying the functionality and applicability of COPE.

### 6.1.1 Conceptual co-simulation architecture

The energy characteristics of cities are complex as they involve interactions of multiple domains, each has complex energy characteristics among different urban energy subsystems. In order to facilitate accurate simulation of urban energy systems, a modularised conceptual co-simulation architecture was proposed in the research presented in this thesis.

The conceptual urban co-simulation architecture was designed in a layered modular architecture. Each layer of the architecture consists of a modular set of components with its own functions and provides disciplined interfaces for other modules to access its functionalities. The design ensured that the platform developed based on this architecture could be extended to include alternative standards, middleware and integration tools.

Specifically, the conceptual co-simulation architecture is consisted of four layers, i.e. simulator layer, interconnection layer, interoperability layer and control layer. Individual urban energy simulation tools could be integrated through the simulator layer and the overall simulation purpose could be implemented through the control and interoperability layers. There are no direct control and data exchange required between the integrated simulation tools. The integrated simulators are orchestrated by a master algorithm in the control layer.

To achieve an accurate and comprehensive simulation of urban energy systems, it is necessary to integrate the most suitable existing simulation tools, which are capable of simulating different sub energy systems such as buildings, stochastic occupants' behaviour, solar radiation, distribution networks and energy storage systems, etc. The modular design of the architecture ensures that suitable software libraries and simulators could be selected and integrated in order to obtain the required simulation task of specific energy systems.

Strengths

- The layered modular design of the architecture makes it extendable and flexible to add more functionalities and easy to couple simulators together for a specific scenario.

Weaknesses

- Programming expertise and profound knowledge of the FMI standard are required to develop a co-simulation platform based on this architecture. This brings complexity and could represent a barrier to the real-world implementation.
- In order to be part of this architecture, simulators need to be FMI-compliant with FMU export functionality. For some simulators without FMI support, it's required to develop FMI co-simulation API functions to be capable of encapsulating in FMUs for co-simulation. This requires a deep understanding of the simulators and the FMI standard.

### 6.1.2 COPE

Based on the conceptual co-simulation architecture, COPE was developed to tackle simulation interoperability challenge of urban energy systems. The platform addressed such a challenge through the layered modular design using FMI as the co-simulation standard. In addition, it uses middleware mosaik to orchestrate the simulation and data flow of individual simulators programmed in different programming languages by calling FMI interfaces provided by the FMI adapter through the PyFMI and FMI Library. They built the basis for the interconnection layer and interoperability layer of the platform.

In comparison to the direct coupling approach, the COPE approach is able to tackle co-simulation involving more than two simulators. There is no limitation of the number of simulators for COPE to couple and orchestrate. The second case presented in Chapter 5 demonstrated six simulators, e.g. three EnergyPlus shoe box office simulators and three No-MASS occupants behaviour simulators, coupled and orchestrated by COPE successfully. However, because communications among simulators are conducted through the COPE platform in the COPE approach, it took longer to run a simulation compared to the direct coupling approach. Therefore, for a

simple simulation scenario, if the direct coupling approach is achievable, it will be more efficient to simulate through direct coupling.

For specific urban energy simulation purposes, a variety of urban energy subsystem simulation tools could be integrated into the platform in order to obtain required comprehensive simulation results.

The modularity and extendibility of COPE make it capable of coupling and orchestrating existing implementations. Simulation tools, which support the de facto co-simulation standard FMI, can be easily integrated. This enables COPE to easily couple well-known domain specific urban energy simulation tools, which support FMI-standard, with minimum effort and cost. Correspondingly, it can also be readily used by other researchers to couple specialised simulation tools in different domain sectors, e.g. buildings, occupant behaviours, thermal and electrical energy network, etc., for their specific purposes. This makes the interoperability and reuse of existing implementations possible and it can also be used by future research projects to integrate simulators to fit project purposes.

In this research, co-simulation standard FMI 2.0 was used in the current implementation. In recent years, the FMI development team revised it by adding more sophisticated co-simulation features. As a major milestone, the FMI 3.0 was released in May 2022, with new features that enable the use of FMI in important new use cases, such as the next generation of Digital Twins, artificial intelligence, and autonomous driving applications. In the future, it is necessary to upgrade COPE to be compliant with FMI 3.0 standard. Therefore, more advanced co-simulation can be achieved.

Strengths

- Suitable co-simulation software libraries were adopted and mapped together to fulfil requirements of each layer of COPE to achieve the research objectives.
- Able to integrate urban energy simulators from different domains.
- Can integrate simulators programmed in different programming languages.
- Able to capture the dynamic interaction between various aspects of urban energy systems and provide an integrated representation of urban energy use.
- Easy to be maintained, customised or extended for different applications and research purposes.

Weaknesses

- It took longer to run a simulation in comparison to the direct coupling approach.
- The COPE platform is required to be set up in order to run a co-simulation scenario.
- It requires end-users to have a programming background in order to generate FMUs and develop a master algorithm for a co-simulation scenario.

## 6.2 Research methods

The objectives of the research were identified as:

Obj. I: Identify requirements for the generic and scalable urban energy co-simulation.

Obj. II: Design a conceptual co-simulation architecture that will be able to integrate urban energy simulation tools from different domains. The approach will address the identified requirements.

Obj. III: Develop an urban energy co-simulation platform based on the conceptual architecture. Explicit process to integrate well known domain specific urban energy simulation tools with different levels of granularities will also be presented.

Obj. IV: Evaluate the approach and the platform through use cases to demonstrate synchronisation and interaction between the urban energy co-simulation platform and coupled co-simulation components.

Through a **literature review** of simulators, co-simulation standards, co-simulation middleware, and co-simulation architecture, nine requirements of generic urban energy systems co-simulation were identified. The nine requirements as well as their description are listed in Table 2.2. (Obj. I was achieved)

To address the nine requirements, the conceptual urban co-simulation architecture was **designed**. (Obj. II was achieved)

Based on the conceptual architecture, the Co-simulation Platform for Ecological-urban (COPE) was **developed** for urban energy systems simulation. An application process using COPE for a co-simulation scenario was presented in Figure 4.11. (Obj. III was achieved)

In order to **validate** the conceptual co-simulation architecture, as well as **evaluate** the fundamental functionalities of COPE and demonstrate the platform application, two **case studies** (single building and multiple buildings simulations) were designed, set up and simulated. The integration, synchronisation and interaction between the co-simulation platform and coupled simulation tools were demonstrated. (Obj. IV was achieved)

The two case studies simulated two different scenarios: one was a single shoe box office building co-simulation, and the other was multiple buildings co-simulation. The first case showed COPE achieved equivalent simulation results in comparison with the results obtained from Chapman's direct coupling approach. The master algorithm and coupled simulators integrated into COPE synchronised and interacted as intended. Therefore, the fundamental integration and orchestration functionality of COPE were verified. The second case demonstrated the simulation of energy flows in urban energy systems in a neighbourhood context by coupling and orchestrating the co-simulation of three EnergyPlus shoebox models and corresponding No-MASS models. This demonstrated the applicability of COPE for a more complex energy system co-simulation.

In both case studies, EnergyPlus and No-MASS models were integrated to simulate shoe box office energy behaviour and the effect of occupants' stochastic behaviours on the building energy performance.

The two use cases focus on simulating energy flows in a neighbourhood context. However, the approach can be applied to the city level using suitable simulators because the platform is capable of coupling and orchestrating multiple simulation models, which are from neighbourhood to urban scale level. As a result, a flexible and sustainable approach to address various simulation requirements can be achieved.

Strengths

- A systematic approach was adopted which includes a number of methods from literature review to evaluation.
- The main research contribution was validated through the platform COPE.
- The COPE approach was evaluated through two use cases.

Weaknesses

- Only a neighbourhood energy system was simulated in the use cases, though more complex multiple domains could be included.
- Though multiple simulators could be integrated into COPE (two in Case 1 and six in Case 2), an extensive test of the number of simulators was not conducted, i.e. the number of simulators that could be integrated with a standard server configuration without performance compromise.

The approach helps to further explore the subject of urban energy domain to achieve energy efficiency improvement in new and existing urban areas. To take the most advantage of this approach in practice, urban designers and planners need to meet various domain experts to design appropriate plans by utilising COPE and therefore propose confident a suitable, high-quality and sustainable solution.

Overall, the research aim and objectives listed in Section 1.2 were achieved through the research methods adopted in the research. The COPE approach is well suited for decision making in urban energy planning and optimisation processes to improve energy efficiency and urban sustainability through comprehensively modelling the dynamic, complex and interactive nature of urban energy systems. This is viewed to be of great importance with the current global climate crisis and government target for emission control.

## 6.3 Outlook and future work

In recent years, the energy transition in the UK has been accelerated with increasing decarbonisation, decentralisation and digitalisation of local energy systems which bring challenges, but also opportunities (Wang and Wade 2021). Modelling and simulation as a state-of-the-art method are expected to play a significant role during this transition.

The co-simulation approach tackles the challenge of comprehensive modelling of urban energy systems by coupling existing domain-specific energy simulation tools, each with its own speciality. The modularity and extendable capability of the COPE approach facilitate the demonstration of more urban representative cases by coupling specialised simulators in different domains, sectors and scales as illustrated in Figure 6.1.

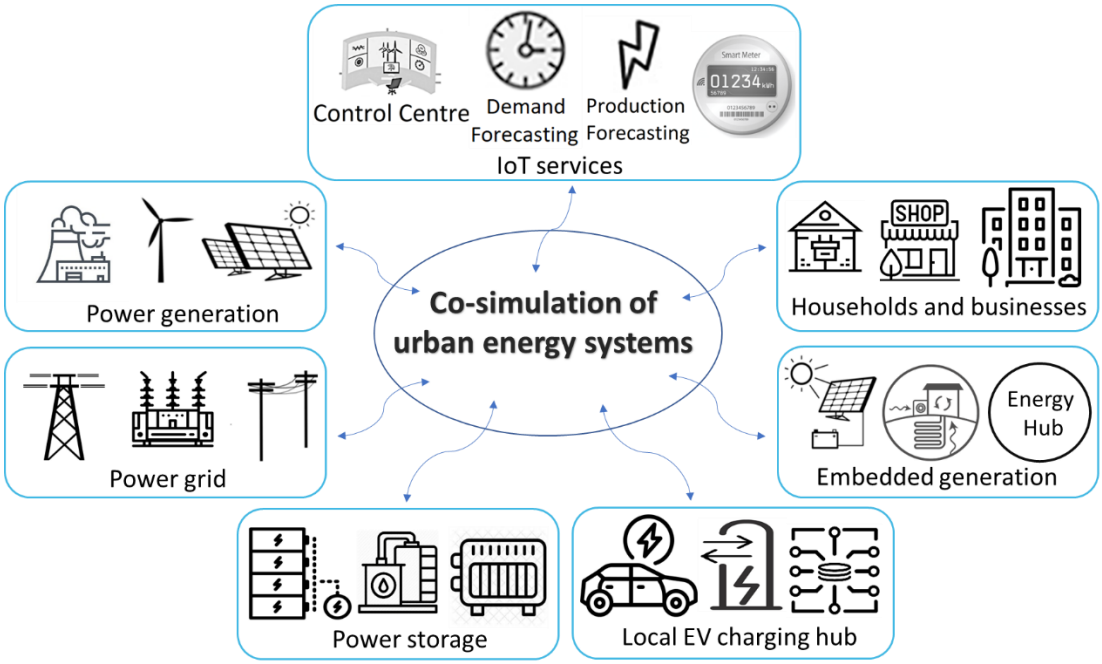



**Figure 6.1: Co-simulation of urban energy systems**

The COPE approach is capable of rapid prototyping and enables the implementation of complex multi-domain system simulations. This helps to simplify cross-domain collaborations significantly by allowing collaborators to utilise preferred simulators, and eventually a holistic collaborative simulation by coupling and orchestrating them can be achieved.

The implementation of COPE can be facilitated in parallel and distributed computing by accelerating time-consuming simulations on distributed supercomputer systems. By running the simulation in real-time experiments, such as in the context of hardware-in-the-loop, COPE enables the possibility of application opportunities not only in urban energy systems but also in other fields of industry and research. So optimised solutions of energy generation, distribution, store and utilisation can be achieved.

There are a few desirable improvements to COPE identified during the evaluation of the approach, which are described as follows.

- FMI 3.0 support

There are more features introduced by the current FMI 3.0 release, such as scheduled execution and clock-based simulation. The scheduled execution is for purely discrete, RTOS-like, simulation and supports pre-emption (Junghanns, Gomes et al. 2021).

Furthermore, clocks are introduced to allow precise coordination of global events between FMUs and the importer (Junghanns, Gomes et al. 2021). More data type support comes with FMI 3.0, including binary data and arrays (Junghanns, Gomes et al. 2021). In order to take these advantages, the FMI libraries adopted by COPE need to be upgraded to the latest version and the FMUs to be imported need to be FMI 3.0 compliant.

- HIL (Hardware-In-The-Loop) simulation

Co-simulation can be used in parallel to hardware-in-the-loop simulations with actual controller components in real-time simulations using the techniques and tools (Taveres-Cachat, Favoino et al. 2021). It's expected that COPE is capable of running real-time and hardware-in-the-loop simulation, though this needs to be further evaluated. To gain appropriate performance, proper co-simulation time-step and necessary optimisation of simulation tools need to be evaluated in order to reduce the processing time for a realistic proposition. During the optimisation, more impacting factors might be identified through detailed investigation.

- Optimised control of energy usage for minimal emission and minimal cost

Nowadays, more and more consumers using energy storage systems with a solar PV system. This offers flexibility to consumers in choosing energy source at right time. By controlling a smart system with an embedded program with minimal customer cost and minimal emission options, greenhouse gas emission reduction or cost saving on electricity bills can be achieved. Therefore, it will be helpful to implement optimal scheduling solvers in the master algorithm of COPE.

- Unified data format for result store and data analysis workflow

Simulation numerical results normally are encoded in a variety of structured formats (e.g. JSON, Hive tables, ORC, CSV, HDF5, etc.). Depending on coupled simulation tools, different datasets could be fetched. This introduces overhead for post simulation data processing, especially for large scale urban energy simulation. Therefore, if a unified computationally efficient data result format scheme is adopted, this will help efficiency of organising simulation results and facilitating data analysis. The unified results data could be utilised by data visualisation libraries, like Plotly, Matplotlib etc., to produce interactive plots and perform basic statistical analyses of data.

- Multiple domain implementation

As the case studies presented in this thesis were both about neighbourhood energy systems, it will be desirable to see COPE demonstrate more complex cases by integrating and managing the interactions of diverse energy subsystem simulation tools simulators with different granularities to comprehensively simulate multiple domains of urban energy systems. This can be achieved by coupling more well-known domain specific urban energy simulators, like Dymola, JModelica, OpenModelica, TRNSYS, MATLAB/Simulink, PowerWorld, and PowerFactory etc., in the future work.

- Parallel co-simulation on multi-core processors

In order to deal with large scale urban energy simulation involving a large amount of simulators and much more data interactions, COPE could be enhanced to provide capability to run co-simulations fully parallel on a High Performance Computing (HPC) cluster. By doing so, each subsystem simulator will be executed on individual core and orchestrated by master algorithm over the InfiniBand interconnect. As a result, the simulation speed will be improved.

- Further adoption in simulation and digital analysis domains

The co-simulation approach presented in this thesis provides a generic and scalable approach to simulate urban energy systems. The layered modular design of the conceptual architecture makes it easy to be extended and maintained. The approach can be adopted in other research domains. For example, it could be adapted to the real-time data analysis field. The local energy systems in UK nowadays is in a transition towards increased decentralisation and digitalisation (Wang and Wade 2021). In order to make the best use of increasing distributed energy generation and storage, maximise customer engagement and schedule demand-side resources, accurate simulation and enhanced data analysis are required for providing optimised solution evaluation in this trend.

Inspired by the methodology adopted by COPE, a modularised multi-layer design of smart digital energy platform can be formalised providing data analysis function to be used to evaluate sustainable solutions and propose green, affordable, resilient and optimal solutions by conducting experimental data analysis as proposed in (Wang and Wade 2021). Similarly, layered architecture provides functionalities of data fetching,

processing and comprehensively analysing local energy systems' data to illustrate insights and outcomes.

With more functionalities to be implemented in a specific layer of COPE and the design and implementation of a smart digital energy platform based on the conceptual architecture proposed in this thesis, a simulation and data analysis ecosystem can be established and be used to explore and investigate hypothetical and real-time operating scenarios for future smart local energy systems. This provides new ways of local energy system monitoring, optimal control and fault detection, which ultimately ensures demands are satisfied and energy sources are optimised. All this future work needs to be evaluated through a thorough functionality and performance verification and evaluation process. In this procedure, the incorporation of domain expertise is needed in order to fully understand and interpret solutions for different scenarios. By using such expert knowledge, a better and deeper understanding of the results can be achieved.

# 7 REFERENCES

(DOE), D. o. E. (2018). "Weather Data." Retrieved Oct 20, 2018, 2018, from https://energyplus.net/weather.

Abbasabadi, N. and M. Ashayeri (2019). "Urban energy use modeling methods and tools: A review and an outlook." Building and Environment **161**: 106270.

Agugiaro, G. (2016). "Energy planning tools and CityGML-based 3D virtual city models: experiences from Trento (Italy)." Applied Geomatics **8**(1): 41-56.

Agugiaro, G., J. Benner, P. Cipriano and R. Nouvel (2018). "The Energy Application Domain Extension for CityGML: enhancing interoperability for urban energy simulations." Open Geospatial Data, Software and Standards **3**(1): 1-30.

Agugiaro, G., F. Nex, F. De Remondino, R. Filippi, S. Droghetti and C. Furlanello (2012). "Solar radiation estimation on building roofs and web-based solar cadastre." ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences **1**(2): 177-182.

Ali, U., M. H. Shamsi, C. Hoare, E. Mangina and J. O'Donnell (2021). "Review of urban building energy modeling (UBEM) approaches, methods and tools using qualitative and quantitative analysis." Energy and Buildings **246**: 111073.

Allegrini, J., K. Orehounig, G. Mavromatidis, F. Ruesch, V. Dorer and R. Evins (2015). "A review of modelling approaches and tools for the simulation of district-scale energy systems." Renewable and Sustainable Energy Reviews **52**: 1391-1404.

Andersson, C. (2013). A Software Framework for Implementation and Evaluation of Co-Simulation Algorithms. 2013.

Andersson, C., J. Åkesson and C. Führer (2016). "PyFMI: A Python Package for Simulation of Coupled Dynamic Models with the Functional Mock-up Interface." Technical Report in Mathematical Sciences **2016**(2).

Aprilia, E., K. Meng, M. A. Hosani, H. H. Zeineldin and Z. Y. Dong (2019). "Unified Power Flow Algorithm for Standalone AC/DC Hybrid Microgrids." IEEE Transactions on Smart Grid **10**(1): 639-649.

Arnold, M. (2004). Simulation algorithms in vehicle system dynamics, Univ., Fachbereich Mathematik und Informatik.

Arnold, M. (2010). "Stability of Sequential Modular Time Integration Methods for Coupled Multibody System Models." Journal of Computational and Nonlinear Dynamics **5**(3): 031003-031003-031009.

Association, M. (2022). "FMI standard." Retrieved June 29, 2022, 2022, from https://fmi-standard.org/.

Azar, E., W. O'Brien, S. Carlucci, T. Hong, A. Sonta, J. Kim, M. S. Andargie, T. Abuimara, M. El Asmar, R. K. Jain, M. M. Ouf, F. Tahmasebi and J. Zhou (2020). "Simulation-aided occupant-centric building design: A critical review of tools, methods, and applications." Energy and Buildings **224**: 110292.

Bahu, J., A. Koch, E. Kremers and S. Murshed (2013). Towards a 3D spatial urban energy modelling approach. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Proceedings of the ISPRS 8th 3D GeoInfo Conference & WG II/2 Workshop.

Bass, L., P. Clements and R. Kazman (2003). Software architecture in practice, Addison-Wesley Professional.

Bastian, J., C. Clauß, S. Wolf and P. Schneider (2011). Master for co-simulation using FMI. 8th International Modelica Conference, Dresden, Citeseer.

Blochwitz, T., M. Otter, J. Åkesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss and D. Neumerkel (2012). Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. 9th International Modelica Conference.

Blochwitz, T., M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro and T. Neidhold (2011). The functional mockup interface for tool independent exchange of simulation models. 8th International Modelica Conference, Dresden.

Boehm, B. (1995). Engineering Context (for Software Architecture). Invited talk, First International Workshop on Architecture for Software Systems. Seattle, Washington.

Brand, L., A. Calvén, J. Englund, H. Landersjö and P. Lauenburg (2014). "Smart district heating networks – A simulation study of prosumers' impact on technical parameters in distribution networks." Applied Energy **129**: 39-48.

Brange, L., J. Englund and P. Lauenburg (2016). "Prosumers in district heating networks – A Swedish case study." Applied Energy **164**: 492-500.

Brooks, C., E. Lee, M. Wetter, T. Nouidui, D. Broman and S. Tripakis. (2012). "JFMI-A Java Wrapper for the Functional Mock-up Interface." from http://ptolemy.eecs.berkeley.edu/java/jfmi/.

Brooks, C., E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, H. Zheng, S. S. Bhattacharyya, E. Cheong, I. Davis and M. Goel (2008). Heterogeneous concurrent modeling and design in java (volume 1: Introduction to ptolemy ii), CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE.

Brown, S. (2014). Software Architecture for Developers, Barnes & Noble.

Buckley, N., G. Mills, C. Reinhart and Z. M. Berzolla (2021). "Using urban building energy modelling (UBEM) to support the new European Union's Green Deal: Case study of Dublin Ireland." Energy and Buildings **247**: 111115.

Busch, M. and B. Schweizer (2010). Numerical stability and accuracy of different co-simulation techniques: analytical investigations based on a 2-DOF test model. Proceedings of The 1st Joint International Conference on Multibody System Dynamics, IMSD.

Cerezo Davila, C., C. F. Reinhart and J. L. Bemis (2016). "Modeling Boston: A workflow for the efficient generation and maintenance of urban building energy models from existing geospatial datasets." Energy **117**: 237-250.

Chapman, J., P.-O. Siebers and D. Robinson (2018). "On the multi-agent stochastic simulation of occupants in buildings." Journal of Building Performance Simulation **11**(5): 604-621.

CityGML. (2015). "CityGML Basic Information." Retrieved 18 June 2015, 2015, from http://www.citygmlwiki.org/index.php/Basic_Information.

Clarke, J. (2013). "Moisture flow modelling within the ESP-r integrated building performance simulation system." Journal of Building Performance Simulation **6**(5): 385-399.

Clements, P. C. and L. M. Northrop (1996). Software Architecture: An Executive Overview, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

Connolly, D., H. Lund, B. V. Mathiesen and M. Leahy (2010). "A review of computer tools for analysing the integration of renewable energy into various energy systems." Applied Energy **87**(4): 1059-1082.

Crawley, D. B., L. K. Lawrie, C. O. Pedersen, F. C. Winkelmann, M. J. Witte, R. K. Strand, R. J. Liesen, W. F. Buhl, Y. J. Huang and R. H. Henninger (2004). "EnergyPlus: new, capable, and linked." Journal of Architectural and Planning Research: 292-302.

Crawley, D. B., L. K. Lawrie, F. C. Winkelmann, W. F. Buhl, Y. J. Huang, C. O. Pedersen, R. K. Strand, R. J. Liesen, D. E. Fisher, M. J. Witte and J. Glazer (2001). "EnergyPlus: creating a new-generation building energy simulation program." Energy and Buildings **33**(4): 319-331.

Dahmann, J., M. Salisbury, P. Barry and P. Blemberg (1999). HLA and beyond: Interoperability challenges. Simulation Interoperability Workshop.

Dahmann, J. S., R. M. Fujimoto and R. M. Weatherly (1997). The Department of Defense High Level Architecture. Proceedings of the 29th conference on Winter simulation. Atlanta, Georgia, USA, IEEE Computer Society**:** 142-149.

Dalla Rosa, A., R. Boulter, K. Church and S. Svendsen (2012). "District heating (DH) network design and operation toward a system-wide methodology for optimizing renewable energy solutions (SMORES) in Canada: A case study." Energy **45**(1): 960-974.

De Mello, B. A. and F. R. Wagner (2002). A standardized co-simulation backbone. SoC Design Methodologies, Springer**:** 181-192.

De Sturler, E., J. Hoeflinger, L. Kale and M. Bhandarkar (2001). A new approach to software integration frameworks for multi-physics simulation codes. The Architecture of Scientific Software, Springer**:** 87-104.

Deng, Z., Y. Chen, J. Yang and Z. Chen (2022). "Archetype identification and urban building energy modeling for city-scale buildings based on GIS datasets." Building Simulation **15**(9): 1547-1559.

Diba, K., K. Batoulis, M. Weidlich and M. Weske (2020). "Extraction, correlation, and abstraction of event data for process mining." WIREs Data Mining and Knowledge Discovery **10**(3): e1346.

DIgSILENT. (2019). "PowerFactory." Retrieved Feb 17, 2019, 2019, from http://www.digsilent.de.

Dijkstra, E. W. (1968). The structure of the “THE” multiprogramming system. The origin of concurrent programming, Springer**:** 139-152.

Dols, W. S., S. J. Emmerich and B. J. Polidoro (2016). Coupling the multizone airflow and contaminant transport software CONTAM with EnergyPlus using co-simulation. Building simulation, Tsinghua University Press.

Evora, J., J. Hernandez and O. Roncal (2014). "JavaFmi." URL https://bitbucket. org/siani/javafmi.

Fairbanks, G. (2010). Just enough software architecture: a risk-driven approach, Marshall & Brainerd.

Falcone, A. and A. Garro (2019). "Distributed Co-Simulation of Complex Engineered Systems by Combining the High Level Architecture and Functional Mock-up Interface." Simulation Modelling Practice and Theory **97**: 101967.

Fowler, J. W. and O. Rose (2004). "Grand challenges in modeling and simulation of complex manufacturing systems." Simulation **80**(9): 469-476.

Fujimoto, R. and P. Hoare (1998). HLA RTI performance in high speed LAN environments. in Proceedings of the Fall Simulation Interoperability Workshop, Citeseer.

Garlan, D. and D. E. Perry (1995). "Introduction to the special issue on software architecture." IEEE Trans. Software Eng. **21**(4): 269-274.

Garlan, D. and M. Shaw (1993). An introduction to software architecture. Advances in software engineering and knowledge engineering, World Scientific**:** 1-39.

Gea (2012). Global Energy Assessment - Toward a Sustainable Future. Cambridge University Press, Cambridge, UK and New York, NY, USA and the International Institute for Applied Systems Analysis, Laxenburg, Austria.

Gervásio, H., P. Santos, L. S. da Silva and A. Lopes (2010). "Influence of thermal insulation on the energy balance for cold-formed buildings." Advanced Steel Construction **6**(2): 742-766.

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

Gomes, C., C. Thule, D. Broman, P. G. Larsen and H. Vangheluwe (2018). "Co-Simulation: A Survey." ACM Comput. Surv. **51**(3): 1-33.

Grubler, A., X. Bai, T. Buettner, S. Dhakal, D. J. Fisk, T. Ichinose, J. E. Keirstead, G. Sammer, D. Satterthwaite, N. B. Schulz, N. Shah, J. Steinberger and H. Weisz (2012). Chapter 18 - Urban Energy Systems. Global Energy Assessment - Toward a Sustainable Future. Cambridge University Press, Cambridge, UK and New York, NY, USA and the International Institute for Applied Systems Analysis, Laxenburg, Austria**:** 1307-1400.

Gurecky, W., D. De Wet, M. S. Greenwood, R. Salko Jr and D. Pointer (2020). Coupling of CTF and TRANSFORM using the Functional Mockup Interface, Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States).

Hamilton, A., H. Wang, A. M. Tanyer, Y. Arayici, X. Zhang and Y. Song (2005). "Urban information model for city planning." Journal of Information Technology in Construction (ITCon) **10**(6): 55-67.

Hedegaard, R. E., M. H. Kristensen, T. H. Pedersen, A. Brun and S. Petersen (2019). "Bottom-up modelling methodology for urban-scale analysis of residential space heating demand response." Applied Energy **242**: 181-204.

Heinzl, B., W. Kastner, Du, x, F. r, F. Bleicher, I. Leobner and I. Kovacic (2014). Using coupled simulation for planning of energy efficient production facilities. Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2014 Workshop on.

Hensen, J. (1995). Modelling coupled heat and airflow: ping pong vs. onions. DOCUMENT-AIR INFILTRATION CENTRE AIC PROC, Citeseer.

Hong, T., H. Sun, Y. Chen, S. C. Taylor-Lange and D. Yan (2016). "An occupant behavior modeling tool for co-simulation." Energy and Buildings **117**: 272-281.

Hopkinson, K., W. Xiaoru, R. Giovanini, J. Thorp, K. Birman and D. Coury (2006). "EPOCHS: a platform for agent-based electric power and communication simulation built from commercial off-the-shelf components." IEEE Transactions on Power Systems **21**(2): 548-558.

Howell, S., Y. Rezgui, J.-L. Hippolyte, B. Jayan and H. Li (2017). "Towards the next generation of smart grids: Semantic and holonic multi-agent management of distributed energy resources." Renewable and Sustainable Energy Reviews **77**: 193-214.

IEA (2021). Empowering Cities for a Net Zero Future: Unlocking Resilient, Smart, Sustainable Urban Energy Systems, OECD Publishing.

IEEE (2010). "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-- Framework and Rules." IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000): 1-38.

Junghanns, A., C. Gomes, C. Schulze, K. Schuch, R. Pierre, M. Blaesken, I. Zacharias, A. Pillekeit, K. Wernersson and T. Sommer (2021). The Functional Mock-up Interface 3.0-New Features Enabling New Applications. Modelica Conferences.

Kampf, J. H. and D. Robinson (2007). "A simplified thermal model to support analysis of urban resource flows." Energy and Buildings **39**(4): 445-453.

Kang, M. (2010). "HLA/RTI and Relative Key Implementation Technologies." Modern Applied Science **4**(5): 143.

Kavgic, M. (2013). A city scale physically disaggregated bottom-up energy model: technical options for decarbonising Belgrade residential stock, UCL (University College London).

Keirstead, J., M. Jennings and A. Sivakumar (2012). "A review of urban energy system models: Approaches, challenges and opportunities." Renewable & Sustainable Energy Reviews **16**(6): 3847-3866.

Keirstead, J., N. I. Samsatli and N. Shah (2010). SynCity: An Integrated Tool Kit for Urban Energy Systems Modeling, WORLD BANK INST**:** 41-61.

Keirstead, J. and K. H. Van Dam (2010). A comparison of two ontologies for agent-based modelling of energy systems. ATES 2010: 1st International Workshop on Agent Technologies for Energy Systems, Toronto, Canada, 11 May 2010. Workshop of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), 10-14 May 2010, Toronto, Canada.

Koppelaar, R., H. Kunz and T. Ravalde (2013). "Review of Current Advanced Integrated Models for City Regions." Prepared for UK Technology Strategy Board Future Cities Catapult, London, under Contract No. REQ010006 by the Institute for Integrated Economic Research and Imperial College London on behalf of the Ecological Sequestration Trust.

Kosek, A. M., O. Lünsdorf, S. Scherfke, O. Gehrke and S. Rohjans (2014). Evaluation of smart grid control strategies in co-simulation-integration of IPSYS and mosaik. in 18th Power Systems Computation Conference (PSCC2014).

Kruger, A. and T. H. Kolbe (2012). "Building Analysis for Urban Energy Planning Using Key Indicators on Virtual 3d City Models - the Energy Atlas of Berlin." Xxii Isprs Congress, Technical Commission Ii **39-B2**: 145-150.

Laaroussi, Y., M. Bahrar, M. El Mankibi, A. Draoui and A. Si-Larbi (2020). "Occupant presence and behavior: A major issue for building energy performance simulation and assessment." Sustainable Cities and Society **63**: 102420.

Ledoux, H., K. Arroyo Ohori, K. Kumar, B. Dukai, A. Labetski and S. Vitalis (2019). "CityJSON: A compact and easy-to-use encoding of the CityGML data model." Open Geospatial Data, Software and Standards **4**(1): 1-12.

Lehnhoff, S., O. Nannen, S. Rohjans, F. Schlogl, S. Dalhues, L. Robitzky, U. Hager and C. Rehtanz (2015). Exchangeability of power flow simulators in smart grid co-simulations with mosaik. Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2015 Workshop on.

Lévesque, M., C. Béchet, E. Suignard, M. Maier, A. Picault and G. Joós (2014). "From co-toward multi-simulation of smart grids based on HLA and FMI standards." arXiv preprint arXiv:1412.5571.

Li, Y., C. Wang, S. Zhu, J. Yang, S. Wei, X. Zhang and X. Shi (2020). "A comparison of various bottom-up urban energy simulation methods using a case study in Hangzhou, China." Energies **13**(18): 4781.

Long, G., M. Alalwany and D. Robinson (2015). Deliverable D2.1 Building typologies simulation report. http://www.insmartenergy.com/work-package-2/.

Luo, X., K. P. Lam, Y. Chen and T. Hong (2017). "Performance evaluation of an agent-based occupancy simulation model." Building and Environment **115**: 42-53.

Management Association, I. R. (2016). Renewable and Alternative Energy: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications, IGI Global.

Manfren, M., P. Caputo and G. Costa (2011). "Paradigm shift in urban energy systems through distributed generation: Methods and models." Applied Energy **88**(4): 1032-1048.

Mauree, D., S. Coccolo, J. Kaempf and J. L. Scartezzini (2017). "Multi-scale modelling to evaluate building energy consumption at the neighbourhood scale." PLoS One **12**(9): e0183437.

Medvidovic, N. and R. N. Taylor (2010). Software architecture: foundations, theory, and practice. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2, ACM.

Meyer, M. H. and P. H. Webb (2005). "Modular, layered architecture: the necessary foundation for effective mass customisation in software." International Journal of Mass Customisation **1**(1): 14-36.

Miller, C., D. Thomas, J. Kämpf and A. Schlueter (2017). "Urban and building multiscale co-simulation: case study implementations on two university campuses." Journal of Building Performance Simulation: 1-13.

Mirzahosseini, A. H. and T. Taheri (2012). "Environmental, technical and financial feasibility study of solar power plants by RETScreen, according to the targeting of energy subsidies in Iran." Renewable and Sustainable Energy Reviews **16**(5): 2806-2811.

MODELISAR. (2018). "fmi-standard." Retrieved Feb 17, 2017, 2017, from http://www.powerworld.com.

Modelon, A. (2014). FMI library.

Mohammadi, S., B. de Vries and W. Schaefer (2013). A Comprehensive Review of Existing Urban Energy Models in the Built Environment. Planning Support Systems for Sustainable Urban Development. S. Geertman, F. Toppen and J. Stillwell, Springer Berlin Heidelberg. **195:** 249-265.

Möller, B. (2013). "The HLA tutorial v1. 0." Pitch Technologies, Sweden.

Müller, S. C., H. Georg, J. J. Nutaro, E. Widl, Y. Deng, P. Palensky, M. U. Awais, M. Chenine, M. Küch and M. Stifter (2018). "Interfacing power system and ict simulators: Challenges, state-of-the-art, and case studies." IEEE Transactions on Smart Grid **9**(1): 14-24.

Natanian, J., O. Aleksandrowicz and T. Auer (2019). "A parametric approach to optimizing urban form, energy balance and environmental quality: The case of Mediterranean districts." Applied Energy **254**: 113637.

Neaimeh, M., R. Wardle, A. M. Jenkins, J. L. Yi, G. Hill, P. F. Lyons, Y. Hubner, P. T. Blythe and P. C. Taylor (2015). "A probabilistic approach to combining smart meter and electric vehicle charging data to investigate distribution network impacts." Applied Energy **157**(0): 688-698.

Neema, H., J. Gohl, Z. Lattmann, J. Sztipanovits, G. Karsai, S. Neema, T. Bapty, J. Batteh, H. Tummescheit and C. Sureshkumar (2014). Model-based integration

platform for FMI co-simulation and heterogeneous simulations of cyber-physical systems. 10th International Modelica Conference.

New, J. and M. Adams (2018). EnergyPlus performance improvements via JSON input refactoring. 2018 Building Performance Modeling Conference and SimBuild, Chicago, IL.

Nouidui, T. S. (2014). Tool coupling for the design and operation of building energy and control systems based on the Functional Mock-up Interface standard. 10th Modelica conference, Lund, Sweden.

Nouvel, R., K.-H. Brassel, M. Bruse, E. Duminil, V. Coors, U. Eicker and D. Robinson (2015). SimStadt, a new workflow-driven urban energy simulation platform for CityGML city models. Proceedings: Conference CISBAT.

Nouvel, R., R. Kaden, J.-M. Bahu, J. Kaempf, P. Cipriano, M. Lauster, J. Benner, E. Munoz, O. Tournaire and E. Casper (2015). Genesis of the citygml energy ADE. Proceedings of International Conference CISBAT 2015 Future Buildings and Districts Sustainability from Nano to Urban Scale, LESO-PB, EPFL.

Nouvel, R., M. Zirak, H. Dastageeri, V. Coors and U. Eicker (2014). Urban energy analysis based on 3D city model for national scale applications. IBPSA Germany conference.

OFFIS (2017). "Mosaik." URL https://mosaik.offis.de/.

Page, J., D. Basciotti, O. Pol, J. N. Fidalgo, M. Couto, R. Aron, A. Chiche and L. Fournié (2013). "A MULTI-ENERGY MODELLING, SIMULATION AND OPTIMIZATION ENVIRONMENT FOR URBAN ENERGY INFRASTRUCTURE PLANNING."

Palensky, P., A. A. V. D. Meer, C. D. Lopez, A. Joseph and K. Pan (2017). "Cosimulation of Intelligent Power Systems: Fundamentals, Software Architecture, Numerics, and Coupling." IEEE Industrial Electronics Magazine **11**(1): 34-50.

Palensky, P., E. Widl and A. Elsheikh (2014). "Simulating Cyber-Physical Energy Systems: Challenges, Tools and Methods." Ieee Transactions on Systems Man Cybernetics-Systems **44**(3): 318-326.

Pang, X., T. S. Nouidui, M. Wetter, D. Fuller, A. Liao and P. Haves (2016). "Building energy simulation in real time through an open standard interface." Energy and Buildings **117**: 282-289.

Pang, X. F., M. Wetter, P. Bhattacharya and P. Haves (2012). "A framework for simulation-based real-time whole building performance assessment." Building and Environment **54**(0): 100-108.

Pantaleo, A., j. keirstead and N. Shah (2013). Urban Energy Systems An Integrated Approach.

Parnas, D. L. (1972). "On the criteria to be used in decomposing systems into modules." Commun. ACM **15**(12): 1053-1058.

Parnas, D. L. (1976). "On the Design and Development of Program Families." IEEE Transactions on Software Engineering **SE-2**(1): 1-9.

Parnas, D. L. (1979). "Designing Software for Ease of Extension and Contraction." IEEE Transactions on Software Engineering **SE-5**(2): 128-138.

Peng, J. and K. H. Law (2010). Brief Review of Data Models for NEESgrid.

Perez, D. (2014). A framework to model and simulate the disaggregated energy flows supplying buildings in urban areas, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE.

PSCAD. (2019). "PSCAD." Retrieved Feb 17, 2019, 2019, from http://pscad.com.

Ptolemaeus, C. (2014). System Design, Modeling, and Simulation: Using Ptolemy II, Ptolemy. org.

QTronic (2014). FMU SDK.

Raad, A., V. Reinbold, B. Delinchant and F. Wurtz (2015). FMU software component orchestration strategies for co-simulation of building energy systems. Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), 2015 Third International Conference on.

Reinhart, C., T. Dogan, J. A. Jakubiec, T. Rakha and A. Sang (2013). Umi-an urban simulation environment for building energy use, daylighting and walkability. 13th Conference of International Building Performance Simulation Association, Chambery, France.

Reinhart, C. F. and C. Cerezo Davila (2016). "Urban building energy modeling – A review of a nascent field." Building and Environment **97**: 196-202.

RJ, D., W. A, H. O, K. J, D. S and F. E (2014). Understanding Cities: Advances in integrated assessment of urban sustainability Centre for Earth Systems Engineering Research (CESER), Newcastle University

Robinson, D. (2011). Computer modelling for sustainable urban design: Physical principles, methods and applications, Routledge.

Robinson, D., F. Haldi, J. Kämpf, P. Leroux, D. Perez, A. Rasheed and U. Wilke (2009). CitySim: Comprehensive micro-simulation of resource flows for sustainable urban planning. Proc. Building Simulation.

Rohjans, S., S. Lehnhoff, S. Schutte, S. Scherfke and S. Hussain (2013). mosaik - A modular platform for the evaluation of agent-based Smart Grid control. Innovative Smart Grid Technologies Europe (ISGT EUROPE), 2013 4th IEEE/PES.

Rotger-Griful, S., S. Chatzivasileiadis, R. H. Jacobsen, E. M. Stewart, J. M. Domingo and M. Wetter (2016). Hardware-in-the-Loop co-simulation of distribution Grid for demand response. 2016 Power Systems Computation Conference (PSCC).

Scherfke, S. and S. Schütte (2012). "mosaik–Architecture Whitepaper." OFFIS–Institute for Information Technology, Tech. Rep.

Schierz, T. and M. Arnold (2012). "Stabilized overlapping modular time integration of coupled differential-algebraic equations." Applied Numerical Mathematics **62**(10): 1491-1502.

Schildt, M., C. Behm, A. Malhotra, S. Weck-Ponten, J. Frisch and C. van Treeck (2021). "Proposed Integration of Utilities in the Energy ADE 2.0."

Schmidt, D. C. and F. Buschmann (2003). Patterns, frameworks, and middleware: their synergistic relationships. 25th International Conference on Software Engineering, 2003. Proceedings.

Schütte, S. (2011). Composition of simulations for the analysis of smart grid scenarios, Energieinformatik.

Schutte, S., S. Scherfke and M. Troschel (2011). Mosaik: A framework for modular simulation of active components in Smart Grids. Smart Grid Modeling and Simulation (SGMS), 2011 IEEE First International Workshop on, IEEE.

Schweiger, G., C. Gomes, G. Engel, I. Hafner, J. Schoeggl and T. Nouidui (2018). Functional Mock-up Interface: An Empirical Survey Identifies Research Challenges and Current Barriers. The American Modelica Conference, Cambridge, MA, USA.

Schweiger, G., C. Gomes, G. Engel, I. Hafner, J. Schoeggl, A. Posch and T. Nouidui (2019). "An empirical survey on co-simulation: Promising standards, challenges and research needs." Simulation Modelling Practice and Theory **95**: 148-163.

Shorrock, L. D. and J. E. Dunster (1997). "The physically-based model BREHOMES and its use in deriving scenarios for the energy use and carbon dioxide emissions of the UK housing stock." Energy Policy **25**(12): 1027-1037.

Siegfried, R. (2014). Modeling and Simulation of Complex Systems: A Framework for Efficient Agent-Based Modeling and Simulation, Springer.

Sousa, J. (2012). Energy simulation software for buildings: review and comparison. International Workshop on Information Technology for Energy Applicatons-IT4Energy, Lisabon, Citeseer.

Steinbrink, C., M. Blank-Babazadeh, A. El-Ama, S. Holly, B. Lüers, M. Nebel-Wenner, R. Ramirez, T. Raub, J. Schwarz, S. Stark, A. Nieße and S. Lehnhoff (2019). "CPES Testing with mosaik: Co-Simulation Planning, Execution and Analysis." Applied Sciences **9**: 923.

Steinbrink, C., S. Lehnhoff, S. Rohjans, T. I. Strasser, E. Widl, C. Moyo, G. Lauss, F. Lehfuss, M. Faschang, P. Palensky, A. A. van der Meer, K. Heussen, O. Gehrke, E. Guillo-Sansano, M. H. Syed, A. Emhemed, R. Brandl, V. H. Nguyen, A. Khavari, Q. T. Tran, P. Kotsampopoulos, N. Hatziargyriou, N. Akroud, E. Rikos and M. Z. Degefa (2017). Simulation-Based Validation of Smart Grids – Status Quo and Future Research Trends, Cham, Springer International Publishing.

Stifter, M., R. Schwalbe, F. Andren and T. Strasser (2013). Steady-state co-simulation with PowerFactory. Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2013 Workshop on.

Stifter, M., E. Widl, F. Andren, A. Elsheikh, T. Strasser and P. Palensky (2013). Co-simulation of components, controls and power systems based on open source software. Power and Energy Society General Meeting (PES), 2013 IEEE, IEEE.

Strasser, T., M. Stifter, F. Andren and P. Palensky (2014). "Co-Simulation Training Platform for Smart Grids." Ieee Transactions on Power Systems **29**(4): 1989-1997.

Swan, L. G. and V. I. Ugursal (2009). "Modeling of end-use energy consumption in the residential sector: A review of modeling techniques." Renewable and Sustainable Energy Reviews **13**(8): 1819-1835.

Taveres-Cachat, E., F. Favoino, R. Loonen and F. Goia (2021). "Ten questions concerning co-simulation for performance prediction of advanced building envelopes." Building and Environment **191**: 107570.

Thomas, D., C. Miller, J. Kämpf and A. Schlueter (2014). Multiscale co-simulation of EnergyPlus and CitySim models derived from a building information model. Bausim 2014: Fifth German-Austrian IBPSA Conference.

TNEI. (2019). "IPSA Power." Retrieved Feb 17, 2019, 2019, from http://www.ipsa-power.com/.

Tol, H. İ. and S. Svendsen (2012). "Improving the dimensioning of piping networks and network layouts in low-energy district heating systems connected to low-energy buildings: A case study in Roskilde, Denmark." Energy **38**(1): 276-290.

Tolk, A. and L. Rainey (2014). Modeling and Simulation Support for System of Systems Engineering Applications.

Trčka, M., J. L. Hensen and M. Wetter (2010). "Co-simulation for performance prediction of integrated building and HVAC systems–An analysis of solution characteristics using a two-body system." Simulation Modelling Practice and Theory **18**(7): 957-970.

Tunzi, M., R. Boukhanouf, H. Li, S. Svendsen and A. Ianakiev (2018). "Improving thermal performance of an existing UK district heat network: A case for temperature optimization." Energy and Buildings **158**: 1576-1585.

UNDESA (2018). World Urbanization Prospects: The 2018 Revision, United Nations Department of Economic Social Affairs New York, NY, USA.

Vaculin, O., W. R. Kruger and M. Valasek (2004). "Overview of coupling of multibody and control engineering tools." Vehicle System Dynamics **41**(5): 415-429.

Vadiee, A., A. Dodoo and L. Gustavsson (2018). A comparison between four dynamic energy modeling tools for simulation of space heating demand of buildings. Cold Climate HVAC Conference, Springer.

Valasek, M. (2008). Modeling, simulation and control of mechatronical systems. Simulation techniques for applied dynamics, Springer**:** 75-140.

Van Dam, K. H. (2009). Capturing socio-technical systems with agent-based modelling.

van Dam, K. H. and J. Keirstead (2010). Re-use of an ontology for modelling urban energy systems. Infrastructure Systems and Services: Next Generation Infrastructure Systems for Eco-Cities (INFRA), 2010 Third International Conference on.

Vesterlund, M., A. Toffolo and J. Dahl (2016). "Simulation and analysis of a meshed district heating network." Energy Conversion and Management **122**: 63-73.

Wade, N. S., P. C. Taylor, P. D. Lang and P. R. Jones (2010). "Evaluating the benefits of an electrical energy storage system in a future smart grid." Energy Policy **38**(11): 7180-7188.

Wang, K. and N. Wade (2021). An integration platform for optimised design and real-time control of smart local energy systems. 2021 12th International Renewable Energy Congress (IREC).

Wang, Z., T. Hong and R. Jia (2019). "Buildings. Occupants: a Modelica package for modelling occupant behaviour in buildings." Journal of Building Performance Simulation **12**(4): 433-444.

Wang, Z., H. Zhang, R. Zhang, Y. Li and B. Xu (2013). "A Run-time infrastructure based on service-distributed architecture."

Wetter, M. (2011). "Co-simulation of building energy and control systems with the Building Controls Virtual Test Bed." Journal of Building Performance Simulation **4**(3): 185-203.

Wetter, M. (2011). "Co-simulation of building energy and control systems with the Building Controls Virtual Test Bed." Journal of Building Performance Simulation **4**: 185-203.

Widl, E., W. Müller, A. Elsheikh, M. Hörtenhuber and P. Palensky (2013). The FMI++ library: A high-level utility package for FMI for model exchange. 2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), IEEE.

Wieland, M., A. Nichersu, S. M. Murshed and J. Wendel (2015). Computing solar radiation on CityGML building data. 18th AGILE international conference on geographic informaton science.

Yao, J. (2014). "Determining the energy performance of manually controlled solar shades: A stochastic model based co-simulation analysis." Applied Energy **127**: 64-80.

Zhou, Q. and J. W. Bialek (2005). "Approximate Model of European Interconnected System as a Benchmark System to Study Effects of Cross-Border Trades." IEEE Transactions on Power Systems **20**(2): 782-788.

Zhu, D. (2012). "Comparison of building energy modeling programs: building loads."

# 8 APPENDICES

# APPENDIX 1 ESSENTIAL FMI CO-SIMULATION API

**fmiComponent fmiInstantiate(fmiString instanceName, fmiString GUID, fmiString fmuLocation, fmiString mimeType, fmiReal timeout, fmiBoolean visible, fmiBoolean interactive, fmiCallbackFunctions functions,fmiBoolean loggingOn)**

Description:

Instantiate the slave FMU. Locate FMU zip archive and then call instantiateModel. A new instance of a co-simulation slave is returned. If a null pointer is returned, then instantiation failed. In that case, "functions->logger" is called and detailed information is logged. A slave can be instantiated many times. This function must be called successfully, before any of the following functions can be called.

Parameters:

*instanceName* is unique identifier for a given FMI Component instance.

*GUID* is the Globally Unique Identifier used to make sure the XML file and the

DLL match.

*fmuLocation* is the path to the FMU zip archive.

*mimeType* represents the MIME type (ietf RFC 2045, 2046, 2047, 2048,

2049) of the 'simulator'. It's ignored in my implementation.

*timeout* is a communication timeout value in milli-seconds to allow inter-process

communication to take place. A timeout value of 0 indicates an infinite wait

period. It's ignored in my implementation.

*visible* indicates whether or not the simulator application window needed to

execute a model should be visible. It's ignored in my implementation.

*interactive* indicates whether the simulator application must be manually started

by the user. It's ignored in my implementation.

*functions* provides callback functions to be used from the model functions to

utilise resources from the environment. It holds function pointers either *logger*,

which is a function that is called usually when the model function encounters some

problem or functions handle memory allocation and deallocation, e.g.

allocateMemory and freeMemory.

*loggingOn* is either fmiTrue or fmiFalse which enables/disables debug logging

respectively

**void fmiFreeInstance(fmiComponent c)**

Description:

Dispose the given instance, unload the loaded model, and free all the allocated

memory and other resources that have been allocated by the functions of the co-

simulation interface.

Parameters:

*fmiComponent* is a pointer to a co-simulation slave specific data structure. It

contains all information needed by the slave to process the co-simulation.

**static fmiComponent instantiateModel(const char* fname, fmiString instanceName, fmiString GUID, fmiCallbackFunctions functions, fmiBoolean loggingOn)**

Description:

Instantiate the model. Then load and analyse model description file, store model parameters and variables in an array and do pre-processing before simulation.

Parameters:

*fname* is fmiInstantiate

*instanceName* is unique identifier for a given FMI Component instance.

*GUID* is the Globally Unique Identifier used to make sure the XML file and the

DLL match.

*functions* provides callback functions to be used from the model functions to

utilise resources from the environment. It holds function pointers either *logger*,

which is a function that is called usually when the model function encounters some

problem or functions handle memory allocation and deallocation, e.g.

allocateMemory and freeMemory.

*loggingOn* is either fmiTrue or fmiFalse which enables/disables debug logging

respectively

**fmiStatus fmiSetReal(fmiComponent c, const fmiValueReference vr[], size_t nvr, const fmiReal value[])**

Description:

Set values of inputs.

Parameters:

*fmiComponent* is a pointer to a co-simulation slave specific data structure. It

contains all information needed by the slave to process the co-simulation.

*ValueReferences* is a handle to a (base type) variable value of the model.

*nvr* is the number of values.

*value* is a vector containing the values that shall be set.

**fmiStatus fmiGetReal(fmiComponent c, const fmiValueReference vr[], size_t nvr, fmiReal value[])**

Description:

Get values of the variables by providing their variable references.

Parameters:

*fmiComponent* is a pointer to a co-simulation slave specific data structure. It

contains all information needed by the slave to process the co-simulation.

*ValueReferences* is a handle to a (base type) variable value of the model.

*nvr* is the number of values.

*value* is a vector containing the values that shall be set.

**fmiStatus fmiDoStep(fmiComponent c, fmiReal currentCommunicationPoint,**

**fmiReal communicationStepSize, fmiBoolean newStep)**

Description:

Run a computation of a time step. It returns fmiOK or fmiError depending on the internal state of the slave and the last call of the function.

Parameters:

*fmiComponent* is a pointer to a co-simulation slave specific data structure. It contains all information needed by the slave to process the co-simulation.

*currentCommunicationPoint* is the current communication point of the master

*communicationStepSize* is the communication step size.

*newStep* is fmiTrue if the last communication step is accepted by the master and a new communication step is started

**fmiStatus fmiGetStatus(fmiComponent c, const fmiStatusKind s, fmiStatus* value)**

Description:

Inform the master about the actual status of the simulation run.

Parameters:

*fmiComponent* is a pointer to a co-simulation slave specific data structure. It contains all information needed by the slave to process the co-simulation.

*fmiStatusKind* isstatus information to be returned.

**fmiStatus fmiCancelStep(fmiComponent c)**

Description:

Called to cancel the current computation.

Parameters:

*fmiComponent* is a pointer to a co-simulation slave specific data structure. It contains all information needed by the slave to process the co-simulation.

**fmiStatus fmiTerminate (fmiComponent c)**

Description:

Called by the master to signal the slave the end of the co-simulation run.

Parameters:

*fmiComponent* is a pointer to a co-simulation slave specific data structure. It contains all information needed by the slave to process the co-simulation.

# APPENDIX 2 FMI CO-SIMULATION FUNCTIONS DEVELOPED FOR SIMULATION TOOLS DEMONSTRATED IN CASES

**int DataStore::addVariable(const std::string &name)**

Description:

Store input and out variables in an associative container

Parameters:

*std::string* is a pointer to input and out variables and corresponding

refererence

**void DataStore::addValue(const std::string &name, const float value)**

Description:

Set input value of the variable by providing their variable reference.

Parameters:

*std::string* is a pointer to input variable reference

*float* is the input value

**float DataStore::getValue(const std::string & name)**

Description:

Provide output value of the variable by providing their variable reference.

Parameters:

*std::string* is a pointer to output variable reference

*float* is the output value

**void Simulation::preprocess()**

Description:

Do pre-processing before simulation by parsing simulation configuration file to get

simulation period and time step of the simulation and input files for simulation

then pre-conditioning building model.

**void Simulation::parseConfiguration(const std::string & file)**
Description:

Parse the simulation configuration xml file to get all the parameters needed in the

simulation.

Parameters:

*file* is the simulation configuration xml file.

**void Simulation::setupSimulationModel()**
Description:

Pre-condition building model.

**void Simulation::preTimeStep()**
Description:

Process before timestep by reading current simulation time and counted

TimeStep.

**void Simulation::timeStep()**
Description:

Run a computation of a time step and increments the timestep for the simulation.

**void Simulation::postprocess()**
Description:

Save simulation results before next time step.

# APPENDIX 3 URBAN ENERGY SIMULATION TOOLS WITH FMU EXPORT SUPPORT

| Simulation tool | Description | FMU export support |
|---|---|---|
| Dymola | It is a proprietary Modelica-based environment that supports modelling and simulation of multi-physics systems. The open-source Modelica-based simulation tools such as and JModelica and OpenModelica are not as advanced, functional, and detailed as Dymola. | Native support the FMU export for co-simulation |
| JModelica | It is an open-source Modelica-based platform for simulation and analysis of complex dynamic systems | Native support the FMU export for co-simulation |
| OpenModelica. | It is an open-source Modelica-based platform for simulation and analysis of complex dynamic systems | Native support the FMU export for co-simulation |
| TRNSYS | It is a popular simulation software for building and HVAC system simulation including solar and geothermal applications. | Using tool trnsys-fmu to support the FMU export for co-simulation |
| PowerFactory | It is an established tool for the modelling and simulation of electrical networks | Using tool powerfactory-fmu to support the FMU export for co-simulation |
| MATLAB/Simulink | It is a popular software for modelling, simulating and analysing dynamical systems | Using the Modelon FMI toolbox to support FMI export for co-simulation |

# APPENDIX 4 CASES CO-SIMULATION VARIABLES

Case1: EnergyPlus and No-MASS co-simulation variables

Geneva single shoe box office

| Inputs (from EnergyPlus to No-MASS) | Outputs (from No-MASS to EnergyPlus) |
|---|---|
| EnvironmentSiteExteriorHorizontalSkyIlluminance<br>EnvironmentSiteRainStatus<br>EnvironmentSiteOutdoorAirDrybulbTemperature<br>ZoneMeanAirTemperature<br>ZoneAirRelativeHumidity<br>ZoneMeanRadiantTemperature<br>DaylightingReferencePoint1Illuminance | NumberOfOccupants<br>WindowState0<br>LightState<br>AverageGains<br>BlindFraction |

Case2: EnergyPlus and No-MASS co-simulation variables

I. Finningley shoe box office 1

| Inputs (from EnergyPlus to No-MASS) | Outputs (from No-MASS to EnergyPlus) |
|---|---|
| EnvironmentSiteExteriorHorizontalSkyIlluminance<br>EnvironmentSiteRainStatus<br>EnvironmentSiteOutdoorAirDrybulbTemperature<br>Building1Block1Zone1ZoneMeanAirTemperature | Building1Block1Zone1NumberOfOccupants<br>Building1Block1Zone1WindowState0<br>Building1Block1Zone1LightState<br>Building1Block1Zone1AverageGains<br>Building1Block1Zone1BlindFraction |

| | |
|---|---|
| Building1Block1Zone1ZoneAirRelative Humidity<br>Building1Block1Zone1ZoneMeanRadiantTemperature<br>Building1Block1Zone1DaylightingReferencePoint1Illuminance | |

II. Finningley shoe box office 2

| Inputs (from EnergyPlus to No-MASS) | Outputs (from No-MASS to EnergyPlus) |
|---|---|
| EnvironmentSiteExteriorHorizontalSkyIlluminance<br>EnvironmentSiteRainStatus<br>EnvironmentSiteOutdoorAirDrybulbTemperature<br>Building2Block1Zone1ZoneMeanAirTemperature<br>Building2Block1Zone1ZoneAirRelative Humidity<br>Building2Block1Zone1ZoneMeanRadiantTemperature<br>Building2Block1Zone1DaylightingReferencePoint1Illuminance | Building2Block1Zone1NumberOfOccupants<br>Building2Block1Zone1WindowState0<br>Building2Block1Zone1LightState<br>Building2Block1Zone1AverageGains<br>Building2Block1Zone1BlindFraction |

III. Finningley shoe box office 3

| Inputs (from EnergyPlus to No-MASS) | Outputs (from No-MASS to EnergyPlus) |
|---|---|
| EnvironmentSiteExteriorHorizontalSkyIlluminance | Building3Block1Zone1NumberOfOccupants |

| | |
|---|---|
| EnvironmentSiteRainStatus<br>EnvironmentSiteOutdoorAirDrybulbTemperature<br>Building3Block1Zone1ZoneMeanAirTemperature<br>Building3Block1Zone1ZoneAirRelativeHumidity<br>Building3Block1Zone1ZoneMeanRadiantTemperature<br>Building3Block1Zone1DaylightingReferencePoint1Illuminance | Building3Block1Zone1WindowState0<br>Building3Block1Zone1LightState<br>Building3Block1Zone1AverageGains<br>Building3Block1Zone1BlindFraction<br>Building1Block1Zone1GainsAppliance |

# APPENDIX 5 MOSAIK SIMULATOR API

**init**(step_size, sim_params, model_config)

Initialise a simulator. The ***init*** call is the first command that mosaik sends to the simulator and it is sent only once. It is used to configure the simulator and to setup the model instances.

step_size is an integer defines how many seconds (in simulation time) step() function will advance on each call.

sim_params is a dict with various additional parameters for the simulation

model_config is a list of tuples describing a certain model configuration:

(cfg_id, model_name, num_instances, params)

Return: The return value meta is a dict that maps model configurations (cfg_id) to a list of dicts which describe the entities for each model instance.

step()

Advance the simulation by step_size and return the current simulation time. The step command has no parameters.

Return: The current simulation time should be returned.

**create**(num, model, init_val)

Initialise a number of simulation model instances within the simulator. It must return a list with some information about each entity created.

num is an integer for the number of model instances to create.

model needs to be a public entry in the simulator's meta['models'].

init_val is the initial value for the model instances.

Return: A list of objects describing the created model instances (entities).

set_data(data)

Set the values of the given attributes for each entity in data.

data is a dict with the key being the entity id for which data is received.

Return: Nothing

**get_data**(model_name, etype, attributes)

Get current values of a number of attributes within a model.

model_name is a string which defines name of the model to query.

etype is a string which defines an entity type within the model. attributes (list) - List of attribute names to query.

attributes is a list of attribute names to query.

Return: A dict that maps entity IDs to data dictionaries. Each data dictionary maps the entities' attribute names to their values.

finalize()

Do some clean-up operations after the simulation finished (e.g. shutting down external processes).

# APPENDIX 6 MOSAIK SCENARIO API

**start**(sim_name, **sim_params)

Start the simulator named sim_name and pass the parameters of the dict sim_params to it.

Return: A mosaik_api.Simulator instance.

**connect**(src, dest, *attr_pairs, async_requests=, time_shifted=, initial_data=)

Connect the src entity to dest entity. Establish a dataflow for each (src_attr, dest_attr) tuple in attr_pairs. If src_attr and dest_attr have the same name, you can optionally pass one of them as a single string. If the dest simulator may make asynchronous requests to mosaik to query data from src (or set data to it), async_requests should be set to True so that the src simulator stays in sync with dest.

Return:

**run**(until=)

Start the simulation until the simulation time until is reached.

Return: The current simulation time (>= until).

# APPENDIX 7 MASTER ALGORITHM EXAMPLE

## Case 1 Single building co-simulation master algorithm

```
sim_config = {
    'EnergyPlus': {
        'python': 'mosaik_energyplus.mosaik:EnergyPlus',
    },
    'NoMASS': {
        'python': 'mosaik_nomass.mosaik:NoMASS',
    },
}

START = '2015-01-01 00:00:00'
END = 365 * 24 * 3600  # 1 year

E2N = ['Block1:Zone1ZoneAirRelativeHumidity', 'Block1:Zone1ZoneMeanRadiantTemperature', 'Block1:Zone1ZoneMeanRadiantTemperature', 'Block1:Zone1DaylightingReferencePoint1Illuminance'
, 'EnvironmentSiteExteriorHorizontalSkyIlluminance', 'EnvironmentSiteRainStatus', 'EnvironmentSiteOutdoorAirDrybulbTemperature'];
N2E = ['Block1:Zone1BlindFraction', 'Block1:Zone1NumberOfOccupants', 'Block1:Zone1LightState', 'Block1:Zone1WindowState0', 'Block1:Zone1AverageGains']

def main():
    world = mosaik.World(sim_config)
    create_scenario(world)
    world.run(until=END)

def create_scenario(world):
    # Start simulators
    energyplus = world.start('EnergyPlus', step_size=300, tStart=0, tStop=END, sim_params={}, model_config=[('cid','office', 1, {'model_path':
    '/home/kunpeng/ssip_development/citi_paper_simulation/energyplus_citi_fmu/office/energyplus_office_citi_8.4_shade/', 'model_name': 'in.fmu'})])
    nomass = world.start('NoMASS', step_size=300, tStart=0, tStop=END, sim_params={}, model_config=[('cid','officeAgent', 1, {'model_path':
    '/home/kunpeng/ssip_development/citi_paper_simulation/nomass_citi_fmu/office/nomass_office_citi_ssip_8.4_shade/', 'model_name': 'agentFMU.fmu'})])

    # Instantiate models
    emodel = energyplus.office.create(1)
    nmodel = nomass.officeAgent.create(1)

    world.connect(emodel[0], nmodel[0], 'Block1:Zone1ZoneAirRelativeHumidity', 'Block1:Zone1ZoneMeanRadiantTemperature', 'Block1:Zone1ZoneMeanAirTemperature',
    'Block1:Zone1DaylightingReferencePoint1Illuminance', 'EnvironmentSiteExteriorHorizontalSkyIlluminance', 'EnvironmentSiteRainStatus',
    'EnvironmentSiteOutdoorAirDrybulbTemperature', async_requests=True)

if __name__ == '__main__':
    main()
```

# APPENDIX 8 GLOSSARY

| Term | Description |
|---|---|
| Algorithm | A well-defined procedure that solves a specific type of problem. |
| Application programming interface (API) | A set of functions, procedures, methods or classes together with type conventions/declarations (for example C header files) that an operating system, library or service provides to support requests made by computer programs. |
| Co-simulation | Coupling two or more simulation programs that share or exchange values to simulate a system consisting of several subsystems. |
| Communication points | Time grid for data exchange between master and slaves in a co-simulation environment (also known as “synchronisation points”). |
| Communication step size | Distance between two subsequent communication points. |
| Composition | A number of interconnected simulators. |
| Composition API | It is an interface that a simulator has to implement for being able to get integrated into the co-simulation platform. |
| Functional mock-up interface (FMI) | A tool-independent standard to support model exchange and co-simulation. It connects the master solver component with one or more slave solvers. |
| Functional mock-up unit (FMU) | An FMU is stored in one zip file consisting basically of one XML file that defines the model variables and a set of FMI functions, in source |

| | |
|---|---|
| | or binary form, to execute the model equations or the simulator slave. In case of tool execution, additionally, the original simulator is required to perform the co-simulation |
| High Level Architecture (HLA) | An industry standard for distributed modelling and simulation |
| Interoperability | Interoperability is the ability of two or more systems or components to exchange information and to use the information that has been exchanged. |
| Master/slave | A method of communication, where one device or process has unidirectional control over one or more other devices. Once a master/slave relationship between devices or processes is established, the direction of control is always from the master to the slaves. In some systems a master is elected from a group of eligible devices, with the other devices acting in the role of slaves. |
| Model | A model is a mathematical or logical representation of a system. Basically, a model is a simplified abstract view of the complex reality. It can be used to compute its expected behaviour under specified conditions. |
| Scenario | It refers to the system or situation being modelled |
| Set point | Temperature limit in a thermal zone |
| Simulation | The process of executing a (computer-based) simulation model. |

| | |
|---|---|
| System | A system is set of interrelated elements considered in a defined context as a whole and separated from their environment. |
| Solver | Software component, which includes algorithms to solve models, e.g. integration algorithms and event handling methods. |
| Simulation tool | Software to solve simulation models. The software includes a solver, may include a user interface and methods for post processing.<br>Examples of simulation programs are: EnergyPlus, No-MASS, etc. |
| Simulation model | A simulation model is an abstract representation of a system and is executed by a simulation environment |
| Simulator | A simulator is an execution environment for a simulation model, which adds the solver functionality to run the model. |
| Software architecture | The fundamental organisation of a system embodied in its (software) components, their relationships to each other, and to the environment, and the principles guiding its design and evolution (IEEE 2000). |
| Software platform | It is collection of subsystems and interfaces that enables the development of end-user products |
| Thermal zone | Combination of structures sharing the same thermodynamic properties |

| | |
|---|---|
| Variable | A value that is used within a simulation that has the potential to change over the duration of the simulation. |