

Ali Mehraj

CREATING ELECTRIC VEHICLE CHARGING SIMULATION WITH SIMCES PLATFORM

M.Sc. Thesis
Faculty of Information Technology and Communication Sciences (ITC)
Supervisor: Prof. Kari Systä
May 2023

ABSTRACT

Ali Mehraj: CREATING ELECTRIC VEHICLE CHARGING SIMULATION WITH SIMCES PLATFORM

M.Sc. Thesis

Tampere University

Master's Degree Programme in Software, Web & Cloud

May 2023

The global number of electric vehicles (EV) is on a continuous rise, alongside the need for an optimal electric vehicle charging solution. One of the effective approaches for discovering a feasible electric vehicle charging solution is to simulate the electric vehicle charging environment and analyze the results.

This research evaluates Simulation Environment of Complex Energy System (SimCES) platform as a simulation development tool for developing electric vehicle (EV) charging simulations. The aim of the research is to evaluate SimCES platform's ability to simulate electric vehicle charging scenarios. The research further addresses issues and limitations discovered during the development of the simulation using the SimCES platform. The research aims to evaluate the suitability of the SimCES platform in terms of electric vehicle charging simulations.

To evaluate SimCES, an EV charging simulation was developed and run using the platform in this research. To achieve that, three components were developed using the platform and a greedy algorithm was implemented in one of the components to calculate the power output for EV charging stations. Issues, drawbacks and limitations were identified during the development phase of the simulation. Additional external applications were developed and integrated with the simulation platform for end-user usage.

The results identify issues and drawbacks related to logging and debugging that caused delays in development. The results further reveal the limitations of the current version of SimCES alongside limitations of the simulation model itself.

The research concludes with suggestions and recommendations for enhancing the SimCES platform and identifies potential future research topics and approaches.

Keywords: Simulation, Co-simulation, Simulation platform, Electric vehicle charging simulation, EV charging simulation, Simulation Environment of Complex Energy System, SimCES.

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

PREFACE

This research evaluates Simulation Environment of Complex Energy System (SimCES) platform as a simulation development tool for developing electric vehicle (EV) charging simulations. The work was carried out as a part of 'EVCommunities' project at Tampere University.

I would like to thank my supervisor, Professor Kari Systä for guiding me throughout my research and writing. I would also like to thank Petri Kannisto for his insightful suggestions to improve my writing. Special thanks goes to my colleagues Ville Heikkilä and Chalith Haputhantrige for helping and assisting me with the development of the research project. Finally, I would like to thank my family who supported me throughout my master's degree studies.

Tampere, 4th May 2023

Ali Mehraj

CONTENTS

1.	Introduction	1
2.	Related Work and Literature Review	3
2.1	Simulations and Co-simulations	3
2.2	EV Charging Simulations, Platforms, and Tools	4
2.3	SimCES Platform.	5
2.3.1	Core Principles and Architecture	5
2.3.2	Message Broker and Messaging	6
2.3.3	Concept of Epochs	7
2.3.4	Platform Components	7
2.3.5	Platform and Image Registry	8
2.3.6	Development Toolkit	11
2.3.7	Deployment	13
2.3.8	Error Handling and Warnings	14
3.	Electric Vehicle Charging Simulation	15
3.1	Components	15
3.2	EV Charging Algorithm	17
3.3	Description	18
3.3.1	Component parameters	18
3.3.2	Message flow	19
3.4	Environment.	21
3.5	Development Process	22
3.5.1	Phase 1: Initialization and Planning	22
3.5.2	Phase 2: Simulation Development	23
3.5.3	Phase 3: Deployment and End-User Interface.	25
3.6	Simulation Scenarios	30
3.6.1	Scenario 1.	30
3.6.2	Scenario 2.	32
3.6.3	Scenario 3.	35
4.	Results and Findings	38
5.	Discussion and Future Work	41
6.	Conclusions	44
	References.	45

LIST OF FIGURES

2.1	Component origins of SimCES platform (Re-published with permission) . . .	8
2.2	Workflow of SimCES platform managed components (Re-published with permission)	9
2.3	Workflow of externally managed components (Re-published with permission)	10
3.1	EV Charging Simulation Components	16
3.2	Logical message flow in electric vehicle charging simulation	20
3.3	GUI Monitor Simulation list	24
3.4	Power output graph in GUI Monitor	25
3.5	Creating a simulation from GUI Monitor	26
3.6	Simulation starter component workflow	27
3.7	Message bus and components in the research simulation	28
3.8	End-user interface in GUI Monitor	29
3.9	Power output and battery status for User 1, User 2, and User 3 in simulation scenario 1	31
3.10	Power output and battery status for User 1, User 2, and User 3 in simulation scenario 2	34
3.11	Power output and battery status for User 1, User 2, User 3 and User 4 in simulation scenario 3	36

LIST OF TABLES

3.1	Input parameters for User component	18
3.2	Input parameters for Station component	19
3.3	Input parameters for Intelligent Controller component	19
3.4	User and station attributes of Simulation Scenario 1	30
3.5	User and station attributes of Simulation Scenario 2	32
3.6	User and station attributes of Simulation Scenario 3	35

LISTINGS

2.1	Manifest of Simple Component [39].	10
2.2	Simulation Tools EpochMessage Class [41].	11

LIST OF SYMBOLS AND ABBREVIATIONS

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CI/CD	Continuous Integration and Continuous Deployment
DER	Distributed Energy Resources
DSS	Distribution System Simulator
EV	Electric Vehicle
GUI	Graphical User Interface
JSON	JavaScript Object Notation
OpenDSS	Open Distribution System Simulator
SimCES	Simulation Environment of Complex Energy System
SOA	Service Oriented Architecture
URL	Uniform Resource Locator
YAML	YAML Ain't Markup Language

1. INTRODUCTION

Simulation is an essential problem-solving methodology for generating effective solutions for many real-world problems [1]. Simulations provide the ability to analyze performance and investigate potential causes in a controlled and reproducible environment [2]. As the global number of electric vehicles continues to increase, the electric vehicle charging environment has become a complex scenario that requires an optimal charging solution. The research aims to create an electric vehicle charging simulation using a simulation platform called SimCES (Simulation Environment of Complex Energy System), find drawbacks and shortcomings of the simulation platform, and evaluate whether the simulation platform is suitable for creating electric vehicle charging simulations. The simulation will be executed by creating multiple independent components within the SimCES platform, which will communicate with each other using a message broker for exchanging messages and data.

In this study, the simulation of electric vehicles (EV) charging consists of an algorithm that determines the charging and the distribution of the energy from the stations and we investigate if such a simulation can be developed using the SimCES platform. In previous research, energy-based simulations were developed using the SimCES platform [3, 4]. However, the simulations were simple in nature and no algorithm was implemented in the calculation for those simulations. Considering that the SimCES platform is a relatively new simulation platform, and a simulation of this scale and complexity has not been generated previously using the platform, it is expected to have some issues and drawbacks while creating such simulations. Therefore, the first research question is,

RQ1. What are the limitations and challenges encountered when building a simulation for EV charging using the SimCES platform?

The second research question in this study centers on the methodology for building the EV charging simulation on top of the SimCES platform, including the tools, methods, and best practices necessary to ensure successful creation of the simulation. It is focused on assessing the compatibility and effectiveness of the SimCES platform in simulating EV charging scenarios. Therefore, the second research question is,

RQ2. How suitable is the simulation platform alongside the simulation model in terms of developing EV charging simulations?

Next, Chapter 2 reviews the related research alongside the SimCES platform. Then, Chapter 3 describes the architecture, development process, and scenarios of the EV charging simulation. Chapter 4 describes the results and findings gathered from creating the simulation. Finally, Chapter 5 discusses the research questions alongside the answers and Chapter 6 concludes the thesis.

2. RELATED WORK AND LITERATURE REVIEW

This chapter surveys simulations alongside co-simulations and reviews the most relevant simulations, platforms, and tools followed by the introduction of the architecture, design, and features of SimCES platform.

2.1 Simulations and Co-simulations

Simulation is “the process of designing a model of a real system and conducting experiments with this model for the purpose of either understanding the behavior of the system and/or evaluating various strategies for the operation of the system” [5]. Simulation enables the replication of the operations of a real-world process or system over time. It is possible to model both existing and conceptual systems using simulations [1]. Simulation can refer to a realization of a representation of a larger, more complex activity [6]. Among the tools available for designing and operating complex systems or processes, simulation is considered to be one of the most effective analytical methods [5]. Simulations have been used to analyze a very large variety of systems [6] and simulation can be a powerful tool if understood and used properly [7]. Simulation can be used to analyze models of arbitrary complexity [6]. Simulations can further serve as a means to investigate novel policies, operating procedures, decision-making protocols, organizational frameworks, and information dissemination channels without disrupting current operations [5]. Through simulation, we can experiment with scenarios about which we have limited knowledge and experience. Simulations enable the ability to investigate hypothetical scenarios [5].

An emerging technology called co-simulation allows for the global simulation of a coupled system by combining the simulations of its component pieces [8]. The linking of two or more simulations that differ in at least one of the areas of the simulation tool, solver algorithm, and step size is known as co-simulation [9]. Co-simulation is a term used to describe the collective simulation of multiple aspects or domains within a system. Co-simulation considers the intricacy of the simulated system and the inter-dependencies among different domains or aspects that are interconnected within the system [10]. Frequently, modular simulation methods are denoted as co-simulation [8]. The theory and methods used in co-simulation allow for the global simulation of a coupled system using a combination of simulators. Each simulator is designed and provided by the team respon-

sible for that system, and operates as a black-box model [11]. The black box refers to a component or system whose internal workings are not known or are intentionally hidden from the observer. Additionally, it makes it easier for system integrators and suppliers to work together since each team or supplier may focus on its portion of the issue with its tools [11].

2.2 EV Charging Simulations, Platforms, and Tools

GridLAB-D is a popular simulation platform that represents a new generation of simulation tools for power distribution systems. It is open-source and a highly adaptable simulation platform that can be seamlessly integrated with a diverse range of external data management and analysis tools [12]. GridLAB-D provides a simulated setting that can be seamlessly incorporated with a diverse range of external utilities, and merges models for both end-user and electrical power distribution automation [13]. GridLAB-D system incorporates various modules that enable it to carry out a range of system simulation tasks, such as; power flow and control functions including the management of distributed generation and storage, the modeling of end-use appliances, equipment, and associated control systems and comprehensive data collection capabilities for every object within the system. This further includes the management of boundary conditions such as weather and electrical boundaries [12].

Previous studies have investigated the potential of utilizing GridLAB-D in conjunction with advanced simulation tools such as DIgSILENT PowerFactory [14] and OpenModelica [15]. The study aimed at establishing a flexible platform that is capable of simulating demand response through various electric vehicle (EV) charging algorithms [16]. The DIgSILENT PowerFactory is a well-established and specialized tool for simulating and analyzing power systems. It is utilized to depict the electric grid and it offers a range of dynamic models and controls that can be scripted to examine, for example, frequency demand response [14]. OpenModelica is a distinct open-source environment for modeling, simulation, optimization, analysis, and development, based on the Modelica and FMI standards, and designed for large-scale integration [15]. The research aimed to improve the charging process by addressing several key areas, including constraints, grid efficiency, decentralized resources, market dynamics, and supply-related considerations [16].

Previous research focused on optimizing the charging process of a combined fleet of electric vehicles in a way that not only reduces the impact on the overall system load but also lowers the costs incurred by customers [17]. The simulation in the research was simulated by utilizing Matlab [18] using Cvx [19] to solve the charging optimizations.

The FEATHERS simulation platform [20] was utilized in a previous research to generate activity-travel schedules. FEATHERS platform can be utilized to create dynamic activity-

based models to forecast transportation demand [20]. The information generated by FEATHERS for every journey in the research includes the scheduled departure time, estimated duration of the trip, as well as the originating and concluding zones. An external application was developed in this study to calculate energy and power requirements in this study. Two scenarios were considered in this study. The first scenario involves people initiating the charging process during the low tariff period as early as possible, which serves as a baseline reference. The second scenario assumes that people initiate the charging process at a time that is uniformly distributed, while ensuring their expenses are minimized [21].

One previous research utilized a hybrid simulator for EV charging scenarios that incorporates discrete and continuous state variables and mimics a group of electric vehicles (EVs) and their connection to the power grid for recharging purposes [22].

With the aim of reducing charging costs, a study was carried out by developing a simulation with Open Distribution System Simulator (OpenDSS) for implementing smart charging of electric vehicles within a smart grid framework that has access to information such as driving patterns, electricity prices, and voltage data [23]. OpenDSS is a power distribution system simulator (DSS) specifically designed to facilitate the integration of distributed energy resources (DER) and modernization of the grid [24]. The approach in the research considers both the real-time price signal and the cost of battery degradation in optimal charging scheduling [23].

2.3 SimCES Platform

2.3.1 Core Principles and Architecture

"Simulation Environment of Complex Energy System (SimCES) is an open-source simulation platform that facilitates the development of complex simulation systems with a modular, message-broker-based architecture" [4]. Prior research [4] compared SimCES with simulation platforms and tools such as iTETRIS [25], Mosaik API [10], DEMKit [26], SimApi [27], MOOSE [28], LICPIE [29], SpaceCRAFT VR [30], Spine Toolbox [31], CO-COP [32] and concludes that these simulation platforms or tools do not possess the combination of loose coupling, platform independence, domain-agnostic design, and the ability to construct a domain ecosystem using simulation components [4]. Furthermore, it was determined in previous research that SimCES is the only platform that enables a component-based domain ecosystem [4].

The simulations in SimCES are configured in a human-readable file that defines all the parameters and components that are included in the simulation [4]. This facilitates the comparison of simulation results in different scenarios. SimCES offers an additional benefit of having a container ecosystem for simulation images, along with a programming

toolkit that streamlines the development process [4].

Service Oriented Architecture or SOA principles [33] are applied in the design of SimCES components. The principles followed in designing SimCES components are loose coupling, abstraction, autonomy, reusability, and composability [3]. Loose coupling allows the components to be independent and have logical dependencies rather than physical dependencies among components. Abstraction enforces only exposing information that is required for interaction. Autonomy hides the implementation details. Reusability allows the components to be used in multiple scenarios and by multiple components. Composability allows a component the ability to be organized into a larger entity consisting of multiple components [3]. SimCES employs Service-Oriented Architecture (SOA) principles along with the message bus paradigm to facilitate the separation of components and runtimes from one another. This approach further enables the addition, removal, or modification of components without affecting other components in the system [4]. Although the energy domain was the initial motivator for SimCES, it is not restricted to any specific application [4].

Additionally, SimCES utilizes a microservice architecture [4]. Microservices represent an architecture that involves partitioning a system into distinct sections based on particular concerns [34]. SimCES can be considered as a manually executed microservice system [4]. This is due to a re-instantiation process being carried out for every individual simulation run in SimCES. The simulations in SimCES run for a fixed period of time rather than having a long uptime. This allows aspects common in traditional microservice architecture such as resilience, fault tolerance, and deployment at runtime to be relaxed [4].

2.3.2 Message Broker and Messaging

SimCES employs a message broker that utilizes a topic-based publish-subscribe pattern to facilitate communication between the components and enable loose-coupling [4]. The components communicate with each other using message queues and topics. A component that produces information and intends to share this information with other components will transmit the information to a designated topic. A string is used to denote or identify this particular topic. Any component that intends to receive the information is required to subscribe to the topic [3]. Message queues are created by the components for receiving messages from the designated topic. In cases where a topic has multiple subscribers, the message bus will generate a copy of published messages in each respective queue. The utilization of a queue-based approach enables asynchronous execution, as delays in message consumption by one subscriber do not impede the execution of other subscribers [3].

SimCES utilizes the Advanced Message Queuing Protocol (AMQP) version 0-9-1 as its communication protocol, which facilitates the geographical dispersion of simulation com-

ponents via the internet [3]. The implementation of AMQP in SimCES is carried out through RabbitMQ [35], with messages being encoded using JavaScript Object Notation (JSON).

2.3.3 Concept of Epochs

In SimCES platform, although the simulation components are distributed, they should still simulate together. For this, a mechanism is needed to synchronize time [36]. Epochs refer to simulated time periods that the platform uses to communicate with the components. An epoch length is not constant and can vary depending on the need of the simulation [4]. For a single simulation, the value of the epoch length can be set to a fixed value depending on the need of the simulation. Each component is only able to start operations when an epoch starts. An epoch is considered complete when all of the components have concluded their operations [3].

At the beginning of each epoch, a message called the "epoch message" is broadcast throughout the network. All the components receive this epoch message. This epoch message originates from a component named 'Simulation Manager' that is responsible for controlling the timing of the epochs [3]. Within the context of an epoch, an epoch message or the output of a component can trigger the operation of another component. A component that does not need any input from others can be triggered by an epoch message [3].

Epochs can only end when all components of the simulations have completed their calculations and operations. After finishing all calculations and operations, each component publishes a 'ready' message. The Simulation Manager receives these ready messages from the component and can only start a new epoch when all the ready messages have been received from all the components [3].

2.3.4 Platform Components

Components refer to the building blocks of a simulation in SimCES platform. There are two types of components in SimCES platform; energy domain components, and platform components. If components refer to real-life counterparts, then the components are called energy domain components. Components that are exclusively present for the simulation are referred to as platform components [3].

The platform components of the SimCES platform are Simulation Manager, Platform Manager, and Logging System. The execution of a simulation run with epochs is coordinated by the Simulation Manager. For each run of the simulation, a new instance of the Simulation Manager is created. The Platform Manager oversees the initialization of each simulation run [3]. The Logging System comprises two components, namely Log Writer and Log

Reader. The Log Writer component is responsible for storing all messages transmitted during a simulation run and the Log Reader component offers a Restful API (Application Programming Interface) that enables retrieval of the saved messages [3]. This API does not require any authentication and log messages can be retrieved directly. User authentication and traffic encryption are not available for the API which may result in security issues if sensitive information is used during a simulation [4]. Additional security may be implemented with the use of a firewall.

2.3.5 Platform and Image Registry

The SimCES platform provides the flexibility to run components either as Docker containers that are managed by the platform or in any other environment that is managed externally. It is possible to apply both approaches for a single simulation [4]. Externally managed components need to be externally managed if it requires a specific platform. These platform components can be constructed locally or obtained from a public image registry. SimCES has three possible sources for its components: externally managed platform, public or private image registry, and local build. The core of the platform, that is responsible for managing the execution and synchronization of components, is implemented as Docker containers [4]. The component origins of SimCES platform are illustrated in Figure 2.1.

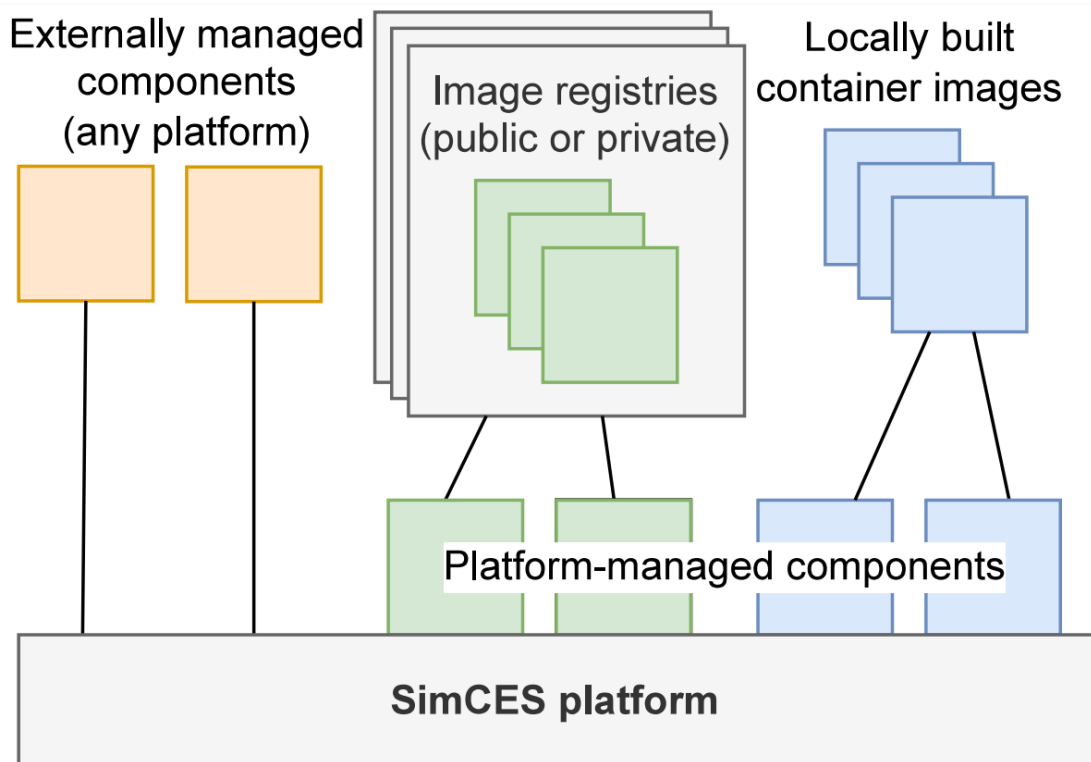


Figure 2.1. Component origins of SimCES platform (Licensed under CC BY 4.0, Republished with permission) [4].

For platform-managed components, the workflow is illustrated in Figure 2.2. The workflow for platform-managed components starts with the user starting the simulation platform by running the start command. The simulation platform starts all the platform-managed components and provides the required parameters to the components. Each platform-managed component starts as a Docker container [37]. The components run the simulation workflow [38]. After finishing the simulation workflow, the components end execution, and the Docker containers are stopped [37].

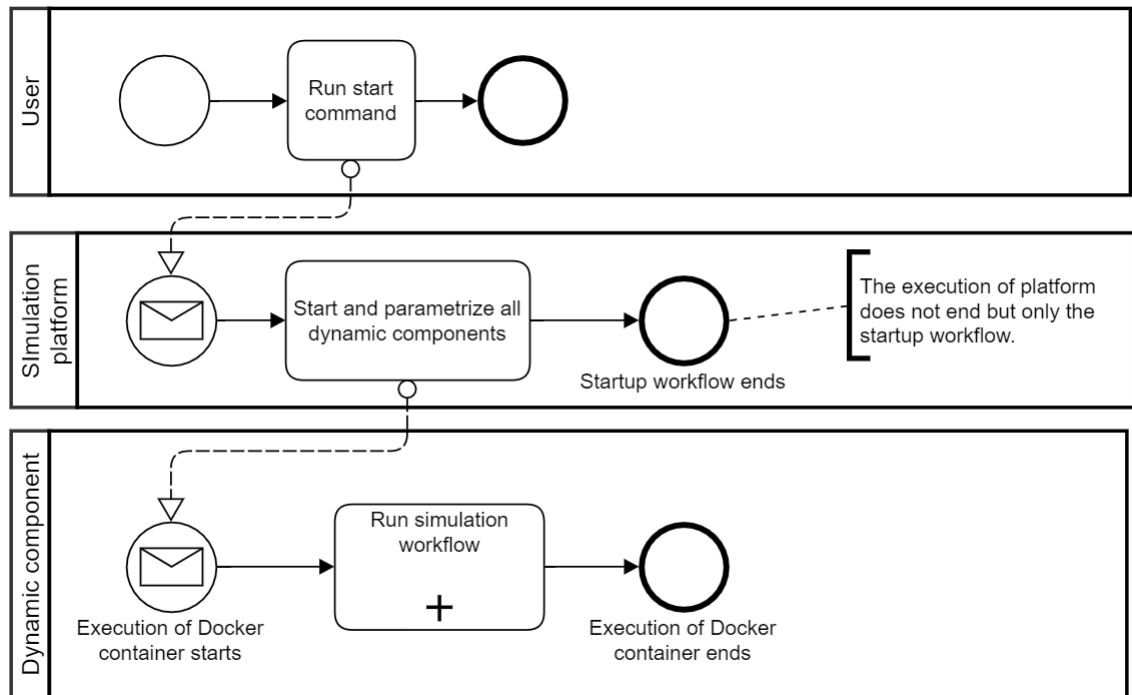


Figure 2.2. Workflow of SimCES platform managed components (Re-published with permission) [37].

For externally managed components, the workflow is illustrated in Figure 2.3. The user needs to ensure that all externally managed or static components are running. After ensuring that all externally managed components are running, the user starts the simulation platform by running the start command. The simulation platform starts and then publishes a Start message [37]. All external components receive this Start message and then run the simulation workflow [38]. After finishing the simulation workflow, it is the responsibility of the user to shut down the externally managed components if needed [37].

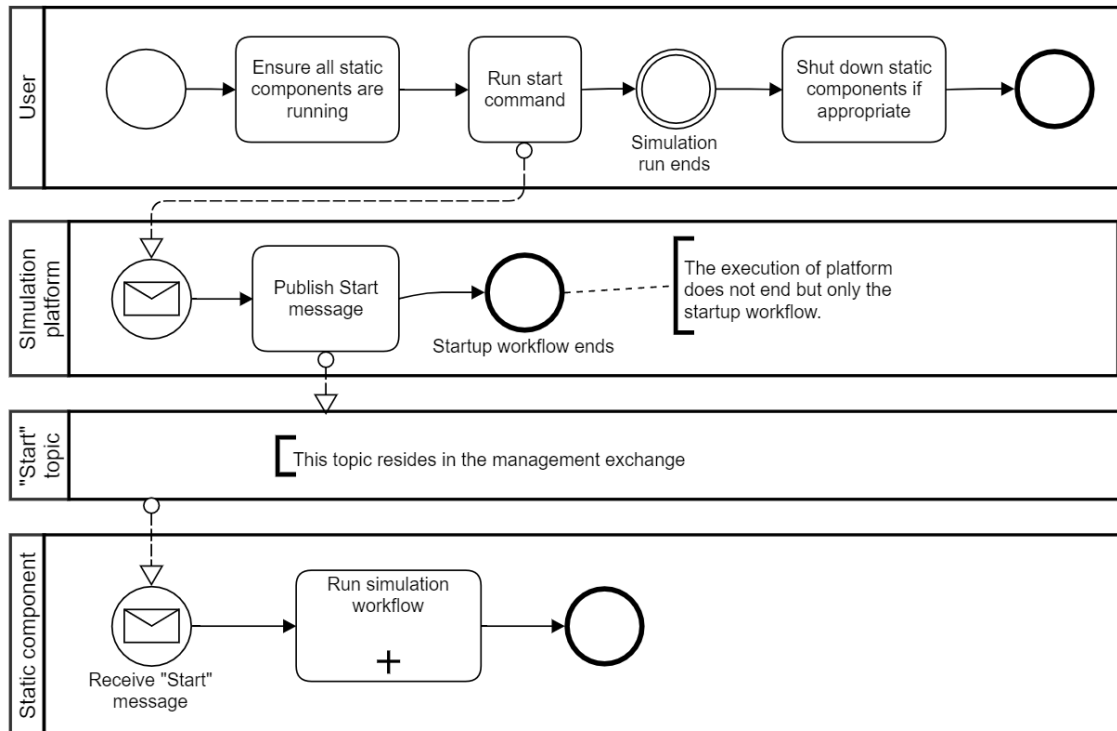


Figure 2.3. Workflow of externally managed components (Re-published with permission) [37].

In order to integrate a newly developed component into the SimCES platform, a manifest needs to be provided. For platform-managed components, the manifest contains the name, type, description, and Docker image name or location of the component [4]. The manifest can additionally define attributes to be delivered at startup that enables the simulations to have starting parameters. These attributes are injected as environment variables for platform-managed components. Dedicated messages containing attributes are transmitted by the broker to externally managed components [4]. YAML (recursive acronym from YAML Ain't Markup Language), is the format of the manifest. The usage of YAML allows even non-technical users to understand and edit the manifests [4]. An example manifest of 'Simple component' is available at [39]. The component contains the name, and description alongside the attributes of the 'Simple component.' The 'type' attribute of the manifest defines how the component is managed. In the Simple component's case, it is managed by the SimCES platform. The Docker image attribute in the manifest corresponds to the location of the component's Docker image. For the Simple component, this corresponds to a link to an image registry where the Docker image is stored. The 'attributes' attribute in the manifest define the input parameters of the component. The manifest of Simple component is displayed in Listing 2.1.

```

1 Name: SimpleComponent
3 Type: platform
Description: "Simple test component for the Decoupled Simulation Platform"

```

```

5 DockerImage: ghcr.io/simcesplatform/simple-component:latest
  Attributes:
7   SimpleValue:
      Environment: SIMPLE_VALUE
9   Optional: false
  InputComponents:
11   Environment: INPUT_COMPONENTS
      Optional: true
13   Default: ""
  OutputDelay:
15   Environment: OUTPUT_DELAY
      Optional: true
17   Default: 0.0

```

Listing 2.1. Manifest of Simple Component [39].

2.3.6 Development Toolkit

The SimCES platform contains a set of tools that implement the fundamental communication requirements. This can enable the developers to implement components faster [4]. The toolkit can be created for other programming languages, as deemed necessary. At present, the toolkit has an implementation in Python and the Python-based version of the toolkit is known as Simulation Tools [40]. Simulation Tools contain proxy classes for message structures that encompass the necessary messages for communicating with the platform [4]. The message classes can be used to handle messages in the simulation platform and the messages contain validation for the message attribute values. The proxy classes provide message classes such as BaseMessage that contains the type of the message, simulation id and the timestamp of a message and EpochMessage that contains the start time and the end time of an epoch. The structure of EpochMessage is displayed in Listing 2.2.

```

2 class EpochMessage(AbstractResultMessage):
      """Class containing all the attributes for a epoch message."""
4   CLASS_MESSAGE_TYPE = "Epoch"
      MESSAGE_TYPE_CHECK = True
6
      MESSAGE_ATTRIBUTES = {
8       "StartTime": "start_time",
          "EndTime": "end_time"
10    }
      OPTIONAL_ATTRIBUTES = []
12
      MESSAGE_ATTRIBUTES_FULL = {
14         **AbstractResultMessage.MESSAGE_ATTRIBUTES_FULL,
          **MESSAGE_ATTRIBUTES

```

```

16     }
    OPTIONAL_ATTRIBUTES_FULL = AbstractResultMessage.OPTIONAL_ATTRIBUTES_FULL +
    OPTIONAL_ATTRIBUTES

18

    @property
20     def start_time(self) -> str:
        """The attribute for the start time of the epoch."""
22         return self.__start_time

24     @property
    def end_time(self) -> str:
26         """The attribute for the end time of the epoch."""
        return self.__end_time

28

    @start_time.setter
30     def start_time(self, start_time: Union[str, datetime.datetime]):
        if self._check_start_time(start_time):
32             new_start_time = to_iso_format_datetime_string(start_time)

34             # Check that the start time is not after the end time.
            if isinstance(new_start_time, str):
36                 if getattr(self, "end_time", None) is not None and new_start_time
                >= self.end_time:
                    raise MessageValueError("Epoch start time ({:s}) should be
20 before the end time ({:s})".format(
38                         new_start_time, self.end_time))
                    self.__start_time = new_start_time
34                 return

42                 raise MessageDateError("'{:s}' is an invalid datetime".format(str(
start_time)))

44     @end_time.setter
    def end_time(self, end_time: Union[str, datetime.datetime]):
46         if self._check_end_time(end_time):
            new_end_time = to_iso_format_datetime_string(end_time)

48             # Check that the end time is not before the start time.
50             if isinstance(new_end_time, str):
                if getattr(self, "start_time", None) is not None and new_end_time
                <= self.start_time:
52                 raise MessageValueError("Epoch end time ({:s}) should be
after the start time ({:s})".format(
54                         new_end_time, self.start_time))
                    self.__end_time = new_end_time
50                 return
56

```

```

        raise MessageDateError("{}:s}' is an invalid datetime".format(str(
end_time)))
58
def __eq__(self, other: Any) -> bool:
60     return (
        super().__eq__(other) and
62         isinstance(other, EpochMessage) and
        self.start_time == other.start_time and
64         self.end_time == other.end_time
    )

66
@classmethod
68 def _check_start_time(cls, start_time: Union[str, datetime.datetime]) -> bool
:
    return cls._check_datetime(start_time)
70

@classmethod
72 def _check_end_time(cls, end_time: Union[str, datetime.datetime]) -> bool:
    return cls._check_datetime(end_time)
74

@classmethod
76 def from_json(cls, json_message: Dict[str, Any]) -> Union[EpochMessage, None
]:
    """Returns a class object created based on the given JSON attributes.
78     If the given JSON does not contain valid values, returns None."""
    if cls.validate_json(json_message):
        return cls(**json_message)
80
    return None

```

Listing 2.2. Simulation Tools EpochMessage Class [41].

Details of other message classes can be found in [42]. The proxy classes feature meta-data fields that are commonly present in messages [4]. The proxy classes create a base structure for creating components and messages that provide ready-made functions and attributes that are needed for the simulation. The toolkit offers an abstract base class for the primary workflow of components that can be extended by the developer for component-specific requirements and functionality [4].

2.3.7 Deployment

The utilization of Docker [43] containers and virtualization in the SimCES platform facilitates easy setup of its components and reduces the need for manual configuration. By utilizing Docker, SimCES allows for applications to be established with minimal lines of code. The applications are run in lightweight Docker containers and Docker Compose [44] can be used to group the containers. Docker Compose enables the grouped containers to be started and stopped using a single command [3].

2.3.8 Error Handling and Warnings

In the case an algorithm within a component fails to generate output due to issues or an unrecoverable error, it is specified that the component reports the error or the failure event to the Simulation Manager [38]. Simulation Manager will signal the other components and end the simulation [3]. If any component is in the process of calculation when the stop message arrives, the component may continue the operation to produce a result and publish the result before quitting [38]. Once this process is complete, it becomes the responsibility of the human user to examine the logs generated by the components to investigate the issue and the errors that occurred [3].

Warnings can occur if there is something wrong in the logic of a process, but this can still produce a meaningful result. Possible causes for warnings can be invalid or unrealistic input data or failure to converge the calculation. Warnings are not fatal and can be informative. However, the presence of a warning in a message can be the cause of a fatal condition in another process. If a warning occurs in a component, the warning should be indicated in the result message [38].

3. ELECTRIC VEHICLE CHARGING SIMULATION

The goal is to simulate an electric vehicle charging environment using SimCES (Simulation Environment of Complex Energy System) platform, analyze shortcomings and difficulties in developing the simulation and analyze the suitability of the SimCES platform for developing electric vehicle charging simulations.

This chapter describes the research simulation components, EV charging algorithm, simulation description, and the simulation environment followed by the simulation development process and the scenarios that are simulated using the SimCES platform.

3.1 Components

The simulation is developed by creating independent components. The environment is built using 3 components: User, Station, and Intelligent Controller. There is one Intelligent Controller in the simulation. There can be multiple User and Station components. The components of the simulation are illustrated in Figure 3.1.

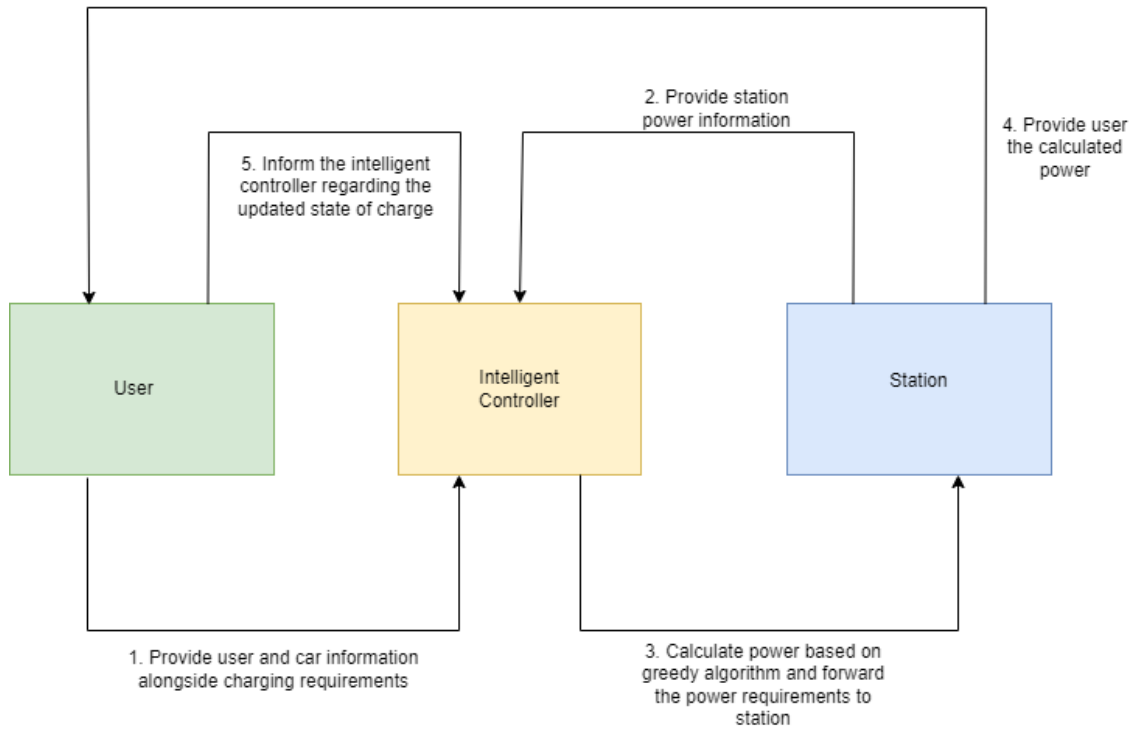


Figure 3.1. EV Charging Simulation Components

User components represent the users alongside their EV cars in the simulation. The role of each User component is to contain and send information regarding the user, user's car and charging requirements to the Intelligent Controller component in each epoch. The User component is responsible for informing the Intelligent Controller component the updated state of charge of the user's car at the end of each epoch.

Station components represent EV charging stations where the users can connect their EV cars for charging. Each Station component contains information about the station and sends the information to the Intelligent Controller component in each epoch. The Station components further provide the power output information to the User components during each epoch.

The Intelligent Controller represents the charging and power output logic controller. The Intelligent Controller component holds all the logic and a greedy algorithm for processing user charging requirements and calculating power output for the stations. The Intelligent Controller component receives information regarding the users, users' cars and charging requirements from the User components and information regarding the stations from the Station components. The Intelligent Controller component also stores the information to process the power output for the stations in each epoch.

3.2 EV Charging Algorithm

The simulation uses a greedy algorithm to calculate the power requirement for each station that determines the final power output to the users' cars from the stations. The Intelligent Controller uses this greedy algorithm and prioritizes power output for the stations based on the following aspects:

1. The earliest leaving time of the user from the station.
2. The highest total energy requirement by the user.
3. The order the users reported in for the current epoch. The earlier reporter is prioritized over later ones.

For the first aspect, if a user leaves earlier than the other users then the user gets the most priority for charging. The users in each epoch are simultaneously sorted based on their leaving time from the stations. The earliest leaving user gets the most priority, and the latest leaving user gets the least priority.

For the second aspect, the highest total energy requirement by a user is the first deterministic factor of the first tiebreaker. If a user has the highest total energy requirement, the user gets the most priority for charging. Similarly, if a user has the lowest power requirement, the user gets the least priority.

For the third aspect and the final tiebreaker, the reporting order of the users in the current epoch is the deterministic factor for prioritization. If a user reports first in an epoch, the user will get the most priority. Similarly, if a user reports the last in an epoch, the user will get the least priority.

The power output for each station is dependent on a few factors such as the total maximum power of the stations, the maximum power output of the connected station, the maximum power input of the user's car, and the power required to reach the target capacity within the current epoch for the user. The amount of power output depends on the following aspects:

1. The maximum power output of the charging station that the user is connected to.
2. The maximum power input capacity of the user's car.
3. Remaining total power during the epoch. The calculation involves subtracting the allocated power from the total maximum power during the epoch.
4. The power needed to achieve the target capacity within the current epoch, as applicable to the user.

The power output for the selected station is determined as the minimum value among the four aspects.

3.3 Description

3.3.1 Component parameters

The User component has 10 input parameters. These parameters are needed to initialize the component with starting values for the first epoch. The input parameters for User components are user id, user model, car model, car battery capacity, car max power, station id, state of charge, target state of charge, arrival time, and target time. Details of the User component input parameters are displayed in Table 3.1.

Input parameter	Datatype	Unit	Description
User id	Integer		Unique id of a user
Username	String		Name of the user
Car model	String		Model of the user's car
Car battery capacity	Float	kWh	The capacity of the battery of the user's car
Car maximum power	Float (>0)	kW	Maximum power input of user's car
Station id	String		The unique id of the station that the user's car connects to during the simulation
State of charge	Float (0-100)	%	User's car's current state of charge
Target state of charge	Float (0-100)	%	User's car's target state of charge at the end of charging
Arrival time	ISO 8601 datetime		The time when the user arrives at the station
Target time	ISO 8601 datetime		The time when the user stops charging the car and leaves the station

Table 3.1. Input parameters for User component

The Station component has 2 input parameters. These parameters are needed to initialize the component with starting values for the first epoch. The input parameters for Station components are station id and maximum power. Details of the station component input parameters are displayed in Table 3.2.

Input parameter	Datatype	Unit	Description
Station id	String		Unique id of a station
Maximum power	Float	kW	The maximum power output of the station

Table 3.2. *Input parameters for Station component*

The Intelligent Controller component has only one input parameter of total maximum power. Details of the Intelligent Controller component input parameters are displayed in Table 3.3.

Input parameter	Datatype	Unit	Description
Total maximum power	Float	kW	Total maximum output power across all the stations in the simulation during each epoch

Table 3.3. *Input parameters for Intelligent Controller component*

3.3.2 Message flow

All the components in the simulation communicate with each other using RabbitMQ message topics. Each message topic names and payloads are unique. Each component publishes and subscribes to certain topics during an epoch. There are a total of 6 messages in the simulation. These messages are car metadata message, car state message, power output message, power requirement message, station state message, and user state message. The message flow in the simulation is illustrated in Figure 3.2.

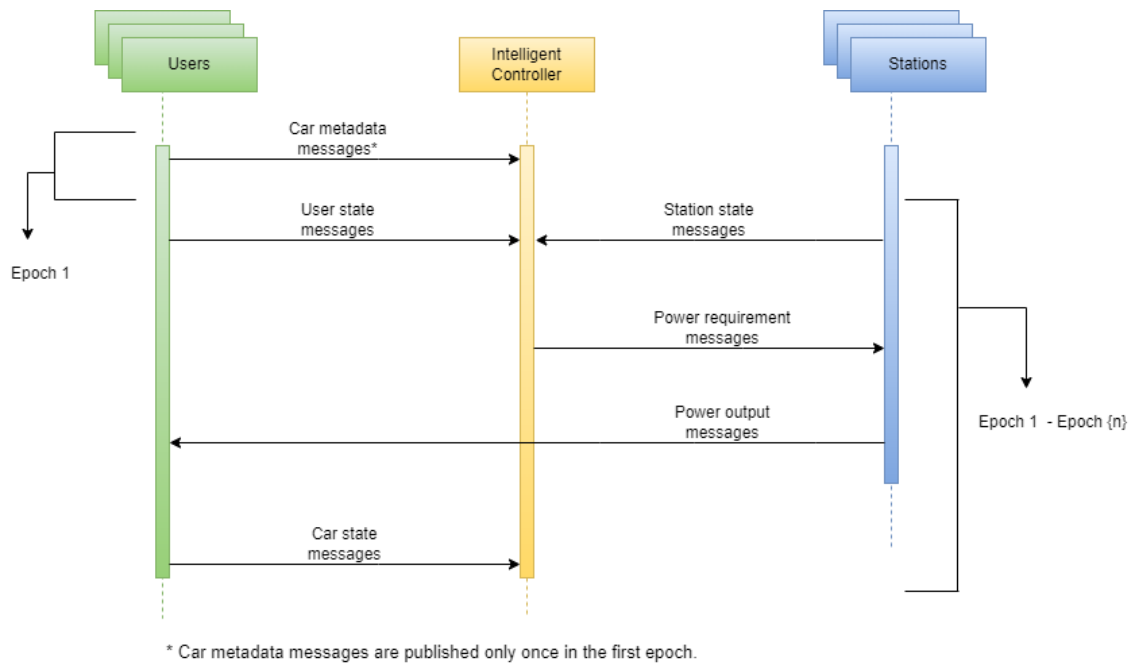


Figure 3.2. Logical message flow in electric vehicle charging simulation

It is important to note that the message flow depicted in Figure 3.2 represents the logical sequence of the simulation. However, in practice, the order in which the messages containing car metadata, station state, and user state are published may differ, since all these messages are published by the User and the Station components at the beginning of an epoch.

In logical sequence, the initial message of the simulation is the car metadata message, and it is published by the User components. This message is published at the start of the simulation in the first epoch and published only once. The Intelligent Controller subscribes to the car metadata message and saves the car information of the user upon receiving the car metadata message. The car metadata message payload contains the user id, username, station id, state of charge, car battery capacity, car model, and car maximum power.

The Station components publish the station state messages. The message contains information about the state of the station. This message payload contains the station id and the maximum power of the station. The Intelligent Controller subscribes to the station state message, receives the station message, and saves the information regarding the station upon receiving the message.

The User components publish the user state messages, and the Intelligent Controller subscribes to the user state messages. The user state messages contain the charging requirements for the users' cars. The payload of the user state messages contains the user id, target state of charge, arrival time, and target time.

After receiving the car metadata, station state, and user state messages, the Intelligent Controller performs computations using a greedy algorithm designed specifically for this simulation that calculates the power output of the stations. Intelligent Controller then publishes the power requirement message for each station in the simulation. The greedy strategy exhibits a myopic approach by always selecting the element that appears best based on pre-defined criteria. In other words, among all the admissible elements, the highest-weighted element is selected and added to the solution [45]. It should be noted that the greedy algorithm is designed solely for calculating the power output of the stations in the current version of the simulation. This algorithm can be modified, or other means of calculation can be added for different versions of the simulation.

The Station components subscribe to the power requirement messages and process the power requirement message concerning their corresponding station ids. The payload of the power requirement messages contains power, station id, and user id. Based on the power requirement message, the station components publish the power output messages and the user components subscribe to these messages. The payload of the power output message contains the user id, station id, and power output. Upon receiving the power output message based on the user id the user components update their car's current state of charge.

The User components then publish the car state message containing their cars' updated state of charge. The Intelligent Controller component subscribes to the car state messages. The payload of the car state messages contains the user id, station id, and the state of charge of the cars. The Intelligent Controller receives the message and saves the updated state of charge. The epoch ends after the information is saved. The process is repeated till the end of the simulation for the number of epochs specified in the simulation.

3.4 Environment

The simulation components and messages are created using Python version 3.7.9. Docker and Docker Compose are used for containerization. The simulation uses MongoDB version 4.2.7 for storing simulation data [46]. The data that was saved for the simulation can be accessed from Mongo Express [47] using a browser. RabbitMQ version 3.8.4 is used as the message broker for the simulation. Git is used for version control. The logs of the simulation can be accessed using the log reader user interface using a web browser.

For viewing the simulations in graphs, an external application is developed using Node.js version 16.17.1 and React version 18.2.0. Node.js is an open-source runtime environment for JavaScript that can be used on various platforms [48]. React is a JavaScript library for building user interfaces [49]. Node.js is used to develop the backend of the application and React is used to develop the front end of the application. The Node.js server fetches simulation data from MongoDB and the React application displays the simulation

data in graphs. The graphs are created using Charts.js. Charts.js is a JavaScript charting library [50].

3.5 Development Process

The agile development process was used to create the simulation. Agile methods promote an iterative mechanism for producing software, and these methods further increase the iterative nature of the software lifecycle by tightening the design-code-test loop [51]. Kanban was used as the project management methodology. Kanban applies Lean principles by providing a tool to optimize an outcome for value through a focus on flow management [52]. Trello was used as the Kanban board. Trello is a visual tool that is used to manage projects, workflow, or task tracking [53]. Meetings were held weekly and a weekly progress report in the form of a demonstration was presented and discussed to review the work's advancement. The demonstration was thoroughly analyzed for the identification of bugs, issues, and areas for further improvement. Additional tasks were assigned based on the feedback received during the meetings.

The development process was divided into three phases. Phase 1 included planning the simulation, Phase 2 included the development and testing of the simulation, and Phase 3 included deployment and creating the demo of the simulation.

3.5.1 Phase 1: Initialization and Planning

This phase was the initial stage of the simulation development process. This phase consists of planning and sketching out the simulation. Weekly meetings are held to brainstorm the following:

- Components needed in the simulation. It was decided that three components User, Station, and Intelligent Controller are required for the simulation.
- Messages that are needed in the simulation. These messages are required for the components to communicate and share information. These messages are car metadata message, car state message, power output message, power requirement message, station state message, and user state message.
- The payloads of the messages in the simulation.
- Message flow within the simulation. The message flow is illustrated in Figure 3.2.
- Input parameters for the components. These are starting input parameters provided to each component at the start of the simulation.
- Algorithm for power distribution. The greedy algorithm is implemented in the Intelligent Controller component that is responsible for calculating the power distribution for each station.

- Simulation scenarios. Three simulation scenarios were selected to be simulated.

3.5.2 Phase 2: Simulation Development

Phase 2 consists of the development of the simulation. The developers of the simulation had no prior knowledge of SimCES platform and therefore they had to go through the documentation that is available on SimCES platform's GitHub page. Two developers were involved in the development of the simulation and both of the developers had prior experience in programming. The developers did not have prior programming experience with Python. However, the developers had prior experience of utilizing Docker and Docker Compose.

For local development of the simulation, the simulation platform needs to be installed in the local environment of the developer's machines. There is a prerequisite of having Docker and Docker Compose to be installed in the local environment before attempting the installation of the simulation platform. There are instructions on the SimCES GitHub page to assist the users to install the simulation platform. Using the instructions, the developers were able to install the platform on their local machines without any issues. For creating the simulation, the messages for the simulation were created first as the components require the messages to communicate with each other. The user and the station components were created afterward with a dummy Intelligent Controller to send and receive messages from the user and station components.

For debugging the components, the only way to check the logs of the component is to run an existing shell script to copy the logs to log files for each component. The Docker containers are auto-deleted after the simulation has exited so there is no way to check the container logs. Running the shell script creates log files for each component and component logs can be checked from there for debugging.

Following the development of the initial working version of the application with the dummy Intelligent Controller, the algorithm and logic for power distribution were developed and subsequently integrated into the Intelligent Controller component. After this stage, it was possible to run the simulation locally using Docker containers.

For displaying the power output of the simulation in graphs, an external application was developed using Node.js and React. The external application is called the GUI Monitor in this simulation. The GUI Monitor fetches the simulation data using the Rest API provided by the Log Reader component. Additionally, the GUI Monitor displays the list of previously run simulations and the ids of the simulations. The list of simulations in the GUI Monitor is illustrated in Figure 3.3.

NAME	SIMULATION ID	DESCRIPTION	EPOCHS	END TIME
Test simulation	2022-10-18T14:24:07.437Z	Test simulation for testing that the core components have been installed correctly.	12	2022-10-18T14:26:06.466000Z
Test simulation	2022-10-18T14:25:04.288Z	Test simulation for testing that the core components have been installed correctly.	12	2022-10-18T14:27:08.432000Z
Energy community demo	2022-10-18T14:30:14.737Z	This scenario describes the behavior of a small rural energy community of four detached houses. The houses have no controllable loads or storages but two of them have a small scale PV systems installed and one has a electric vehicle charging. Scenario spans over one day with hourly resolution (epochs).	24	2022-10-18T14:32:00.331000Z
Test simulation	2022-10-18T16:11:19.541Z	Test simulation for testing that the core components have been installed correctly.	12	2022-10-18T16:13:04.074000Z
Test simulation 2	2022-10-20T10:18:14.149Z	Test simulation 2 for Component testing	12	2022-10-20T10:20:34.822000Z
Test simulation 2	2022-10-20T11:16:11.929Z	Test simulation 2 for Component testing	12	2022-10-20T11:16:34.379000Z
Test simulation 2	2022-10-20T11:52:27.072Z	Test simulation 2 for Component testing	12	2022-10-20T11:52:45.012000Z

Figure 3.3. GUI Monitor Simulation list

With the application, users can view power distribution graphs for each epoch using the unique simulation id that is generated for each simulation. The simulation and the external application are tested using exploratory testing for detecting bugs and issues. Exploratory testing is any testing where the tester actively controls the design of the tests. These tests are performed, and the information gained while testing is used to design new and better tests [54]. The power output of a simulation in the GUI Monitor graph is illustrated in Figure 3.4.

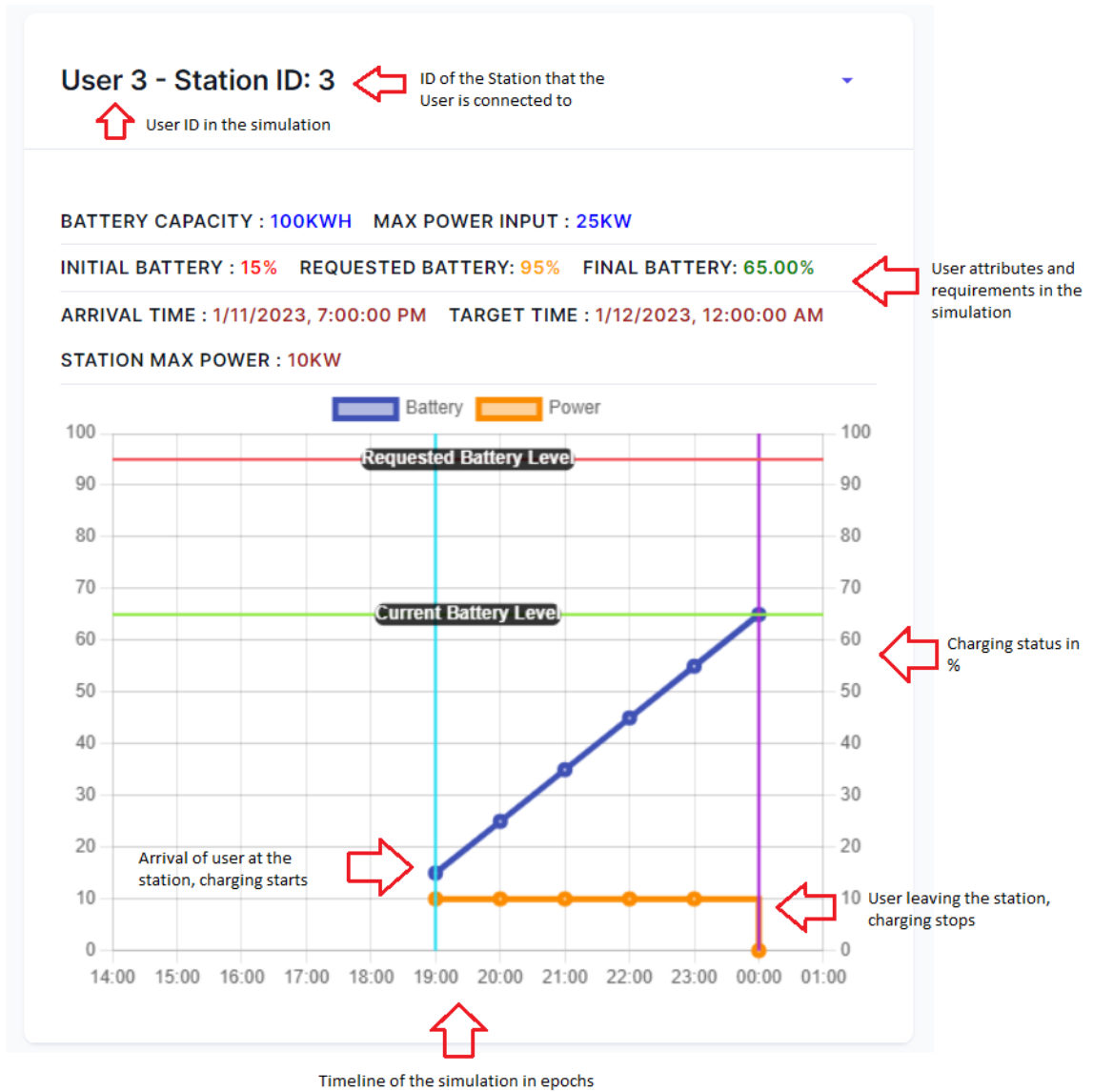


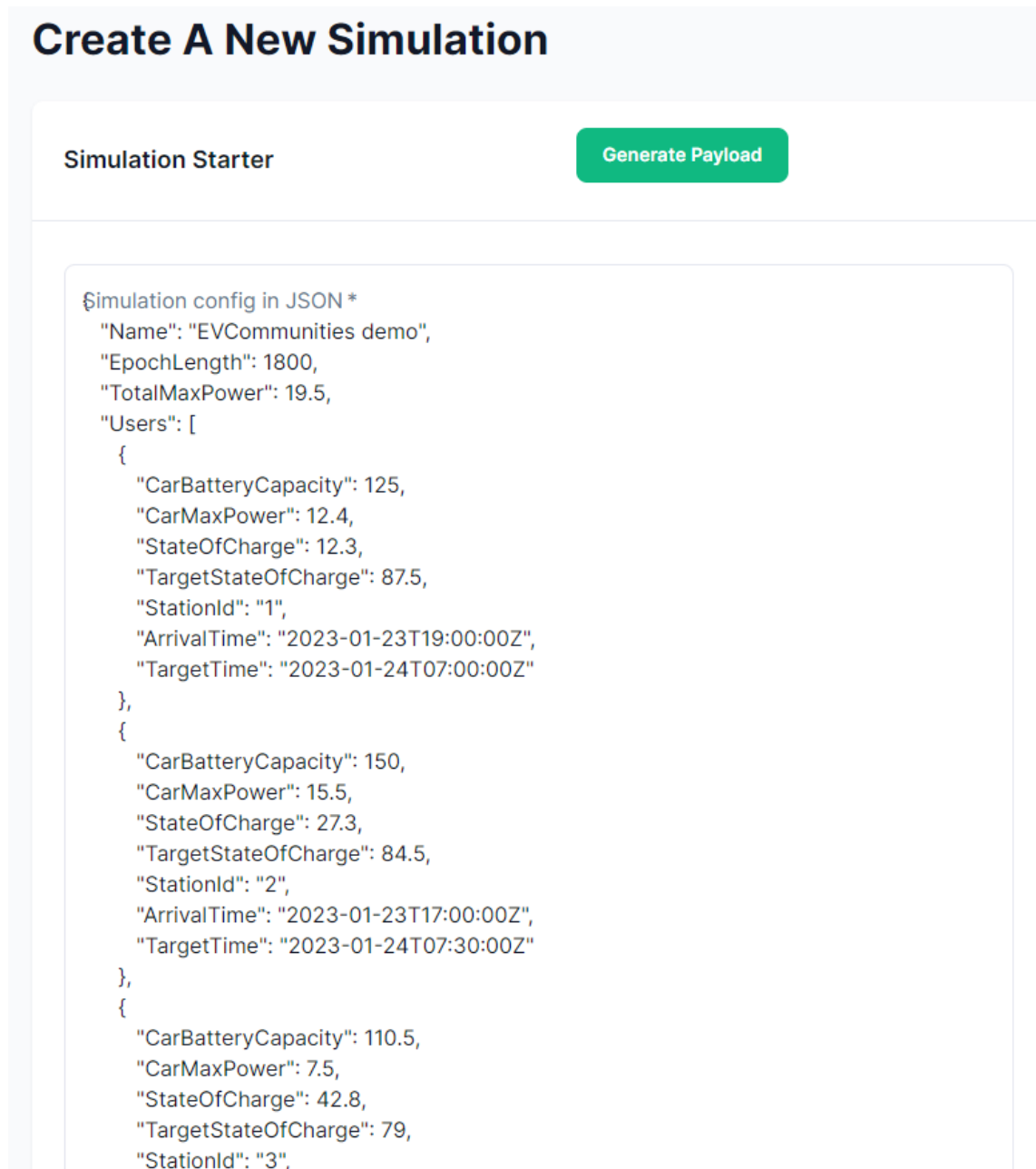
Figure 3.4. Power output graph in GUI Monitor

3.5.3 Phase 3: Deployment and End-User Interface

In Phase 3, the implementation of the simulation and GUI Monitor are deployed on a server, alongside the creation of Continuous Integration and Continuous Deployment (CI/CD) pipelines on GitHub by utilizing GitHub actions. GitHub actions can be used to automate, customize, and execute software development workflows [55]. To enable simulations to run on the server through a simulation starter, modifications are made to the external application. The GUI Monitor will be accessible through the internet and will require basic access authentication, which is a method of verifying user identity by requesting a username and password.

Additionally, a simulation can be initiated from the GUI Monitor by inputting component

names and initial variables in JSON format on the Start Simulation page. It is possible to generate a template payload by clicking the 'Generate Payload' button on the page that helps the user with a ready-made format for the simulation. The user can then change the values in the template as required by the desired simulation. The start simulation page is illustrated in Figure 3.5.



Create A New Simulation

Simulation Starter Generate Payload

```

Simulation config in JSON *
{
  "Name": "EVCommunities demo",
  "EpochLength": 1800,
  "TotalMaxPower": 19.5,
  "Users": [
    {
      "CarBatteryCapacity": 125,
      "CarMaxPower": 12.4,
      "StateOfCharge": 12.3,
      "TargetStateOfCharge": 87.5,
      "StationId": "1",
      "ArrivalTime": "2023-01-23T19:00:00Z",
      "TargetTime": "2023-01-24T07:00:00Z"
    },
    {
      "CarBatteryCapacity": 150,
      "CarMaxPower": 15.5,
      "StateOfCharge": 27.3,
      "TargetStateOfCharge": 84.5,
      "StationId": "2",
      "ArrivalTime": "2023-01-23T17:00:00Z",
      "TargetTime": "2023-01-24T07:30:00Z"
    },
    {
      "CarBatteryCapacity": 110.5,
      "CarMaxPower": 7.5,
      "StateOfCharge": 42.8,
      "TargetStateOfCharge": 79,
      "StationId": "3",
    }
  ]
}

```

Figure 3.5. Creating a simulation from GUI Monitor

To initiate a new simulation from GUI Monitor, a new external component called the Simulation Starter is developed. The Simulation Starter is responsible for validating the input that is received from the GUI Monitor using a Rest API. After validating the input, the Simulation Starter generates the YAML file needed to start the simulation. The Platform

Manager Docker container is then started using the generated YAML file and the simulation is started. The workflow of the Simulation Starter component is illustrated in Figure 3.6.

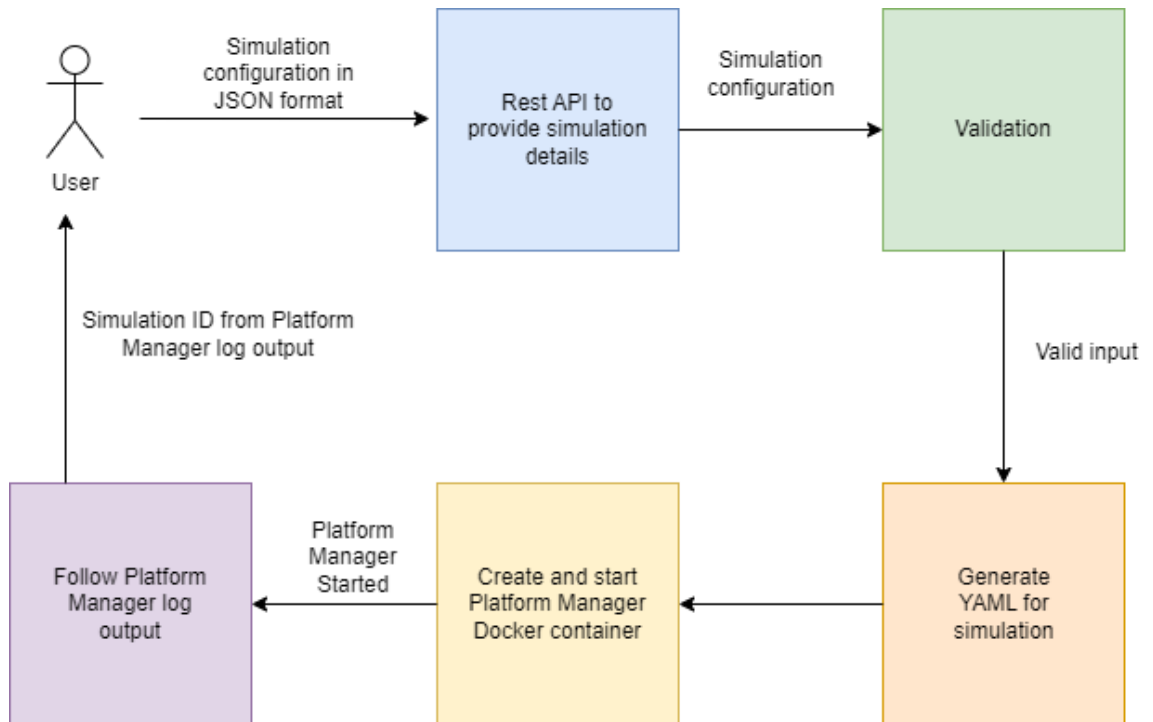


Figure 3.6. Simulation starter component workflow

With the addition of the Simulation Starter component, the final architecture of the simulation is illustrated in Figure 3.7.

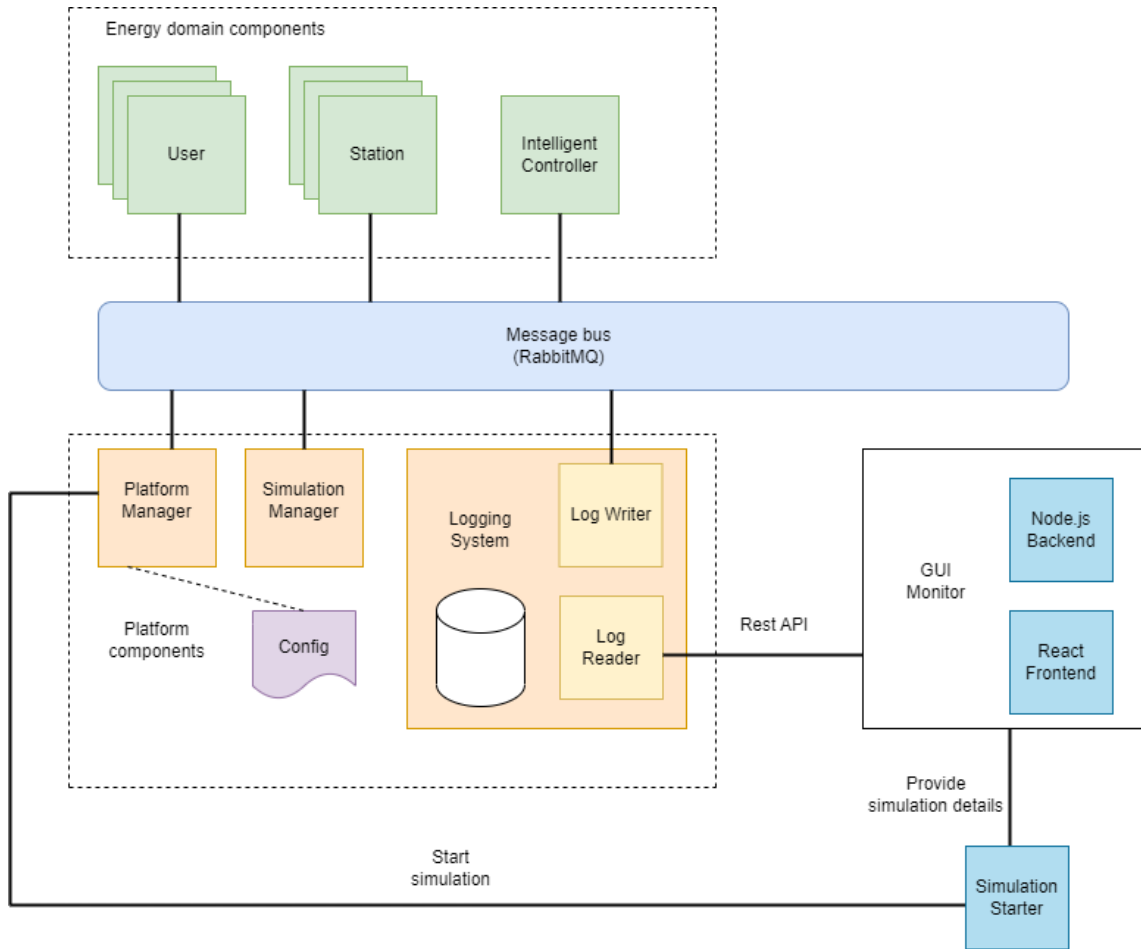
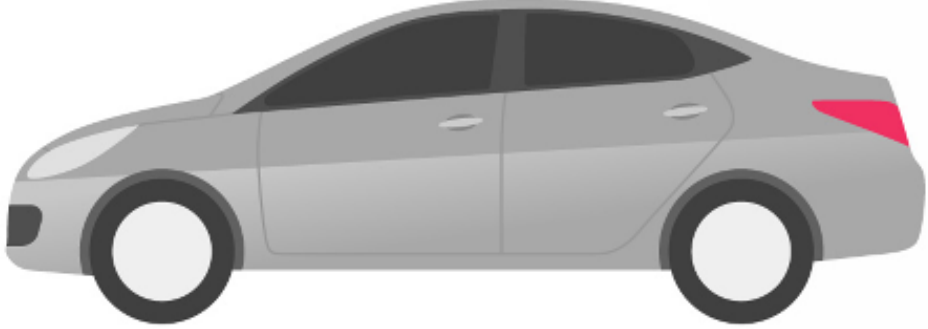


Figure 3.7. Message bus and components in the research simulation

To replicate a real-world scenario where users request charging from charging stations, both end-user and admin views are established within the GUI Monitor. The admin can initiate a simulation and the end-user interface can be used to enter the user's charging requirements for the simulation. This allows for a multi-user mode for initiating the simulation. The end-user interface of the GUI Monitor is illustrated in Figure 3.8.

Schedule New Charging for User1




BATTERY CAPACITY : 90KWH **MAX POWER INPUT : 9KW**

Current State

Current Battery Percentage

5

Arival Time


04/09/2023 01:00 PM 

Requesting State

Requesting Battery Percentage

95

Target Time

04/09/2023 05:00 PM 

Submit

Figure 3.8. End-user interface in GUI Monitor

3.6 Simulation Scenarios

For this research, three main simulation scenarios were created and tested. The first is the simplest scenario, there are 3 stations and 3 users, and all users arrive at the start and leave at the end of the simulation. The second scenario has 3 stations and 3 users, and the users arrive in the middle of the simulation. The third scenario has 3 stations and 4 users, and one user arrives in the middle of the simulation. In the third scenario, one station is used by 2 users. The second user arrives at the station after the first user leaves.

3.6.1 Scenario 1

The scenario has 3 stations, 3 users, and 1 Intelligent Controller. The number of epochs in the simulation is 12 and the length of each epoch is 3600 seconds or 1 hour.

User and station attributes of the simulation are displayed in Table 3.4.

User ID	Connected Station ID	Car battery capacity	Car maximum power input	Station maximum power output
User 1	Station 1	80 kWh	22 kW	20 kW
User 2	Station 2	90 kWh	12 kW	18 kW
User 3	Station 3	98 kWh	25 kW	15 kW

Table 3.4. User and station attributes of Simulation Scenario 1

The total maximum power output for all stations is 30.0 kW. The total maximum power refers to the maximum combined power output of all stations during each epoch. The simulation starts at 2 pm.

Users' requirements in the scenario:

- User 1 arrives at the start of the simulation to Station 1 and requests a target state of charge of 80% from the initial charge of 20%. The target time for User 1 is 12 hours later at 2 am the next day.
- User 2 arrives at the start of the simulation to Station 2 and requests a target state of charge of 70% from the initial charge of 5%. The target time for User 2 is 12 hours later at 2 am the next day.
- User 3 arrives at the start of the simulation to Station 3 and requests a target state of charge of 95% from the initial charge of 2%. The target time for User 3 is 12 hours later at 2 am the next day.

The Intelligent Controller uses a greedy algorithm and prioritizes power output for the stations based on the algorithm. The graphs of the simulation can be displayed using the React application. The power output and battery percentage for the simulation are illustrated in Figure 3.9 for User 1, User 2, and User 3.



Figure 3.9. Power output and battery status for User 1, User 2, and User 3 in simulation scenario 1

According to the greedy algorithm, the earliest leaving time gets the most priority. But in this case, all users' arrival and leaving times are the same. All users; User 1, User 2, and

User 3 arrive at Station 1, Station 2, and Station 3 respectively at the start of epoch 1.

Power output in the scenario:

- User 3 gets the most priority and gets 15 kW power output in the first 4 epochs. This 15-kW power output is limited due to the station's maximum power output of 15 kW. The priority for the first 4 epochs is based on the higher energy required to reach the target. In epoch 5, the energy requirement for User 3 to reach the target is the lowest. Thus, User 3 gets the least priority and gets 0 kW power in the 5th epoch. In the 6th epoch the energy requirement is again highest for User 3 and User 3 gets the most priority for power output and gets 14.4 kW power that fulfills the total target of the user. User 3's requirements are fulfilled at the end of epoch 6. User 3 leaves Station 3 at the end of epoch 12 with a final battery percentage of 95.0. User 3's requirements are fulfilled.
- User 1 gets the highest priority for charging in epoch 5, as the total energy requirement for User 1 is the highest. User 1 receives a power of 20 kW in epoch 5. User 1 leaves Station 1 at the end of epoch 12 with a final battery percentage of 80.0. User 1's requirements are fulfilled.
- User 2 receives a 10 kW power output in the first two epochs. In the third epoch, User 2 gets the least priority and receives no power due to having the least energy required to reach the total target. In epoch 4, User 2 starts receiving power again as the energy required to reach the total target is relatively higher than User 1 in epoch 4. User 2 leaves Station 2 at the end of epoch 12 with a final battery percentage of 70.0. User 2's requirements are fulfilled.

3.6.2 Scenario 2

The scenario has 3 stations, 3 users, and 1 Intelligent Controller. The number of epochs in the simulation is 10 and the length of each epoch is 3600 seconds or 1 hour.

User and station attributes of the simulation are displayed in Table 3.5.

User ID	Connected Station ID	Car battery capacity	Car maximum power input	Station maximum power output
User 1	Station 1	80 kWh	22 kW	20 kW
User 2	Station 2	90 kWh	15 kW	18 kW
User 3	Station 3	95 kWh	20 kW	14 kW

Table 3.5. User and station attributes of Simulation Scenario 2

The simulation starts at 2 pm. The total maximum power output for all stations is 35.0 kW.

- User 1 arrives at epoch 1 at Station 1 at 2pm and requests a target state of charge of 90% from the initial charge of 1%. The target time for User 1 is 5 hours later at 7 pm.
- User 2 arrives at the start of epoch 3 at 4pm at Station 2 and requests a target state of charge of 95% from the initial charge of 4%. The target time for User 2 is 5 hours later at 9 pm.
- User 3 arrives at the start of epoch 4 at 5pm at Station 3 and requests a target state of charge of 95% from the initial charge of 15%. The target time for User 3 is 5 hours later at 10 pm.

The power output and battery percentage for the simulation are illustrated in Figure 3.10 for User 1, User 2, and User 3.

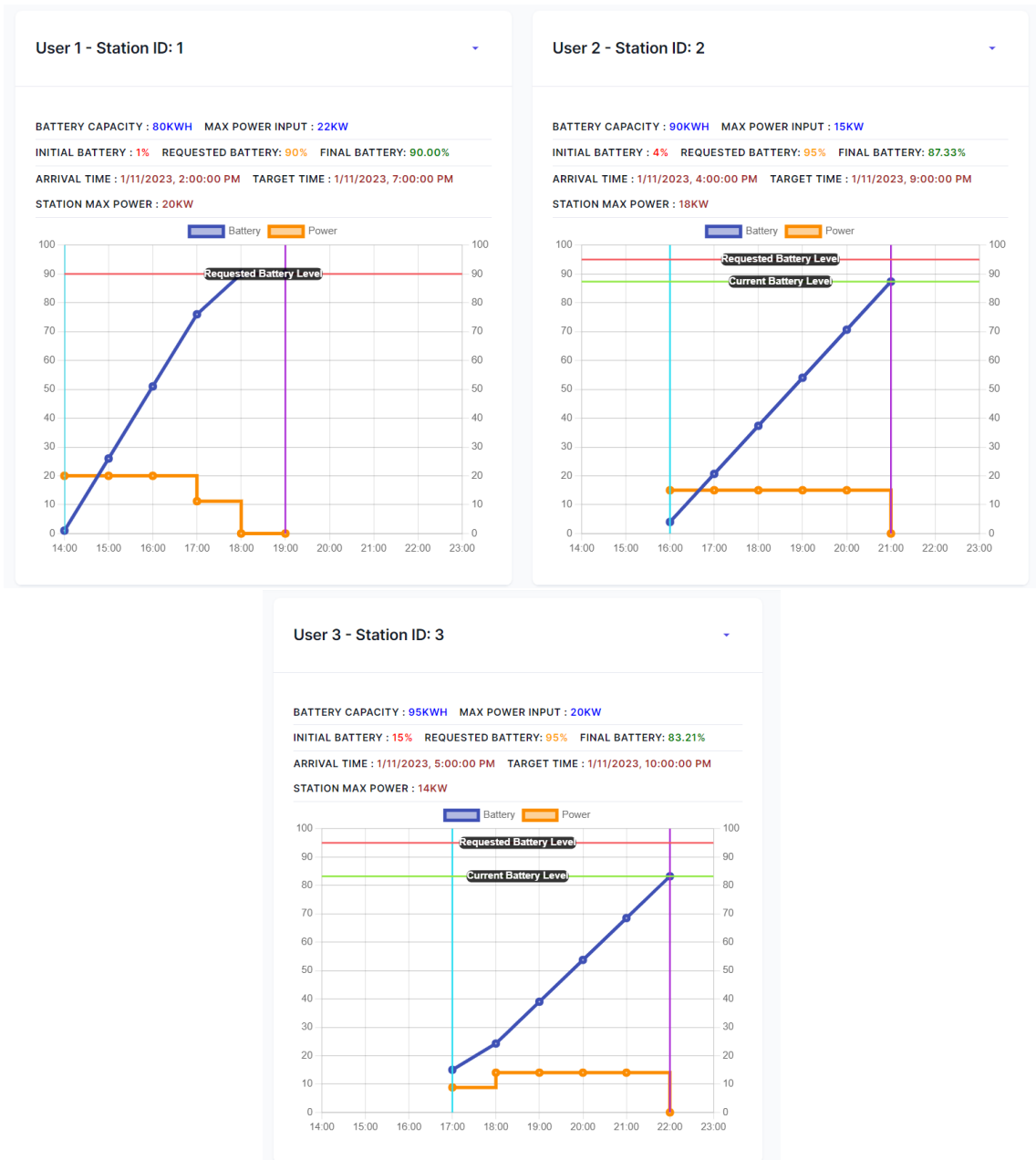


Figure 3.10. Power output and battery status for User 1, User 2, and User 3 in simulation scenario 2

Power output in the scenario:

- User 1 gets the most priority and gets 20 kW power output in the first 3 epochs. In the first two epochs, there were no other cars connected to stations 2 and 3. In the 3rd epoch, User 1 gets the most priority due to having the earliest leaving time. In the 4th epoch user, 1 gets the second most priority due to the total energy requirement for User 1 being lower than User 2. User 1 leaves Station 1 at the end of epoch 5 with a final battery percentage of 90.0. User 1’s requirements are fulfilled.

- User 2 gets the highest priority from epoch 4 to epoch 7 due to leaving earlier than User 3 and thus receiving a power output of 15 kW. User 2 leaves Station 2 at the end of epoch 7 with a final battery percentage of 87.33. User 2's requirements are not fulfilled.
- User 3 leaves Station 3 at the end of epoch 8 with a final battery percentage of 83.21. User 3's requirements are not fulfilled.

3.6.3 Scenario 3

The scenario has 3 stations, 4 users, and 1 Intelligent Controller. The number of epochs in the simulation is 12 and the length of each epoch is 3600 seconds or 1 hour.

User and station attributes of the simulation are displayed in Table 3.6.

User ID	Connected Station ID	Car battery capacity	Car maximum power input	Station maximum power output
User 1	Station 1	120 kWh	22 kW	20 kW
User 2	Station 2	90 kWh	12 kW	15 kW
User 3	Station 3	100 kWh	25 kW	10 kW
User 4	Station 3	100 kWh	25 kW	10 kW

Table 3.6. User and station attributes of Simulation Scenario 3

The simulation starts at 2 pm. The total maximum power output for all stations is 30.0 kW.

Users' requirements in the scenario:

- User 1 arrives at the start of epoch 1 at Station 1 and requests a target state of charge of 80% from the initial charge of 20%. The target time for User 1 is 2 hours later at 4 pm.
- User 2 arrives at the start of epoch 1 at Station 2 and requests a state of charge of 70% from the initial charge of 5%. The target time for User 2 is 12 hours later at 2 am the next day.
- User 3 arrives at Station 3 at the start of epoch 6 at 7 pm and requests a state of charge of 95% from the initial charge of 15%. The target time for User 3 is 5 hours later at 12 am.
- User 4 arrives at the start of epoch 1 at Station 3 and requests a state of charge of 95% from the initial charge of 15%. The target and leaving time for User 4 is 6 hours later at 6 pm.

The power output and battery percentage for the simulation are illustrated in Figure 3.11 for User 1, User 2, User 3, and User 4.



Figure 3.11. Power output and battery status for User 1, User 2, User 3 and User 4 in simulation scenario 3

According to the greedy algorithm, the earliest leaving time gets the most priority in this case.

Power output in the scenario:

- User 1 gets the most priority and gets 20 kW power output in the first 2 epochs. User 1's requirements are not fulfilled even after providing the maximum power output for both epochs. User 1 leaves the station at the end of epoch 2 with a final battery percentage of 53.33. User 1's charging requirements are not fulfilled.
- User 2 does not receive any power for the first 2 epochs as the total maximum power of 30 kW is used by the other 2 stations. After User 1 leaves at the end of epoch 2, User 2 starts receiving power from Station 2. The charging requirements for User 2 are fulfilled in epoch 7 and User 2's car reaches the target battery percentage of 70.0. User 2's charging requirements are fulfilled.
- User 3 receives 10 kW of power per epoch from epoch 6 to epoch 10. User 3 leaves Station 3 at the end of epoch 10 with a final battery percentage of 65.0. User 3's charging requirements are not fulfilled.
- User 4 gets the second most priority and receives 10 kW for the first 4 epochs. User 4 leaves Station 3 at the end of epoch 4 with a final battery percentage of 55.0. User 4's charging requirements are not fulfilled.

All simulation scenarios that were planned to be simulated using the platform, were successfully simulated.

4. RESULTS AND FINDINGS

As specified in section 3.5, the developers had no prior knowledge of the SimCES platform and architecture. However, the documentation of SimCES proved to be beneficial to have a good understanding of developing a new component and creating a simulation. Although SimCES is well documented, the content is vast and can be overwhelming. Developers may often encounter challenges when trying to identify appropriate elements and determining the starting point for developing a simulation. However, the documentation contains a link to an example component in GitHub named 'Simple component' that contains additional detailed instructions to create a new component using the Python toolkit in steps [56]. The utilization of the example component proved to be advantageous for developers during the process of creating simulation components.

The available Python toolkit was particularly useful as it provided reusable software modules that reduced redundant development work. The addition of Log Reader component allows users to check the logs of the simulation using a web browser which is convenient. It was possible to implement a greedy algorithm in one of the components and accurately calculate the power output of the electric vehicle charging simulation.

The implementation of loose-coupling of components was beneficial in the development of individual components. Developers were able to develop each component independently without relying on other components. This was particularly useful while creating the User and the Station components independently without the need for creating the Intelligent Controller component. Abstraction and autonomy are beneficial in terms of not exposing information about the implementation and only focusing on the messages for interaction. For the components, the only information exchanged during the interaction between the components is the component messages. This is beneficial for the developers of a component as they do not require knowledge of the implementation of the other components they are interacting with. The components developed in the research are reusable in different scenarios as observed from the simulations developed in this research. The components can be reused for future simulations as well.

No issues were observed while simulating different scenarios. The simulations ran successfully, and the output results were as expected. The greedy algorithm was implemented in the Intelligent Controller component and the component was able to success-

fully calculate and publish accurate power outputs for the stations in each of the simulation scenarios.

The usage of Docker containers proved to be beneficial for the development of the simulation as well. The simulation alongside all the components in the simulation can be started with a single Docker Compose file which is very convenient. The utilization of Docker greatly benefited the deployment of the simulation to servers as well.

An issue was specified in section 3.5 about checking logs of the components. It was observed that, to check the logs of the component, an existing shell script needs to be run to copy the logs to log files for each component. The Docker containers are auto-deleted after the simulation has exited so there is no way to check the container logs as well. Running the shell script and copying the logs each time after a simulation run is time-consuming. Additionally, the logs only provide information about the container itself. In case of any issues or errors arising from Docker or the Docker containers, there is no means to access the relevant information. The information that is available to the developers is the logs that are generated only after the container is started. Any issue regarding the startup of the container remains unknown. Additionally, since the containers are deleted immediately when the simulation ends or stops, it is not possible to access the logs from the container or Docker. This makes debugging difficult for the developers. This issue caused a significant delay in the development of the initial components.

An updated version of the Platform Manager component of SimCES was developed during the development of the simulation [57]. This updated version does not automatically delete the Docker containers after the simulation has ended. This allows the developers to check the logs from the Docker containers. However, this creates another issue of manual cleanup after running a simulation. The Docker containers now need to be manually removed and can be a hassle if there are a lot of containers in the simulation.

Another issue was discovered during the development of the simulation that concerns debugging the simulation. The aforementioned issue arises when a component is in a state of waiting, awaiting a component message, or when it becomes stuck at a particular point within the simulation due to an error or other technical issue. To find the underlying reason and determine the specific component responsible for the encountered issue, the developer needs to check every single component's log to identify the component and epoch at which the simulation has become stuck. This is time-consuming and caused significant delays in the development of the simulation.

The REST API provided by the Log Reader component is user-friendly and integrating it with the GUI Monitor was a straightforward process. It was observed that extending the simulation with external applications, such as the GUI Monitor and the Simulation Starter, using the REST API was a straightforward process. The documentation for the REST API of the Log Reader component is comprehensive and provides clear guidance on its

usage.

SimCES simulations necessitate setting all attributes and input variables at the beginning of the simulation. The simulation starts with all the attributes and input variables and runs calculations in each epoch till the end of the simulation. Currently, there is no functionality for the User components in the simulation to interact with each other or update charging requirements during the runtime of the simulation in SimCES that replicate more real-world scenarios. The requirements of such scenarios cannot be fulfilled with the current version of the simulation and the SimCES platform. Additionally, in the course of discussing the simulation of real-world scenarios using the platform, it has been observed that replicating real-world scenarios can pose significant challenges as real-world scenarios encompass a multitude of variables. Replicating these complex scenarios in a simulation environment can prove to be exceedingly difficult and replicating some scenarios may not even be possible due to the limitations of a simulation environment. Future work on this domain is discussed further in Chapter 5.

5. DISCUSSION AND FUTURE WORK

The results provide insight into the capability in developing simulations of electric vehicle charging, as well as highlighting the limitations and issues during the development of the simulation with SimCES platform. As specified in Chapter 1, the first research question focuses on the limitations and challenges of SimCES platform that were discovered during the development of the EV charging simulation. The study was able to provide a response to this research question as issues were discovered regarding logging and debugging that caused significant delays in development of the simulation. The second research question specified in Chapter 1 focuses on the suitability of the platform regarding EV charging simulations. It was discovered from the results in Chapter 4 that SimCES platform is suitable for developing simple EV charging simulations that have static requirements for charging that is provided to the platform before the simulation is started. However, in terms of real-world EV charging scenarios that are more complex, the current version of SimCES platform is not suitable for fulfilling the requirements of those scenarios. To replicate a real-world EV charging simulation, it is required for the simulation platform to provide the users the ability to interact, update charging requirements, and provide inputs during the run time of the simulation. This is one drawback that does not allow complex simulations that replicate real-world scenarios to be developed using the current version of the platform. The research showcases the benefits of developing an EV charging simulation with SimCES. Furthermore, the findings demonstrate that incorporating external components and applications is a straightforward process, and deploying them to the server is effortless due to the implementation of Docker containers.

The research is limited in terms of addressing and simulating EV charging simulations that replicate real-world EV charging scenarios. This issue has been previously acknowledged as a potential limitation of the platform. However, one potential approach could involve creating the User component as an external component instead of a SimCES platform managed component. The component can then be designed in such a way that it can enable users to have interactive actions due to changing requirements in the simulation. Another strategy that could have been employed is to use the behavior and decisions of users as the initial parameters in the simulation. Another approach could involve implementing artificial intelligence that mimics user behavior and decision-making in response to changing charging requirements. For instance, artificial intelligence could

simulate how EV users might adjust their charging preferences based on factors such as battery state of charge, charging costs, time constraints, and other relevant variables.

The research is also limited in terms of addressing some aspects of EV charging such as spot pricing, traffic forecasting, and battery degradation that were addressed in previous studies. The idea of combining simulation tools can be beneficial as discussed in previous studies.

The research is limited in terms of combining SimCES with other simulation platforms and tools. Although external applications were developed in this study and integrated with SimCES, they were not simulation platform or tools. The implementation of combining other platforms and tools similar to previous studies [16, 20] may be beneficial to incorporate additional EV charging variables and create more complex EV charging scenarios.

The issues addressed in Chapter 4 regarding debugging can be addressed in future versions of SimCES by updating the platform and logging system to notify the developers in events of Docker or Docker container-related issues and identify the component responsible for the event. This will reduce the debugging time and effort required to check every single container log to find the source of the issue. It may be beneficial to automate the shell script to run after a simulation has ended to reduce more manual work. Similarly, for a component that is waiting for a message or is stuck at a certain point, an interface can be added to the platform that displays the components that are waiting for messages to be processed or have faced some kind of error. Although this does not completely solve the problem as the developers still need to manually check the component logs for the issue, it will certainly reduce the area of search for debugging the issue. Further research can be conducted to resolve this issue.

Future versions of SimCES platform can be modified to allow complex simulations to be developed by the platform. Further research can be conducted to explore potential solutions for simulating user interaction in terms of EV charging simulations. As discussed previously in Chapter 4 replicating real-world EV charging solution can be challenging, additional research in this area could be undertaken to conduct a more comprehensive investigation into the challenges associated with replicating real-world scenarios in a simulation environment, particularly in the context of EV charging.

The future versions of SimCES can focus on data security. As specified in subsection 2.3.4, user authentication and traffic encryption are not available for the Log Reader API which may result in security issues if sensitive information is used during a simulation.

Further enhancements can be made to the SimCES documentation, with one possible improvement being the inclusion of detailed instructions for utilizing the Python toolkit directly within the documentation page, as opposed to merely providing a link to a GitHub

repository.

Further research can be done for SimCES platform by creating non-energy-based simulations. So far, all research conducted using SimCES platform has focused on energy-based simulations. As SimCES platform claims not to be restricted to any specific application, the validity of the claim can be researched. The usage of iterative simulations in SimCES can be another topic of future research. Iterative simulations can be beneficial for real-world EV charging simulations. Further research can be conducted on the EV charging algorithm implemented in this research and discovering the optimal EV charging solution.

6. CONCLUSIONS

The research presented the capability of the Simulation Environment of Complex Energy System (SimCES) platform in simulating electric vehicle charging scenarios. SimCES platform offers a microservice-based simulation environment that empowers developers to create autonomous components and develop electric vehicle charging simulations. The research involved creating an electric vehicle charging simulation using SimCES platform, followed by an evaluation of the compatibility of the platform with the electric vehicle charging domain.

The research answers the research questions regarding the capability of SimCES to simulate an algorithm-based EV charging scenario. Throughout the research, the advantages of using SimCES for simulating such scenarios were discovered. It was noted that developers who are unfamiliar with SimCES but familiar with Docker and Python, can easily adapt to developing simulation components using the platform, aided by the available documentation and the development toolkit.

Even though the comprehensive documentation facilitated the development of the simulation, certain challenges were identified. The study investigates and identifies the issues and drawbacks of the platform related to logging and debugging during the development of a component using SimCES. Furthermore, the study provides recommendations for potential enhancements. Enhancements to the existing logging and debugging capabilities of the platform would enable future developers to create simulations with greater efficiency.

The research highlights the limitations of the platform in accurately replicating real-world scenarios, particularly in the context of simulating EV charging. Understanding and addressing these limitations are important areas for further research and improvement in the simulation of EV charging scenarios.

In addition, the research explores potential avenues for future investigation and research. The research identifies several promising areas for further research, including enhancing the platform itself, optimizing debugging techniques, investigating challenges related to replicating real-world scenarios in a simulation environment and exploring optimal EV charging solutions. These areas offer significant opportunities for future investigations and advancements in the field.

REFERENCES

- [1] J. Banks. "Introduction to simulation". In: *2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165)*. Vol. 1. 2000, 9–16 vol.1. DOI: 10.1109/WSC.2000.899690.
- [2] A. Byrne. "What is simulation for?" In: *Anaesthesia* 67.3 (2012), pp. 219–225. DOI: <https://doi.org/10.1111/j.1365-2044.2011.07053.x>. eprint: <https://associationofanaesthetists-publications.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2044.2011.07053.x>. URL: <https://associationofanaesthetists-publications.onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2044.2011.07053.x>.
- [3] Petri Kannisto, Otto Hylli, Ville Heikkilä, Antti Supponen, Timo Aaltonen, Sami Repo, Kari Systä, Antti Keski-Koukkari, Amir Safdarian, and Anna Kulmala. "Software and Communications Platform for Simulation Environment of Complex Energy System (SimCES)". eng. In: Tampere University. Institute of Electrical and Electronics Engineers Inc, 2021-06-28.
- [4] Petri Kannisto, Ville Heikkilä, Otto Hylli, Mehdi Attar, Sami Repo, and Kari Systä. "SimCES platform for modular simulation: Featuring platform independence, container ecosystem, and development toolkit". eng. In: *SoftwareX* 19 (2022), pp. 101189–. ISSN: 2352-7110.
- [5] Robert Shannon. "Introduction to simulation". eng. In: *Proceedings - Winter Simulation Conference*. WSC '92. ACM, 1992, pp. 65–73. ISBN: 0780307984.
- [6] Andrew Seila. "Introduction to simulation". eng. In: *Proceedings of the 27th conference on winter simulation*. WSC '95. IEEE Computer Society, 1995, pp. 7–15. ISBN: 0780330188.
- [7] Ricki Ingalls. "Introduction to simulation". eng. In: *2013 Winter Simulations Conference (WSC)*. WSC '13. IEEE Press, 2013, pp. 291–305. ISBN: 9781479920778.
- [8] Cláudio Gomes, Casper Thule, David Broman, Peter Larsen, and Hans Vangheluwe. "Co-Simulation: A Survey". eng. In: *ACM computing surveys* 51.3 (2018), pp. 1–33. ISSN: 0360-0300.
- [9] Irene Hafner and Niki Popper. "On the terminology and structuring of Co-simulation methods". eng. In: *ACM International Conference Proceeding Series*. 2017, pp. 67–76. ISBN: 9781450363730.
- [10] Sebastian Lehnhoff, Okko Nannen, Sebastian Rohjans, Florian Schlogl, Stefan Dalhues, Lena Robitzky, Ulf Hager, and Christian Rehtanz. "Exchangeability of power flow simulators in smart grid co-simulations with mosaik". eng. In: *2015 Workshop*

- on Modeling and Simulation of Cyber-Physical Energy Systems, MSCPES 2015 - Held as Part of CPS Week, Proceedings*. 2015. ISBN: 9781479973583.
- [11] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. “Co-simulation: State of the art”. eng. In: *arXiv.org* (2017). ISSN: 2331-8422.
- [12] D.P. Chassin, K. Schneider, and C. Gerkenmeyer. “GridLAB-D: An open-source power systems modeling and simulation environment”. eng. In: *2008 IEEE/PES Transmission and Distribution Conference and Exposition*. IEEE, 2008, pp. 1–5. ISBN: 9781424419036.
- [13] J. A. Martinez, V. Dinavahi, M. H. Nehrir, and X. Guillaud. “Tools for Analysis and Design of Distributed Resources-Part IV: Future Trends”. eng. In: *IEEE transactions on power delivery* 26.3 (2011), pp. 1671–1680. ISSN: 0885-8977.
- [14] Zhao Xu, J. Ostergaard, and M. Togeby. “Demand as Frequency Controlled Reserve”. eng. In: *IEEE transactions on power systems* 26.3 (2011), pp. 1062–1071. ISSN: 0885-8950.
- [15] Peter Fritzon, Adrian Pop, Karim Abdelhak, Adeel Asghar, Bernhard Bachmann, Willi Braun, Daniel Bouskela, Robert Braun, Lena Buffoni, Francesco Casella, Rodrigo Castro, Rüdiger Franke, Dag Fritzon, Mahder Gebremedhin, Andreas Heuermann, Bernt Lie, Alachew Mengist, Lars Mikelsons, Kannan Moudgalya, Lennart Ochel, Arunkumar Palanisamy, Vitalij Ruge, Wladimir Schamai, Martin Sjolund, Bernhard Thiele, John Tinnerholm, and Per Ostlund. “The OpenModelica integrated environment for modeling, simulation, and model-based development”. eng. In: *Modeling, identification and control* 41.4 (2020), pp. 241–285. ISSN: 0332-7353.
- [16] Peter Palensky, Edmund Widl, Matthias Stifter, and Atiyah Elsheikh. “Modeling Intelligent Energy Systems: Co-Simulation Platform for Validating Flexible-Demand EV Charging Management”. eng. In: *IEEE transactions on smart grid* 4.4 (2013), pp. 1939–1947. ISSN: 1949-3053.
- [17] Eric Sortomme and Mohamed A El-Sharkawi. “Optimal Charging Strategies for Unidirectional Vehicle-to-Grid”. eng. In: *IEEE transactions on smart grid* 2.1 (2011), pp. 131–138. ISSN: 1949-3053.
- [18] *MATLAB*. <https://se.mathworks.com/products/matlab.html>. Accessed: 24 April 2023.
- [19] *CVX*. <http://cvxr.com/cvx/>. Accessed: 24 April 2023.
- [20] T Bellemans, B Kochan, D Janssens, G Wets, Theo Arentze, and Harry Timmermans. “Implementation framework and development trajectory of FEATHERS activity-based simulation platform”. eng. In: *Transportation research record* 2175.2175 (2010), pp. 111–119. ISSN: 0361-1981.
- [21] L. Knapen, B. Kochan, T. Bellemans, D. Janssens, and G. Wets. “Activity based models for countrywide electric vehicle power demand calculation”. eng. In: *2011 IEEE 1st International Workshop on Smart Grid Modeling and Simulation, SGMS 2011*. IEEE, 2011, pp. 13–18. ISBN: 1467301949.

- [22] O. Sundstrom and C. Binding. “Flexible Charging Optimization for Electric Vehicles Considering Distribution Grid Constraints”. eng. In: *IEEE transactions on smart grid* 3.1 (2012), pp. 26–37. ISSN: 1949-3053.
- [23] Yue Wang, David Infield, and Simon Gill. “Smart charging for electric vehicles to minimise charging cost”. eng. In: *Proceedings of the Institution of Mechanical Engineers. Part A, Journal of power and energy* 231.6 (2017), pp. 526–534. ISSN: 0957-6509.
- [24] *OpenDSS*. <https://www.epri.com/pages/sa/opensdss>. Accessed: 25 April 2023.
- [25] Michele Rondinone, Julen Maneros, Daniel Krajzewicz, Ramon Bauza, Pasquale Cataldi, Fatma Hrizi, Javier Gozalvez, Vineet Kumar, Matthias Röckl, Lan Lin, Oscar Lazaro, Jérémie Leguay, Jérôme Härrri, Sendoa Vaz, Yoann Lopez, Miguel Sepulcre, Michelle Wetterwald, Robbin Blokpoel, and Fabio Cartolano. “iTETRIS: A modular simulation platform for the large scale evaluation of cooperative ITS applications”. eng. In: *Simulation modelling practice and theory* 34 (2013), pp. 99–125. ISSN: 1569-190X.
- [26] Gerwin Hoogsteen, Johann L. Hurink, and Gerard J. M. Smit. “DEMKit: a Decentralized Energy Management Simulation and Demonstration Toolkit”. eng. In: *2019 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)*. IEEE, 2019, pp. 1–5. ISBN: 1538682184.
- [27] Fabiano Pallonetto, Eleni Mangina, Federico Milano, and Donal P. Finn. “SimApi, a smartgrid co-simulation software platform for benchmarking building control algorithms”. eng. In: *SoftwareX* 9 (2019), pp. 271–281. ISSN: 2352-7110.
- [28] Cody J. Permann, Derek R. Gaston, David Andrš, Robert W. Carlsen, Fande Kong, Alexander D. Lindsay, Jason M. Miller, John W. Peterson, Andrew E. Slaughter, Roy H. Stogner, and Richard C. Martineau. “MOOSE: Enabling massively parallel multiphysics simulation”. eng. In: *SoftwareX* 11.C (2020), pp. 100430–. ISSN: 2352-7110.
- [29] Lili Zhao, Ming Ni, Heqin Tong, and Yuecen Li. “Design and application of distributed co-simulation platform for cyber physical power system based on the concepts of software bus and middleware”. eng. In: *IET Cyber-Physical Systems* 5.1 (2020), pp. 71–79. ISSN: 2398-3396.
- [30] Connor R. Jakubik, Adam Johnston, Patrick Zhong, Neil McHenry, and Gregory Chamitoff. “Spacecraft vr: An event-driven, modular simulation platform with fully-asynchronous physics”. eng. In: *AIAA Scitech 2021 Forum*. 2021, pp. 1–16. ISBN: 9781624106095.
- [31] Juha Kiviluoma, Fabiano Pallonetto, Manuel Marin, Pekka T. Savolainen, Antti Soininen, Per Vennström, Erkka Rinne, Jiangyi Huang, Iasonas Kouveliotis-Lysikatos, Maren Ihlemann, Erik Delarue, Ciara O’Dwyer, Terence O’Donnel, Mikael Amelin, Lennart Söder, and Joseph Dillon. “Spine Toolbox: A flexible open-source workflow

- management system with scenario and data management”. eng. In: *SoftwareX* 17 (2022), pp. 100967–. ISSN: 2352-7110.
- [32] Petri Kannisto, David Hästbacka, Teresa Gutiérrez, Olli Suominen, Matti Vilkkö, and Peter Craamer. “Plant-wide interoperability and decoupled, data-driven process control with message bus communication”. eng. In: *Journal of industrial information integration* 26 (2022), pp. 100253–. ISSN: 2452-414X.
- [33] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. eng. Hoboken: Pearson Education, Limited, 2005. ISBN: 0131858580.
- [34] Marcus Hilbrich and Fabian Lehmann. “Discussing Microservices: Definitions, Pitfalls, and their Relations”. eng. In: *2022 IEEE International Conference on Services Computing (SCC)*. Piscataway: IEEE, 2022, pp. 39–44. ISBN: 1665481463.
- [35] *RabbitMQ*. <https://www.rabbitmq.com/>. Accessed: 24 March 2023.
- [36] *SimCES Time and synchronization with epochs*. https://simcesplatform.github.io/core_time/. Accessed: 18 April 2023.
- [37] *SimCES Workflow of start and end*. https://simcesplatform.github.io/core_workflow-start-end/. Accessed: 24 March 2023.
- [38] *SimCES Workflow of component in simulation*. https://simcesplatform.github.io/core_workflow-sim/. Accessed: 24 March 2023.
- [39] *SimCES Simple Component manifest*. https://github.com/simcesplatform/Simple-Component/blob/master/component_manifest.yml. Accessed: 18 April 2023.
- [40] *SimCES Simulation Tools*. https://simcesplatform.github.io/core_sim-tools/. Accessed: 8 April 2023.
- [41] *SimCES Simulation Tools EpochMessage*. <https://github.com/simcesplatform/simulation-tools/blob/master/tools/message/epoch.py>. Accessed: 25 April 2023.
- [42] *SimCES Simulation Tools, Message Classes*. <https://github.com/simcesplatform/simulation-tools>. Accessed: 18 April 2023.
- [43] *Docker*. <https://www.docker.com/>. Accessed: 25 April 2023.
- [44] *Docker Compose*. <https://docs.docker.com/compose/>. Accessed: 25 April 2023.
- [45] Dieter Jungnickel. *Graphs, Networks and Algorithms*. eng. Vol. 5. Algorithms and computation in mathematics. Berlin/Heidelberg: Springer, 2012. ISBN: 3642436641.
- [46] *MongoDB*. <https://www.mongodb.com/>. Accessed: 8 March 2023.
- [47] *Mongo Express*. <https://github.com/mongo-express/mongo-express>. Accessed: 25 April 2023.
- [48] *Node.js*. <https://nodejs.org/en/>. Accessed: 8 March 2023.
- [49] *React*. <https://reactjs.org/>. Accessed: 8 March 2023.
- [50] *Chart.js*. <https://www.chartjs.org/>. Accessed: 8 March 2023.
- [51] Victor Szalvay. “An introduction to agile software development”. In: *Danube technologies* 3 (2004).

- [52] Muhammad Ovais Ahmad, Denis Dennehy, Kieran Conboy, and Markku Oivo. “Kanban in software engineering: A systematic mapping study”. eng. In: *The Journal of systems and software* 137 (2018), pp. 96–113. ISSN: 0164-1212.
- [53] *Trello*. <https://trello.com/>. Accessed: 17 March 2023.
- [54] James Bach. “Exploratory testing explained”. In: *Online: http://www.satisfice.com/articles/et-article.pdf* (2003).
- [55] *GitHub Actions*. <https://docs.github.com/en/actions>. Accessed: 9 April 2023.
- [56] *SimCES Simple Component*. <https://github.com/simcesplatform/simple-component>. Accessed: 8 April 2023.
- [57] *SimCES Platform Manager Docker images*. <https://github.com/simcesplatform/Platform-Manager/pkg/container/platform-manager>. Accessed: 8 April 2023.