Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Modeling and Simulation of Situations in Mobile Cyber-Physical Systems

Fahmida Israt

Course of Study:	Computer Science
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisor:	M.Sc. Kálmán Képes
Commenced:	Aug 15, 2018
Completed:	Feb 15, 2019

Abstract

People aim to build completely smart environments in the near future. Building a smart environment is possible through integrating the physical and computing systems which is known as Cyber-Physical System. In the era of mobile devices, Mobile Cyber-Physical System is more required and feasible for daily life. Mobile Cyber-Physical Systems can become automatic systems by enabling Situation-Aware execution in management processes. Context/Situation-Awareness in Mobile Cyber-Physical System is practicable if the individual situation of a system can be modeled and simulated at development time. For this, abstract models or architectures that show how situations can be modeled and simulated in Mobile Cyber-Physical Systems are necessary. As Mobile Cyber-Physical Systems depend on multiple disciplines and is a complex combination of sensors, application systems, embedded systems, and users, it is complicated to model situations for these systems. Not only are these systems complex in integration, but they can also be huge in size too. Therefore, while modeling situations, it is essential to consider all objects of the system very carefully. This challenge starts by knowing and monitoring all the sensors to obtain sensor data. Then situations need to be modeled and later detected through simulation based on the sensor data. In this thesis, we propose an abstract system and build a simulation process that helps to model and simulate situations to ease the development and testing of such applications in Mobile Cyber-Physical System. Firstly, we propose a Simulation Framework for Situation-Aware Applications which consists out of two components named Simulation System and Situation Recognition System. Then we have implemented our approach within the scenario of the application of Software Update in vehicular systems.

Contents

List of Figures

List of Tables

Listings

List of Algorithms

List of Abbreviations

CPS Cyber-Physical System. 7

IoT Internet of Things. 7

MCPS Mobile Cyber-Physical System. 17

OSM OpenStreetMap. 63

RSU Road-Side Unit. 13

SitAC A System for Situation-aware Access Control. 36

SitME Situation-Aware Workflow Modeling Extension. 35

SUMO Simulation of Urban MObility. 7

XML Extensible Markup Language. 33

1 Introduction

Computing and communication capabilities are getting embedded in the physical environment day by day [RLSS10]. In near future, all objects and structure of physical environment will have the computing capabilities. This is how the term Cyber-Physical System (CPS), Mobile Cyber-Physical System (MCPS), and Internet of Things (IoT) originated. Figure 1.1 shows that with the help of IoT, MCPS integrates the physical things and computing power of them together to provide an interconnected system. With the aid of the computing power in physical things it can get easier to monitor, control, and interpret the physical activities of a system [SH15].

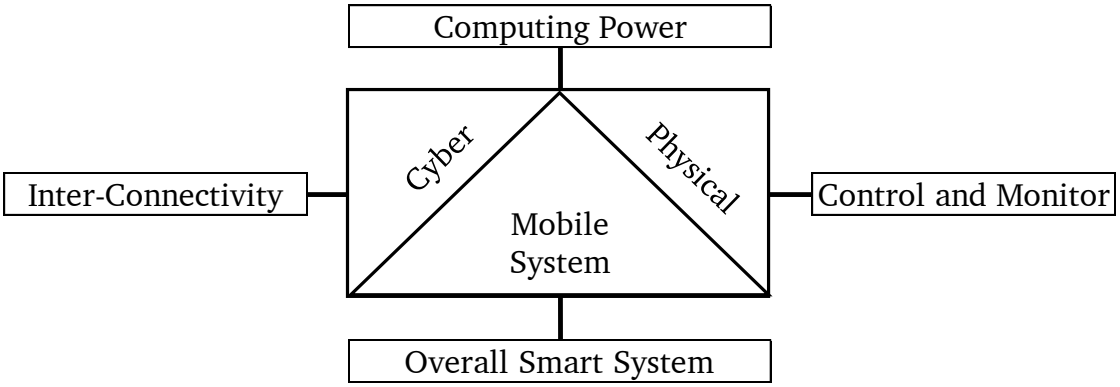

Figure 1.1: Mobile-Cyber Physical System (MCPS) Integration with many Aspects

There are already many software approaches available (e.g. Smart Greenhouse Remote Monitoring Systems, Retail In-Store Analytics, Smart Waste Management Systems, etc.)[1] that aim to make connected ecosystems for different communities. To make smart companies, there are CPS available for interconnected production processes, furthermore, people now are aiming to build smart cities with autonomous vehicles and smart roads system.
To make smart environment, future systems must cope up with a plethora of different hardware and software systems for applications from modern fields like MCPS. The applications of these systems are highly dynamic. They are dynamic in nature because of the continuous changing network and environment. Due to this reason, the management of these MCPSs are challenging and is prone to failure. However, these systems must still be able to cope with any kind of application under any kind of situation. Therefore, a

[1] https://www.postscapes.com/categories/

Context- and Situation-Aware Application Management is needed that is able to execute management logic in the right situation. For example, the management of software update, connection management between peers for vehicles.

1.1 Problem Domain and Motivation

Running a Situation-Aware CPS is not easy because modeling and recognizing situations of such a system is not trivial. The size of the MCPSs make the development of such Situation-Aware Applications a challenge, as different hardware and software platforms are used in a highly dynamic and large environment. The modeling, execution, and hence, the recognition of situations in such environments is a complex challenge and must be developed with care. But to progress with building smart environments, it is important that the system can recognize situations and act according to it. The situation recognition for dynamic systems is a very complex and difficult process [HWS+16]. It first starts with modeling as many scenarios as possible. Based on these scenarios, the first level of the work is to acquire sensor data from all devices of the system. Sensors raw data can be turned into context information by adding metadata to them. Sensor data later is combined and interpreted to derive properly understandable situations based on this context information. This is how sensor data lead to knowledge about a smart environment [VF13]. Once the situations are derived, the system can be a self-organizing system based on the modeled and recognized situations.

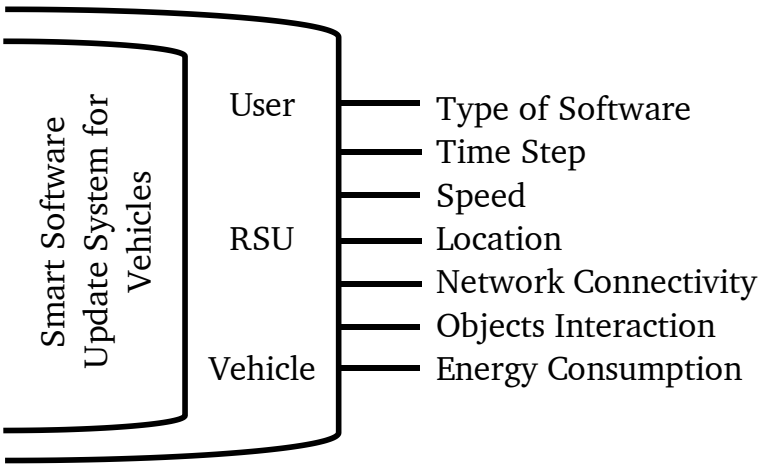

Figure 1.2: Objects (e.g., time step, speed, location) to Consider while Building a Smart Software Update System for Vehicles

Let us consider that a Situation-Aware workflow for smart vehicular environment can be modeled. Think of a smart workflow system for vehicle's software update. The situation modeling and simulation for such system is definitely sensitive as human safety is related to this. A lot of factors need to be taken care of while taking the decision (See Figure 1.2).

Additionally, not only the vehicles and users are related to this system, other things like Road-Side Unit (RSU) also need to be considered. Modeling situations for this smart software update system for vehicles needs to reconsider objects like, which software needs to be updated, time step, vehicle's speed, location, network connectivity, user interaction, energy consumption data, remaining energy, and so on.
Therefore this thesis proposes a Simulation Framework for Situation-Aware Applications to enable the development of such applications in MCPS. We need a simulator for the development of such systems, as it is complex, error-prone and basically not achievable at run-time in the real world.

1.2 Scope of Work

The goal of this thesis is to do modeling and simulation of situations in MCPS. Therefore, it is necessary to look at the related topics like CPS, IoT, modeling and simulation methods, context, situation and workflow of Situation-Aware Systems. There are established research work on situation recognition, modeling CPS and Situation-Aware MCPSs that help to build the concept and implementation of this work. The main contribution of this thesis is presenting a framework of modeling situations for MCPS including the requirements, relationships among the objects of the model. Based on the framework, a vehicular simulator is selected as we choose vehicular environment as the Mobile CPS.
The feasibility of this concept is also shown with the help of prototypical implementation developed in the selected simulator. Additionally, the modeled abstract situations are recognized through prototypical python implementation.

1.3 Thesis Outline

This thesis structured as follows:

Chapter 1 – Introduction: The motivation of this thesis including basic introduction and fundamentals of CPS and situation modeling are explained in this chapter.

Chapter 2 – Fundamentals: Some related important technologies, terms and theories are explained for further better understanding.

Chapter 3 – Related Work: Some related research work to modeling and simulation of situations in MCPS are presented.

Chapter 4 – Concept and Modeling: The high-level concepts behind the thesis are explained. The overview of the thesis's model, requirements, analysis of the model and situations modeling for MCPS are discussed.

Chapter 5 – Simulator Selection: Based on the concept, the simulator selection criteria and a brief description on the selected simulator SUMO is presented.

Chapter 6 – Simulator Environment and Prototypical Implementation: The prototypical implementation of this thesis is discussed in this Chapter including the preparation for simulator environment and recognition of the modeled situations.

Chapter 7 – Conclusion and Future Work: A summary of the this thesis and scope for further research work are discussed.

2 Fundamentals

In this chapter, we discuss different technologies, terms and theories in details that are necessary for a better understanding of this thesis. The first Section deals with the definition of Cyber-Physical System (CPS), MCPS, their features, generic architecture and example of vehicular MCPS. In Section 2.2, we take a look at Internet of Things (IoT) paradigm and how IoT and CPS are interconnected. Then in Section 2.3, we discuss what context and situation are. Followed by that, there is a brief overview of Situation-aware Systems and Workflow Technology in Section 2.4. Finally, in Section 2.5, we explain what generally simulation and modeling are.

2.1 Cyber-Physical System (CPS)

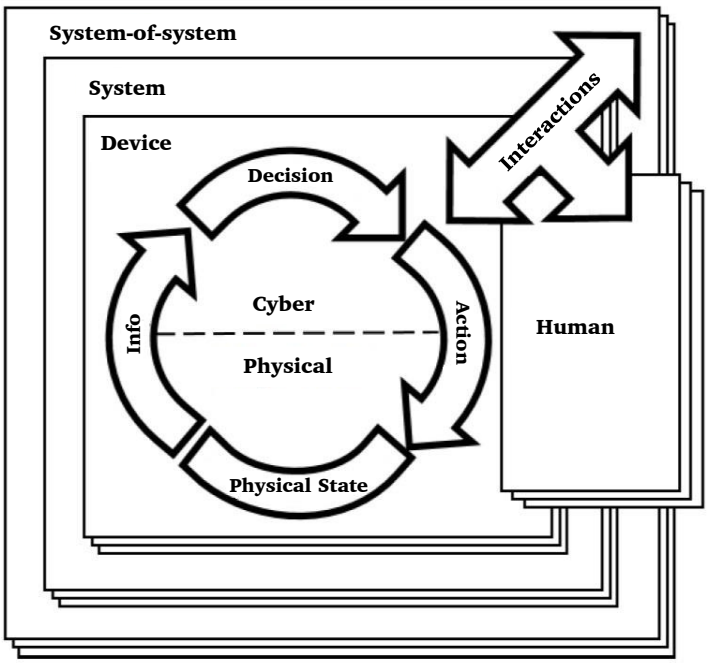

Figure 2.1: CPS Conceptual Model [Cyb16]

CPS is getting increasing attention on the market and in research. CPS is simply the integration of physical processes and computation which is the cyber part in this context [HPK12]. The goal of CPS is to combine the dynamics of physical processes with software

and communication which can be monitored, controlled and coordinated (See Figure 2.1) [SWYS11] [RLSS10]. This technology requires new abstraction, modeling, analysis and design as it relies on computers, complex embedded systems, networking and communication among sensors and actuators [SWYS11]. CPS is already used in many fields such as aerospace, transportation, health care, factory automation and defense system [AKK13].

Mobile Cyber–Physical Systems (MCPS)s use mobile sensors and computing devices using augmented human interactive communication networks with the physical world [CHHK18]. CPS uses stationary and big sensors or machines to communicate to the physical world but as MCPS uses mobile sensors, it is all about mobility. Due to this characteristic of it, MCPS is available easily in day to day life and deployed into the broad range [GHH+17]. Figure 2.2 shows how CPS and MCPS are similar or different.

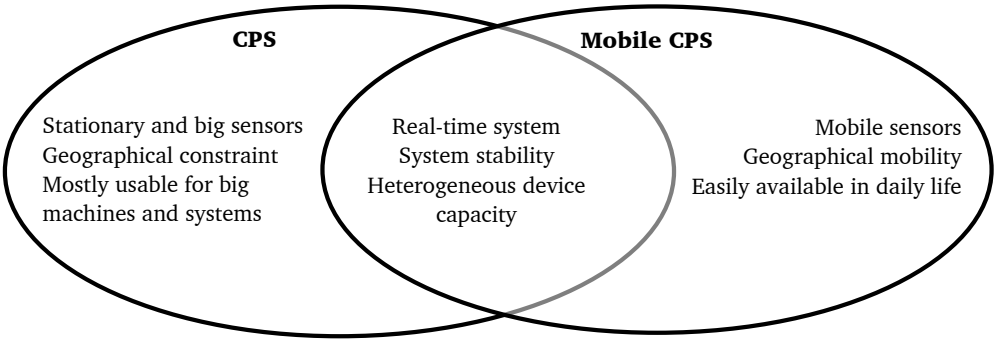

Figure 2.2: Relation between CPS and MCPS [GHH+17]

2.1.1 Features and Characters of CPS and MCPS

Some common features of CPS and MCPS as follows:

- Physical system is the most important field of CPS with a requirement of a high degree of automation and network characteristics [LPW+17] and all physical components have to have cyber capabilities [SM12].
- As CPSs are the integration of physical and computational process, everything must be closely integrated [SWYS11]. Moreover, the integration of the whole architecture must be consistent to capture all necessary physical information [RLSS10].
- Operational and managerial independence of the components in the overall system is very necessary. At the same time, there should also be distributed control [OTV17].
- In a CPS or an MCPS system, different categories of devices need to appear and for them wired or wireless networking should be available at multiple and maximum levels [SWYS11].
- To generate a new CPS, a Model-based development is necessary to put together physical, computing and communication dynamics. Also, to test the control logic of the whole system software testing is necessary [RLSS10].

- Real-time performance is expected from CPSs. All the inputs and outputs communication among physical and computing environment should happen within the fixed minimum response time.
- The real-time communication needs to be done in secured communication channels always [SM12].
- When necessary, it should be possible to do dynamic reconfiguration or reorganization of the whole system [OTV17].

2.1.2 Generic Architecture of CPS and MCPS

CPS architecture is the basis of development and further research to cope up with the challenges. A generic and basic architecture of the CPS must contain end-user access layer, information system layer and physical system layer (See Figure 2.3) [LPW+17].

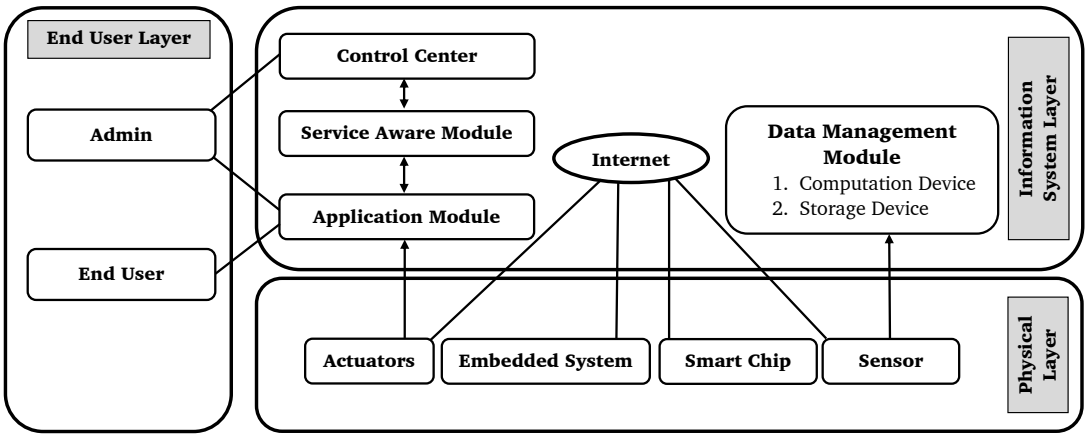

Figure 2.3: Generic Architecture of CPS and MCPS

End-user access layer is mainly responsible for making the CPS available to end users. Users can be system administrators or general end users. System admins control the system by sending inquiry instructions to the control center and also can revise the defined control strategies. All end users must be able to communicate with the system anytime, do data query in real-time and receive feedback data [LPW+17].

Information system layer does transmission and processing of data that is collected from the physical layer. It consists Data Management Module (DMM) and Service-Aware Modules (SAM). DMM has at least computational devices and storage media [AKK13]; so that DMM can collect sensor data and forward them to SAM. SAM is mainly responsible for making decisions, analyzing tasks and sending data to the available services [AKK13].

Physical system layer contains a variety number of embedded systems, sensors and actuators, sensors networks, and smart chips. Sensors collect data from the physical world. So that the responsible embedded systems or smart chips are able to achieve pre-processed preliminary data and forward to the information system layer. Actuators take requests from the application module; actually from the end users to execute the requests.

2.2 Internet of Things (IoT)

The Internet of Things (IoT) paradigm of a diverse technology is growing quickly as the integration of the smart environment is improving. The IoT is a computing concept which enables network connectivity to identify, connect, interact, and exchange data of different devices (e.g., smartphone, vehicles, camera, and home appliances, etc.) in a smart environment that contain electronics, sensors, software, and actuators [SS16].

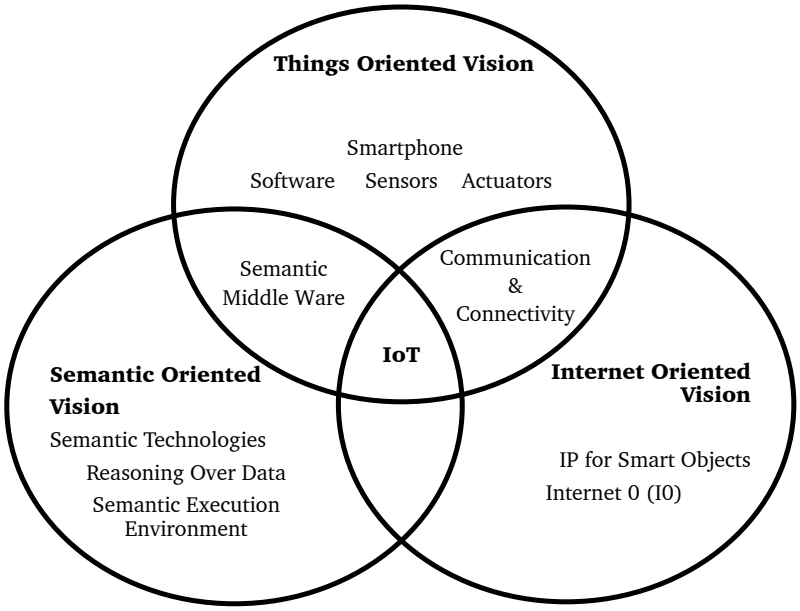

Figure 2.4: IoT paradigm according to Atzori et. al [AIM10]

According to Atzori et al. [AIM10] IoT is a united form of 3 visions: things-oriented visions (e.g., smart daily life items like smartphones, sensors, actuators, RFID[1]), internet-oriented vision (e.g., internet protocols for smart objects, Internet 0[2]) and semantic-oriented visions (e.g. semantic technologies[3], reasoning over data, semantic execution environments). Here, things and semantic oriented visions are connected through the smart semantic middleware. On the other hand, things and internet oriented visions are connected through communication and connectivity of things with the internet. See Figure 2.4 as a visualization of the description.

IoT and CPS are interconnected with each other. IoT can be seen as an enabling technology for CPS [Cyb16] for its characteristics. Some of them are:

Interconnected - By using IoT facilities, people can be connected to devices and also devices can be connected to devices [SS16].

Heterogeneity and Enormous Scale - Different hardware platforms and network can be combined together in IoT. Therefore, it is heterogeneous. The number of devices that are connected and communicated in IoT is huge. So, they also produce a big scale of data. IoT can also safely handle data because of its semantic oriented vision [PP16].

Intelligent Sensing - The IoT connected devices have smart sensing abilities. They can also have intelligence attached to them [SS16]. For example, an IoT connected bulb in a bedroom will check with its senses if there is any person in the room. If there is a person, it will be turned on. If not, then it will be automatically turned off. If the bulb has intelligence attached to it, then when the person is in the room and reading in the table, it will give brighter light. Whereas, when the person is sleeping in the bed, the bulb will behave like a dim light.

Save Energy and Increases Productivity - IoT can make a CPS very cost and energy efficient. In the previous point, we have an example of a smart sensing IoT connected bulb which gets turned off automatically when there is no person in the room. This is a small step in saving energy. As IoT can utilize resource and time optimally, it can increase productivity [SS16].

[1] Radio frequency identification (RFID) uses electromagnetic fields to automatically identify and track tags attached to objects.

[2] Internet 0 (I0) as a framework to bridge together heterogeneous devices via Internet Protocols-therefore in a manner that is compatible with designing globally large computer networks (Source: https://dspace.mit.edu/handle/1721.1/28866)

[3] Semantic Technology defines and links data on the Web (or within an enterprise) by developing languages to express rich, self-describing interrelations of data in a form that machines can process (Source: https://www.ontotext.com/knowledgehub/fundamentals/semantic-web-technology/)

Dynamic in the Real-Time - As the context of the devices and system changes frequently, the state of the devices can change dynamically in real-time. Also, the number of connected devices can change anytime based on the system's demand [PP16].

Safety - Safety of data, physical well-being can be designed and included in IoT connected CPS [PP16]. For example, in summer countries a lot of accidents happen because the tires get exploded due to overheating. These sort of accidents can be prevented if the tires have smart sensors to show the current temperature of them in the car dashboard [SS16].

Expressing - As IoT connected devices are connected with other connected devices of the environment too, any device can express the current state of the whole environment or any particular device. Therefore, it facilitates better understanding about the communication flow and state of the system between human and machines [SS16].

2.3 Context and Situation

To model and simulate situations, we have to understand first what context and situation are. For more than 20 years, researchers have defined context and situation or their synonyms in their own work. Rodden et al. [RCDD98] referred context as environment or situation. Pascoe [Pas98] defined context as the subset of physical and conceptual states of a particular entity. Abowd et al. [AM00] identified the five W's (Who, What, Where, When and Why) as the minimum information that is necessary to understand context. The definition from Abowd et al. [ADB+99] is the most accepted one and that is: "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

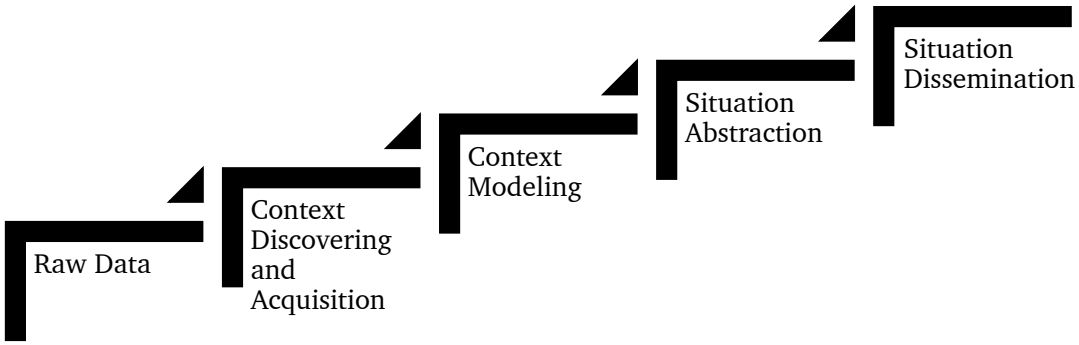

Figure 2.5: Different phases of context and situation

Few attributes are needed to define a context but the most important attributes are *context type* and *context value*. Context type defines the category of the particular context. For example, context type says if the context carries the information of speed, time, temperature or distance, etc. Whereas, context value can be the sensor's raw data, composite data or

calculated values. Suppose, the value of the temperature sensor in a car engine will say what the temperature of the engine at a particular time step is.

A context goes through different phases and they are shown in Figure 2.5. The phases explain how and why the context is retrieved and needed. At first, the context needs to be discovered and attained from all possible sources of a system [PZCG14]. In CPS, the sources are physical and virtual sensors. The lowest level of sensors production contains raw data. The difference between raw data and context is that raw data is directly retrieved unprocessed data, e.g., a float value produced by the speed checking sensors of a car. The raw data become context information through consistency checking and other processing, e.g., the speed value is associated with the running car in a highway containing the sensor. In the later phase, the context needs to go through modeling. The context modeling is required to use the context in an organized way for further requirement. After that, the situations can be abstracted from the modeled context information. Therefore, situation level stays higher than the context processing level [BHK+15]. Situation describes the states of relevant entities, e.g., the speed of a car is crossing the speed limit of the highway. At last, all low-level context information, high-level context information and situations are ready to be used by the end users or the CPS when needed. This availability and distribution are important for Situation-Aware workflow for any system.

2.4 Situation Aware System and Workflow

Context-Aware software was first introduced by Schilit et al. [SAW94] in 1994 and according to them this software can "adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time".

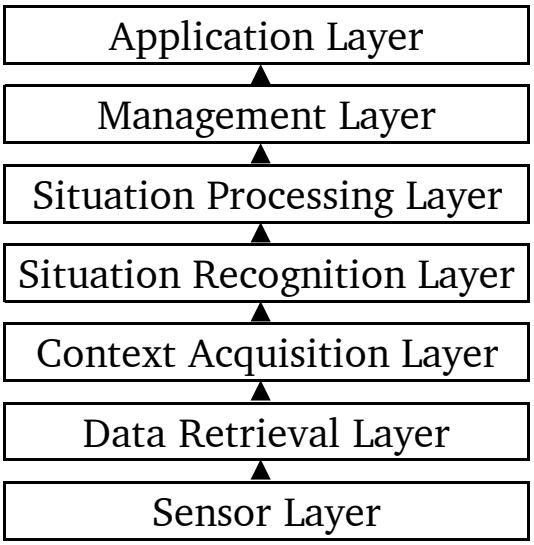

Figure 2.6: Several layers of a situation aware system

Situation-Aware Systems are a subset of Context-Aware System [HZL10]. A context- or situation-aware system can present information and execute services automatically for users. It is also possible to retrieve tagged context information later for situation modeling in such system [Dey99]. The Situation-Aware System requires expert knowledge to presume any kind of situation recognition technique and the recognition technique is reusable, adjustable and extensible. Moreover, a Situation-Aware System normally needs more expert or specific knowledge to run the system than a general Context-Aware System [HZL10]. The whole situation-aware system can have several layers like a workflow: sensors layer, data retrieval layer, context acquisition layer, situation recognition layer, situation processing and management layer and application layer (Figure 2.6).

In situation-aware system, at first, the sensors get registered at the IoT platform [MHWM17]. From the registered sensors reference ID, the context information is acquired. After that, a situation recognition algorithm or template can be modeled to do the situation recognition. A situation recognition technique specifies the ways of detecting a situation [HZL10]. The modeling of situations is created in such a way that they can be transformed into an executable implementation of situation recognition for the system. So that the system can get the situation notification and continue the pre-defined workflow. Workflows are nothing but an established step by step processing concept for a system [WSBL15]. A situation history can also be maintained by storing all the situations. Situation history later can help to do optimization of the smart environments [MHWM17]. All the steps are explained and depicted in Figure 2.7.

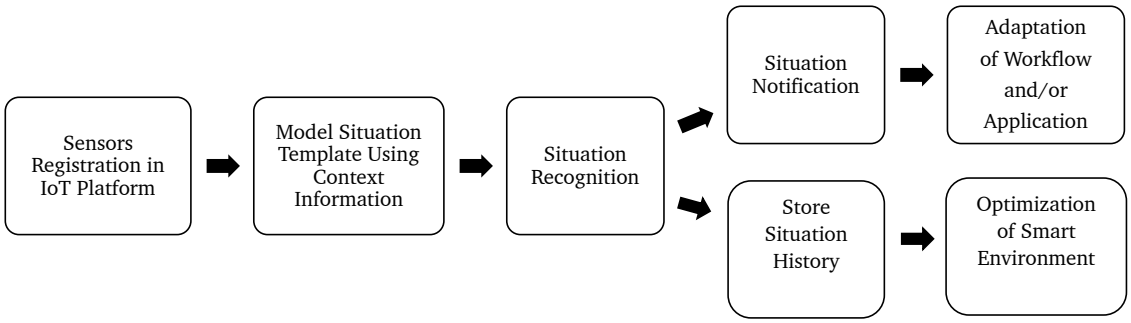

Figure 2.7: Situation recognition method using detected situations [MHWM17]

2.5 Modeling and Simulations

As the goal of this thesis is to do modeling and simulation of situations, here we discuss what actually modeling and simulation are. According to Maria [Mar97] "Modeling is the process of producing a model; a model is a representation of the construction and working of some system of interest." A modeling is close approximation to a real-time system. The modeling of any system should not be very complex and easy to understand. So that it is possible to do experiments with the modeling using different methods. The model can be validated by simulating the model with different situations and inputs.

The operation of any model of a system is the simulation of that system [Mar97]. The modeling actually gets tested through the simulation process. A simulation software can be built according to the modeling to do the simulation. So simulation is done before making a new system or altering parts of an existing system. Through simulation, the errors and concerning matters of any modeled system can be known. Moreover, simulation of an existing system can be used for further process improvement.

The modeling and simulation process are iterative processes. It first starts with identifying and formulating the problem. Then to develop the modeling part, real-world system data should be collected and processed. After the modeling part is done, the modeling must be experimented through using a simulation software. Therefore an appropriate experimental design and software need to be selected. When the simulation software is ready to be used, the simulation run can be performed. The simulation results must be analyzed, interpreted and documented to validate the model. Validation means comparing the model's performance with the performance of the established real system under known conditions. The documentation with detailed objectives, assumptions, and input variables is also useful for further uses and recommendation. In all stages of the modeling and simulation, human decision making is required [Mar97]. Figure 2.8 depicts the whole process of modeling and simulation.

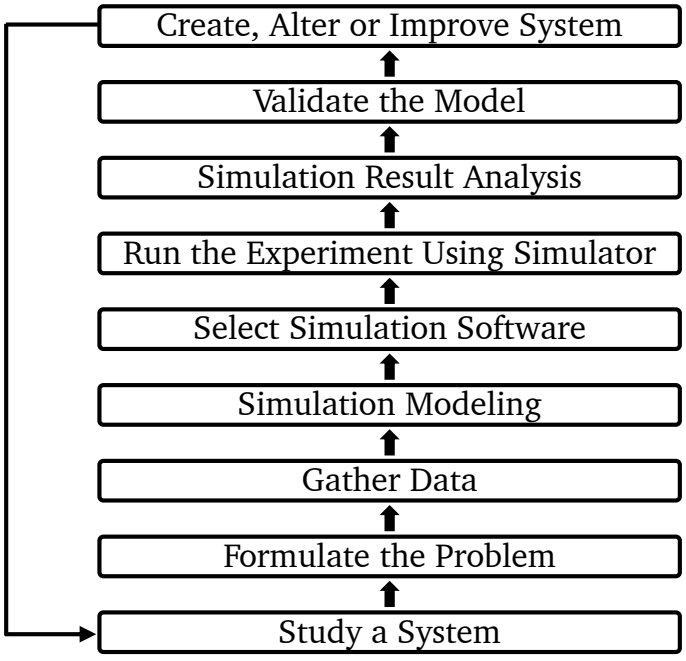

Figure 2.8: The modeling and simulation process

3 Related Work

In this chapter, we will discuss related works that are relevant to the concept and implementation presented in this thesis. In Section 3.1, the established methods of situation- and context-recognition are discussed. Several works on modeling and simulation of situation and Situation-Aware Systems have also been introduced. The Sub-Sections handle exclusively the SitOPT project where the purpose of the project is to provide Situation-Aware Workflow Management System. Later, in Section 3.2, we take a look at previous CPS modeling and simulation methods. The chapter finishes with the related work of CPS for vehicles in Section 3.3.

3.1 Situation Recognition

Häussermann et al. [HZL10] have provided a level model of situation recognition (See Figure 3.1). In this model, it is shown how data become information and information become knowledge at last. Information can be divided into two types: rudimentary information and high-level information. Knowledge is divided into three types: general knowledge, specific knowledge and knowledge about certain needs. This transformation of data becoming knowledge revolves the whole process of sensors data becoming a situation. A sensor system gathers sensor data from the existing context which can be mapped to the data level. The sensor data become observable context by adding metadata, e.g. quality, into it. Observable context is mapped to rudimentary information. With analysis, the observable context can be turned into the high-level context which is equal to the high-level information. The high-level information is then used for the recognition of situations. From the high-level context, the situation template is prepared by modeling the action of the context. The situation template is actually a situation type which can also be taken as general knowledge. Situation type can be modeled as situation token by adding event data with the type. Therefore, general knowledge becomes specific knowledge as it also gives the options to end users to know about what can happen from the situation template. This is how the situation is modeled and recognized step by step. From the situation token or recognized situation, the workflows of a Situation-Aware System can be triggered as from the recognized situation the system can acquire the knowledge about certain needs. Therefore, the workflow is selected and started based on the system's need.

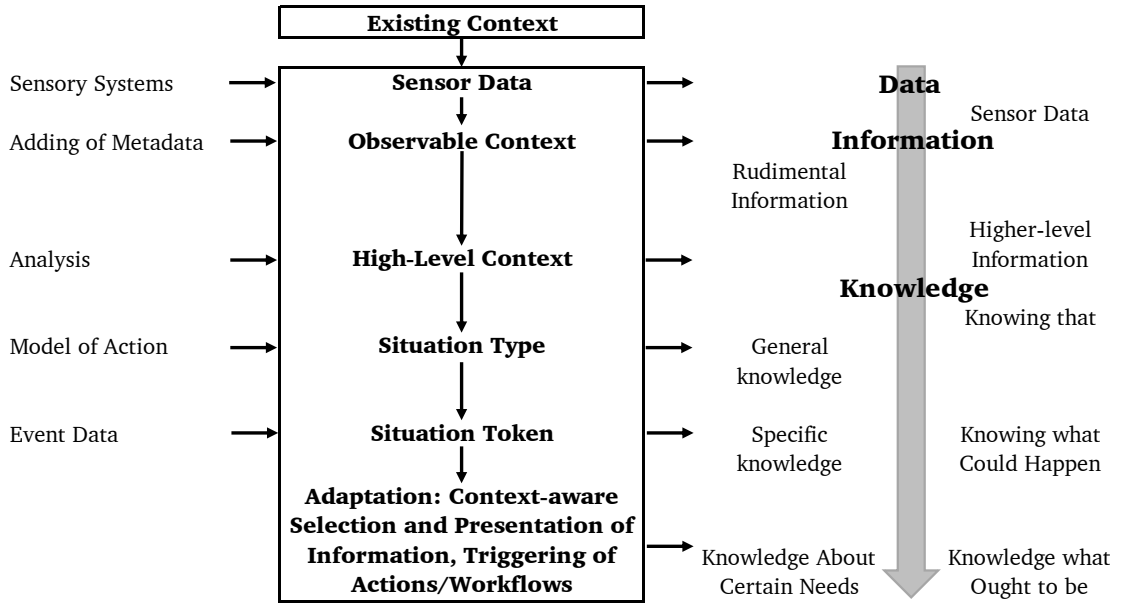

Figure 3.1: Level model of situation recognition [HZL10]

Hirmer et al. [HWS+16] present a situation recognition layer for a Situation-Aware System. Situation recognition layer has situation recognition service which can be deployed on a local machine or as a cloud service or in a hybrid way. The method for the situation recognition is based on the following three steps: register sensors, modeling situation templates, executing the executable situation templates to do the recognition of situations.

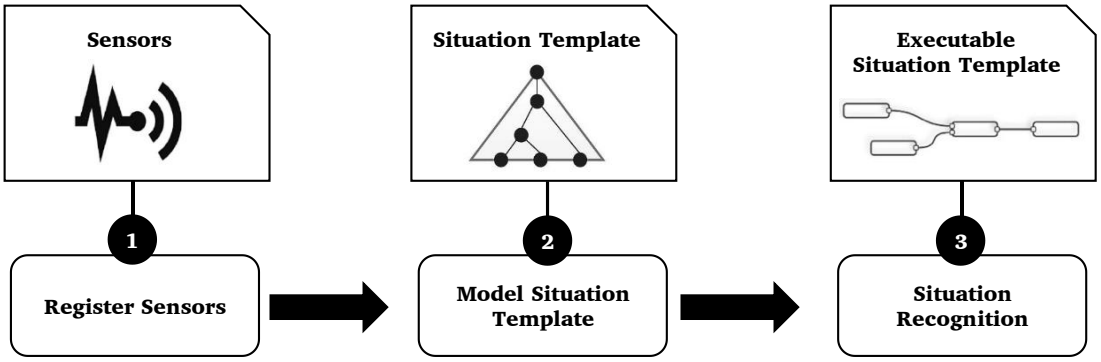

Figure 3.2: Method of situation recognition according to [HWS+16]

Figure 3.2 depicts all steps. In the first step, all the sensors get registered to a sensor registry where they are stored with their unique ID and type. In the second step, to recognize the situation later, situation templates need to be built. The situation templates (See Figure 3.3) are built here using Situation-Aggregation-Trees (SAT). The leaf nodes the situation template represent sensors and called context nodes. To filter all the incoming sensor data, some conditions are made to do the comparison.

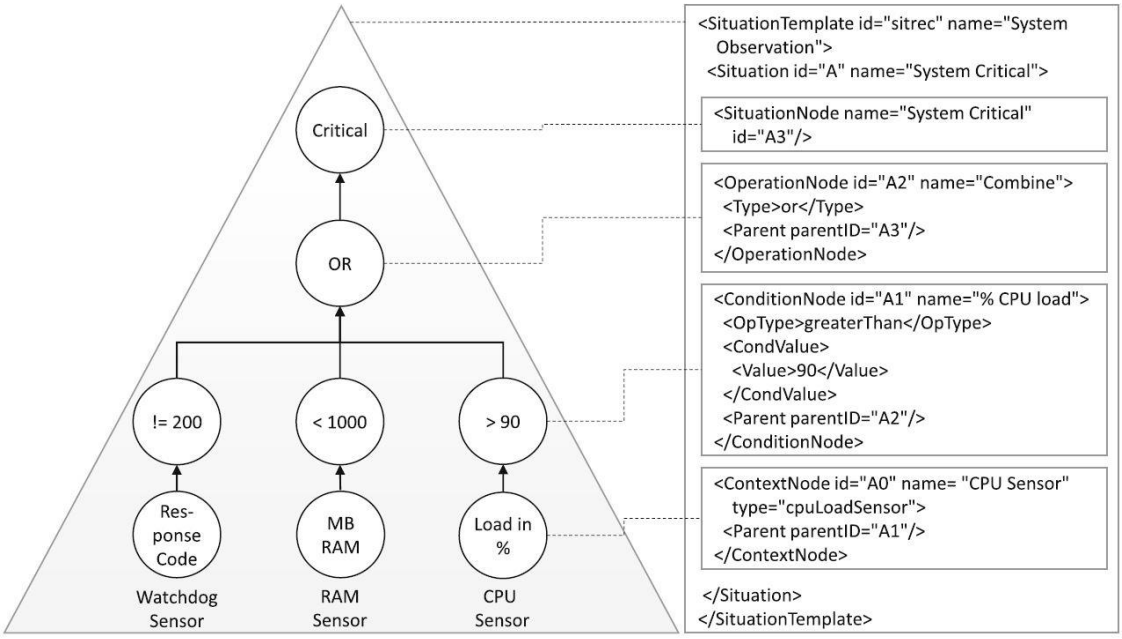

Figure 3.3: Example of a Situation Template Modeled by Hirmer et. al [HWS+16]

These are specified as so-called condition nodes. So, the context nodes get connected to the condition nodes. Then some operation nodes (e.g. logical operations like and, or, xor) are used to sum up the output of the condition nodes. When one situation template is executed, one situation is recognized by the system.
As an example, the authors [HWS+16] have provided a situation template which recognizes the situation "Critical" of a web server. Figure 3.3 shows an example of a situation template with Extensible Markup Language (XML) snippet. To recognize the situation here, data of 3 sensors are taken; watchdog sensor, RAM sensor and CPU sensor. The condition nodes that make the output true are (1) the CPU load percentage should be greater than 90 percent, (2) the available RAM should be lower than “1000MB”, and (3) the response code of the machine should not equal the number “200”. The operation node is set to ’OR’. So, if one or more than one of the condition nodes are true the web server situation can be recognized as ’Critical’.
A situation aware system has more than one situation templates. So to keep the system running, the recognized situations from many situation templates get registered, transformed to a flow for deployment, executed and later de-registered.

To model and simulate a situation aware system, the situations should be recognized at first. The modeling and simulation can be followed by the recognition of situation. When an individual situation is recognized, modeled and simulated, the whole situation-aware system can be modeled and built. This whole process is very tedious and expensive [BHK+15]. Therefore, the SitOPT Project was started to ease the modeling and execution of Situation-Aware Applications based on Workflow Technology.

3.1.1 SitOPT Project

SitOPT is a research project with the aim for optimizing and adapting situational applications based on workflow fragments[1]. The workflow fragments are dynamic in nature as the situations are very dynamic. The SitOPT project's main architecture is developed based on three layers. In Figure 3.4 it is shown, the top layer of the architecture which is the application level depends on the dynamic situations and workflow fragments. The lowest level does efficient sensing by automatically connecting and centralizing all sensor data. The situation recognition level autonomously recognizes the situations based on the situation templates that are developed from the sensor data [HWS+16].

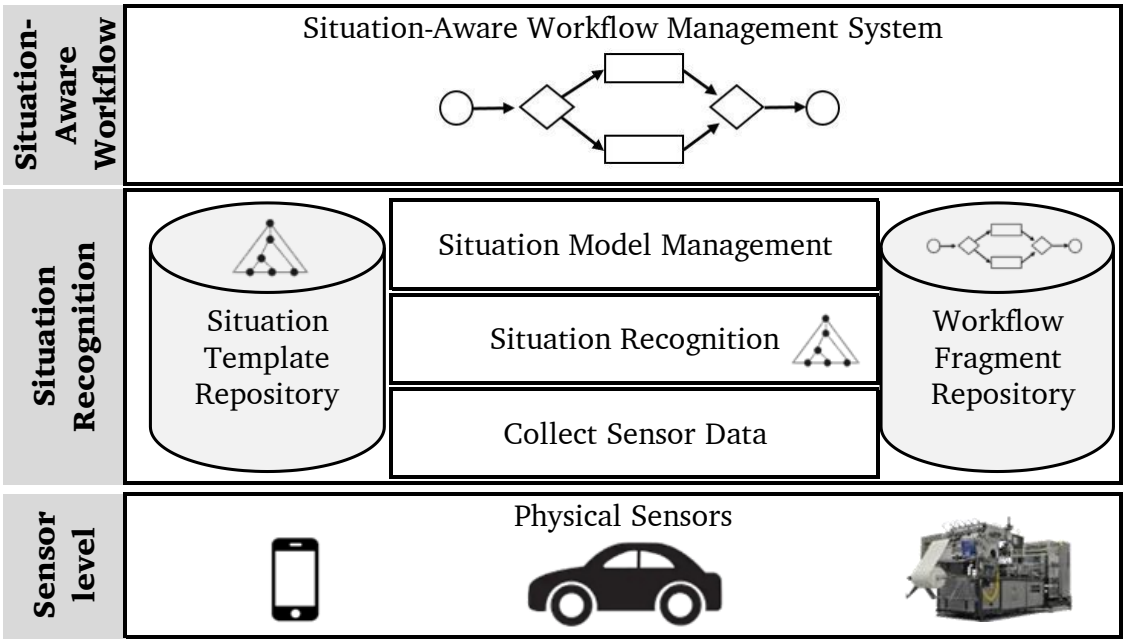

Figure 3.4: SitOPT architectural overview [HWS+16]

3.1.2 SitRS and SitRS XT

Hirmer et al. [HWS+15] present the concept of SitRS architecture with three layers, from the bottom: context layer, situation recognition service layer and application layer. The two layers in the bottom consist of the situation model, the situation recognition service, and the sensors. The physical objects with sensors, e.g., machines, transports, registered their sensors in the sensor registry. From the sensor registry, the situation templates get input and these templates are stored in the situation template repository. There is a situation registration service which is responsible for registering the occurred situation based on the templates. The situation templates become executable situation template as they get

[1] https://www.ipvs.uni-stuttgart.de/abteilungen/as/forschung/projekte/SitOPT

mapped to executable representation. An execution engine does the execution of situation templates and from the output of the engine, it can be confirmed which modeled situation has occurred and when.
Franco da Silva et al. [FHWM16] represent a situation recognition service SitRS XT which is an extension of SitRS. SitRS XT enables the situation recognition in real-time. SitRS XT does the situation recognition based on Complex Event Processing (CEP). According to David Luckham [Luc08] "Complex Event Processing (CEP) is a defined set of tools and techniques for analyzing and controlling the complex series of interrelated events that drive modern distributed information systems."

3.1.3 Situation-Aware Workflow Modeling Extension (SitME)

Breitenbücher et al. [BHK+15] introduce an approach to model situation aware processes without making the situation handling logic too complex or unmanageable. This is known as Situation-Aware Workflow Modeling Extension (SitME) method and Figure 3.5 depicts the overview of it.

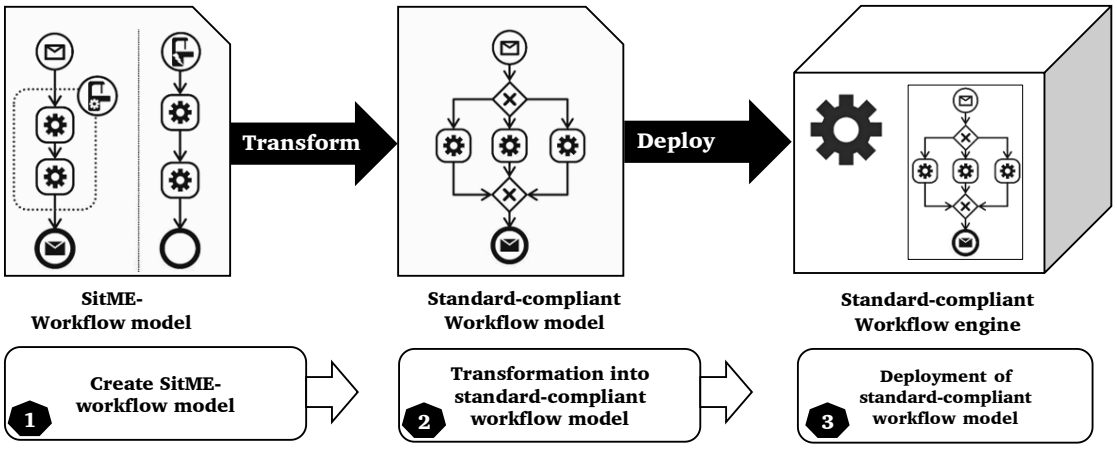

Figure 3.5: Overview of the SitME-method [BHK+15]

In SitME, the first step is to create a SitME workflow model. Then the model is transformed into a standard-complaint workflow model so that it can be deployed in a standard-complaint workflow engine. So the first step is very related to our work here which is situation-aware modeling. To make situation-aware modeling, situation events are introduced at the very first place. Situation events are actually activities. They hold two information: i) a situation name and (ii) the unique identifier of the corresponding object for which the situation has to be observed (See Figure 3.6).

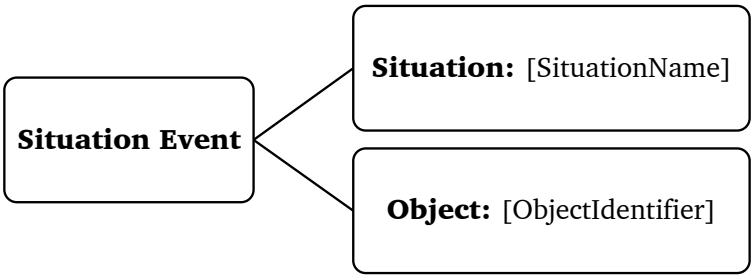

Figure 3.6: Situation event with name and object [BHK+15]

Then comes situation scope that means making a group of activities which will be executed only if certain specified situations come over. The situation scope brings the situation-triggered workflow fragments. These fragments handle the changing situations of a system through these event-driven process chain.

3.1.4 A System for Situation-aware Access Control (SitAC)

Taking the SitOPT architecture as a basis, Hüffmeyer et al. [HHM+17] a propose Situation-Aware Access Control System (SitAC) to protect various kinds of RESTful services. These sort of systems have to have the capability to determine access decisions in a short time. The system also needs to be flexible to create, change and remove services easily. The architecture of A System for Situation-aware Access Control (SitAC) is built on top of physical and environmental objects in the real world. The architecture has three layers: service layer, security layer and client layer. The service layer covers the services that need to be protected and the SitOPT system. From the top client layer, clients register sensors at the situation administrator API which is available in the security layer. The situation administrator API is connected with the RestACL[2] of the same layer. The RestACL sends the sensor registration request to the SitOPT system. A client can also register situation templates to the situation administrator API. Like the sensors registration, the RestACL forwards the situation templates to SitOPT system. At the same time. the RestACL also registers itself for callbacks so that the SitOPT system will inform RestACL if any situation occurs. This is how the situation recognition method of SitOPT helps here. Additionally, when a situation occurs, the access control enforcement point gives access to clients who have the permission to access the system.

[2] RestACL: An Access Control Language for RESTful Services (Source: https://dl.acm.org/citation.cfm?id=2875494)

3.2 Related Modeling of CPSs

The CPS modeling represents the key to the system implementation. Tan et al. [TGP08] represented prototypical modeling of CPS concept in their paper. Their modeling has some of the following parts: *Event/Information Driven* - here events are the raw data collected by the sensors units and actions that are made by the actuators or humans. The abstraction of the physical world is represented by this information. *Global Reference Time* -is a part of the Next Generation Internet initiative that will provide the Global Reference Time to all CPS components. The Next Generation Internet [AKK13] enables the applications to have the ability to select paths for data transfer between the source and destination. *Quantified Confidence* - a standard method to calculate the confidence of the events/information at any point in time. *Publish/Subscribe Scheme* - based on the system goals; every CPS control unit takes care of an individualistic group of events or information to publish them when necessary. *Semantic Control Laws* - the control laws control the system behavior as they have the user-defined condition or action forms. Finally comes, *New Networking Techniques* - provide the global reference time, new event routing or new information routing and data management schemes.

Talcott et al. [Tal08] propose an event-based semantics for CPSs as it gives the detailed information about the system and it is easier to integrate the interactions between components in this way. Two compositional models are introduced here; one for the autonomous agents and another one for the interactive agents. In both models, the interaction and communications between the cyber and physical components are emphasized.

Wang et al. [WCG08] analyzed CPS for large-scale industrial processes where the system needs to handle heterogeneous components together. They presented a unified control and network concept to integrate all the heterogeneous components to keep running the real-time operations.

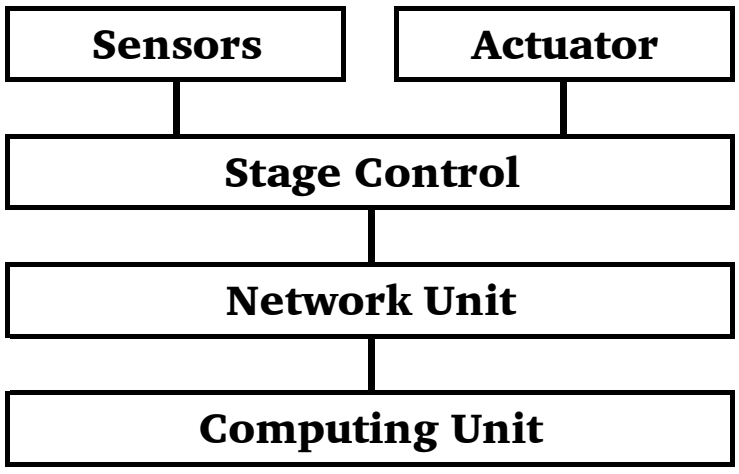

Figure 3.7: Proposed CPS architecture by Wang. et al[WCG08]

According to the paper [WCG08], a broader sense of cross-layer design needs to be employed in CPS. From the cross-layer communication techniques, a modeling for cross-layer integration and communication is designed where each device needs to be designed based on the individual's sensing, actuation, hardware, type, operating system (OS), and middleware (See Figure 3.7).

Balaji et al. [BFD+15] presented a modeling that combines individual models for information, energy, user, operation and maintenance of a CPS. To develop a CPS, understanding the *information model* is the key point as other parts of the system depend on it. Information can be static (e.g., traffic point location) or dynamic (e.g., vehicle's speed, location). To understand the energy flow and identify the opportunities to improve the energy level, an *energy model* is necessary to build an efficient energy simulator for the CPS. As user comfort, safety and productivity are the main reasons to build a CPS; *user model* must concentrate more on giving application and control level access to users. As thousands of sensors, actuators and embedded systems are installed in different levels of the whole Cyber-Physical System, there should be a defined *operation and maintenance model* to maintain, operate, and enhance the system later on.

West and Parmer [WP06] propose to develop a CPS based on a software architecture composed of a collection of application-specific services. According to them, "A cyber-physical system architecture would sensibly consist of a small executive capable of dynamic service composition and component service isolation" [WP06]. Not all of the services of a distributed embedded application might be physically local, so the small executives will have the capability of locating, authenticating, retrieving and communicating with remote services. Moreover, to satisfy the system constraints which is made with heterogeneous hardwares, the modeling of the CPS must handle the automatic composition of services.

Bujorianu and Barringer [BB09] propose a modeling solution to overcome the challenges of formal semantics of the CPSs paradigm. Semantics should have the capability to combine continuous and discrete mathematics of the physical and computational aspects. To complete that, the approach is to combine the denotational semantics with an algebraic model for physical processes in order to evidence the holistic perspective of the CPSs paradigm. So, the actual proposal of the authors is an integrated logic for the specification of the observability and quantitative properties. The logic is inspired by the "Hilbertean formal methods" and is developed and constructed by integrating different system features.

3.3 CPS for Vehicles

CPS for the vehicles is a sensitive area to model as any system for vehicles needs to meet few standards for safety and efficiency. For example, a CPS for the traffic control system will need to calculate a lot of complex traffic control algorithms. So that it can give the best route from the current situation as output to the end user. Therefore, CPSs for the automotive industry require high computing power [AKK13].

Ahmed et al.[AKK13] propose a vehicular CPS scenario where the vehicles are connected to RSU. RSU is connected to the Next Generation Internet which provides different services through a wired network. Next Generation Internet has two main modules: (1) Service Aware Modules (2) Data Management Module.
Modern cars nowadays have different actuators like speed controlling, lights, brakes, etc. Service aware module can communicate and control the different actuators in real-time through the Application Module which is the bridge between them. This is how their proposed vehicular CPS modeling can provide a single car the best application for different types of services efficiently. For example, in the case of waiting people on a traffic point, lights should be on automatically.

Chen et al. [CYH+17] studied the topology of the vehicles in traffic flow to propose CPS enabled traffic flow modeling. They primarily calculated velocity and density of the vehicles to understand the topology. To estimate the density and the velocity of the vehicles, the mathematical model uses local CPS sensors of the autonomous vehicles. The local and cloud layers of the CPS use visual, location and motion sensors data of the vehicles. The model provides a differential equation by taking 5 scenarios in consideration: (1) Continuous Traffic Flow; (2) Vehicle-following Traffic Flow; (3) Non-Ramp/Intersection Traffic Flow; (4) Traffic Pressure and (5) Viscous Traffic Flow. From these scenarios, information like the instant velocity, the speed limit, the distance between vehicles and the momentum are collected to analyze the driving scene.

Wan et al. [WZZ+14]s multi-layered architecture for the vehicular system have different layers for computation: vehicle computational Layer, location computational layer, cloud computational layer. In the vehicle computational layer, sensors are installed in the vehicles, so that they can provide all kind of environmental and their own body's parameter data. In the location computational layer, there are RSUs. RSUs are deployed at strategic locations with necessary sensors and equipments. They are deployed on the roads in such a way that they can exchange information with all the vehicles available on roads in a certain area. Vehicle's sensors and neighboring RSU's sensors share context-aware traffic information and entertainment resources. If some vehicles are not connected to the RSU, they may be still connected to the rest of the vehicles which are connected to RSU. So with the help of each other, the CPS can produce real-time traffic information. In the cloud computational layer, traffic-related authorities and different companies can supply various services and applications to the end users.

Hoang et al. [HPK12] propose another CPS architecture named WebMed which can be used for vehicles parking management. Figure 3.8 visualizes that in their architecture, sensors are in the core level as they provide the data of the vehicles to the WebMed nodes. WebMed nodes consist of the node API, data model, device control manager, and data manager. These nodes provide the gathered information to the Web Service Enabler. Later the Service Enabler transfer the data to the Web Service Repository where all the service like reservation, currency, SMS, payment are integrated. Users can access those services by using user application interfaces. So from the calculations from those services, system administrators and end users get messages like confirmation of parking reservation, overtime parking alert, non-pay alert through their own versions of applications.

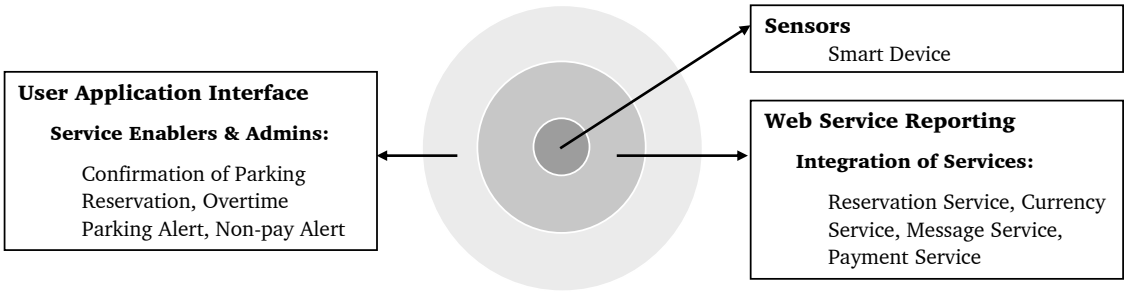

Figure 3.8: Garage management system of WebMed[HPK12]

When vehicles maintain a short distance to follow each other, it can be said that they form a platoon-based driving pattern. Jia et al. [JLW+16] mentioned in their paper platoon-based driving pattern of vehicles can actually improve road capacity and energy efficiency. They propose that a platoon-based vehicular CPS can be modeled if traffic simulator and networking simulator can be combined together with the help of communication interface. The traffic simulator, in this case, broadcasts the real-time tracking information of the vehicles to the network simulator. At the same time, the network simulator will inform the vehicles any received alert-message and also will inform the traffic simulator the vehicle identities. So that traffic simulator can change the traffic control plans accordingly.

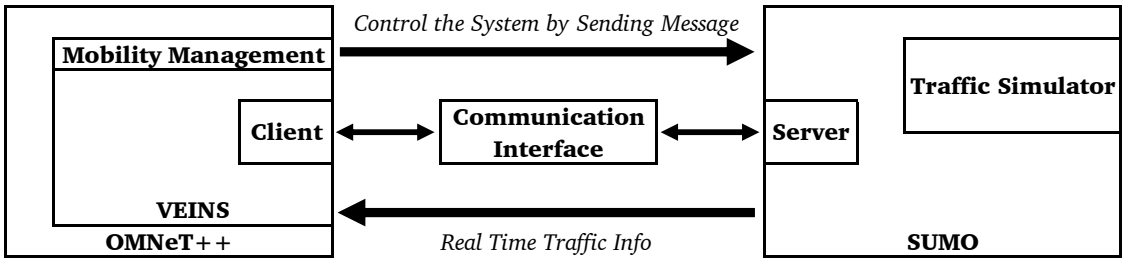

Figure 3.9: Veins architecture detail [JLW+16]

The implementation of this modeling is named 'Veins Architecture'. Some well known vehicular simulators are integrated here. Veins is an open source framework for running vehicular network simulations. Figure 3.9 shows the details of Veins architecture. It is based on two well-established simulators: OMNeT++, an event-based network simulator, and Simulation of Urban MObility (SUMO), a road traffic simulator[3]. The simulation steps are enlisted below:

- In every simulation step, a vehicle in Veins will send related traffic information to the SUMO simulator.
- Then every vehicle in SUMO uses the received information from Veins as the input of the controller to have the expected velocity and acceleration.
- In the next step of the simulation, the movement of the vehicle is simulated by implemented SUMO.
- Later, the movement information of the vehicle is sent back to Veins, so that Veins can update the movement information of the vehicle in the networking graph.

[3]https://veins.car2x.org/

4 Concept and Modeling

This chapter gives an overview of the conceptual system of modeling, simulation and recognition of situations in MCPS in the first Section 4.1. Section 4.2 explains the required basic elements to establish the Simulation System. In Section 4.3, the relationship among different objects in Simulation System and Situation Recognition System is analyzed. Section 4.4 discusses modeling scenarios for MCPS providing an example system. Section 4.5 provides the algorithms details of the recognized situations of the Situation Recognition System for MCPS.

4.1 System Overview

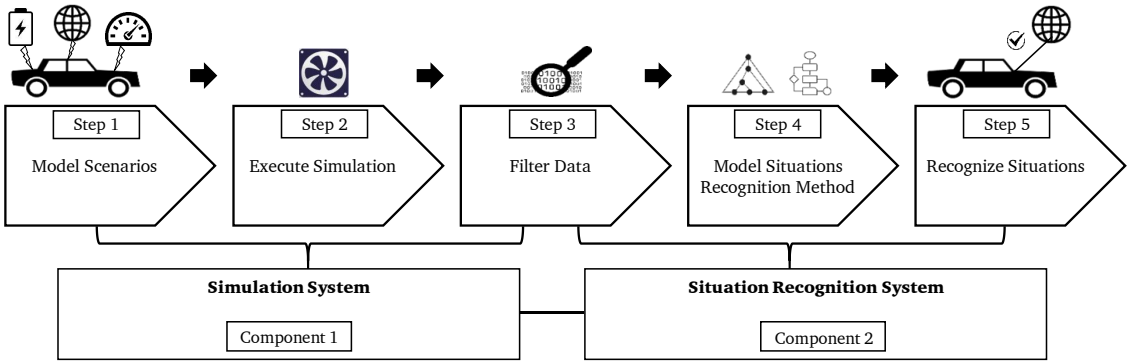

Figure 4.1: Overview of the complete system with Simulation System and Situation Recognition System including Processes and Structures of Modeling, Simulating and Recognizing Situations

We present a conceptual overview of the Simulation Framework that enables the modeling and simulation of Situation-Aware MCPS scenarios. The system structure consists of a *Simulation System* which is considered as the *Component 1* and *Situation Recognition System* which is considered as *Component 2* of the system. Figure 4.1 depicts the overview of the Simulation Framework, the process of the system starts with depicting the basic structure as components that implement the delineated process.
The first main component which is the *Simulation System*, has the task of generating simulation data. The second main component of the system which is *Situation Recognition System*, is used to detect situations on the generated simulation data.

With the defined scenarios, a suitable simulator needs to be selected or built. The simulator environment should be prepared in a way that most of the scenarios can be fit into it. Simulator needs to give sensors and actuators detail data as output in a compatible format, e.g., XML, JSON.

The first step of the *Simulation System* is modeling scenarios (Figure 4.1). To model scenarios, some basic components of an MCPS are considered, e.g. sensors, devices, users, and Section 4.2 explains about them. While modeling MCPS scenarios, all the objects related to the system are needed to be considered. Section 4.4 defines some scenarios that should be considered for an example MCPS which is vehicular software update.

To get the output, after defining the necessary scenarios in the selected simulator, the simulation is executed. This is the second step available, in the *Simulation System*. As the simulator needs to accumulate the defined scenarios of the MCPS, the simulator also should have the components from Section 4.2 in it. The execution of the simulation provides virtual sensor and actuator data for the defined MCPS scenarios. The generated simulation output data contain all raw data. Here, the data defines the environment of the selected MCPS (e.g., vehicle, smart factory, etc.) where the situations occurred.

The next step is filtering the output data of the simulator. Based on the modeled scenarios, the system may need multiple simulators. The simulators may generate multiple outputs of different objects. The formats of the outputs can be different. But it is easier to define, recognize, and execute situations from one data format and stream. So it is important to make one output file using one format by aggregating all output files from all used simulators and get rid of the unnecessary context or sensor information.

In *Step 4* which lies in the *Situation Recognition System*, the situations are defined. This step comes in the system to define the method to recognize the situations. To recognize situations, it is needed to define them first [HWS+16]. Situations are definable through situation recognition algorithms. We have modeled situation recognition algorithms in Section 4.5. Suppose, an electric car wants to take any software update while running on the highway. So a high-level situation could be "Ready for Update". For this, it needs to be checked if the car has enough charge in the battery, a strong network connection to RSU, is not crossing the maximum speed limit of taking an update. If the car's maximum battery capacity is 35000wh, the battery charge should be more than 15000wh, the distance between the car and any nearby RSU should be within $200m^2$ and the car must not cross the speed limit of 60kph. When all of them are true which means combined with an AND clause, only then the car can take a software update which, e.g., takes 40 seconds.

The last step of the system is executing the situation recognition algorithms or pseudocodes by using any available programming language to recognize the situations from the aggregated filtered output data. The aim later is to aggregate multiple situation detection algorithms to come to a decision at any time step. For example, to change the temperature inside a car at a particular time step, it is necessary to check different attributes of different situation recognition algorithms.

After the situations are detected by situation detection methods, the simulation data is enhanced with the situation data. These situation data later can be used to take decisions on the application level.

4.2 Basic Elements of Modeling and Simulation

Based on the overview of the system, the basic required elements of the *Simulation System* component can be derived. It is very essential to distinguish and define every component of the MCPS to define and execute simulation (Figure 4.2). It is more convenient to manage the system when all the functional components are combined and utilized together. This is an abstract view of the elements required in an MCPS [GBF+16]. Moreover, as they are required to define scenarios of an MCPS (Step 1 of *Component 1* which is *Simulation System* of Figure 4.1), they are also required in the simulator to execute the simulation (Step 2 of Figure 4.1). The required components are defined as follows:

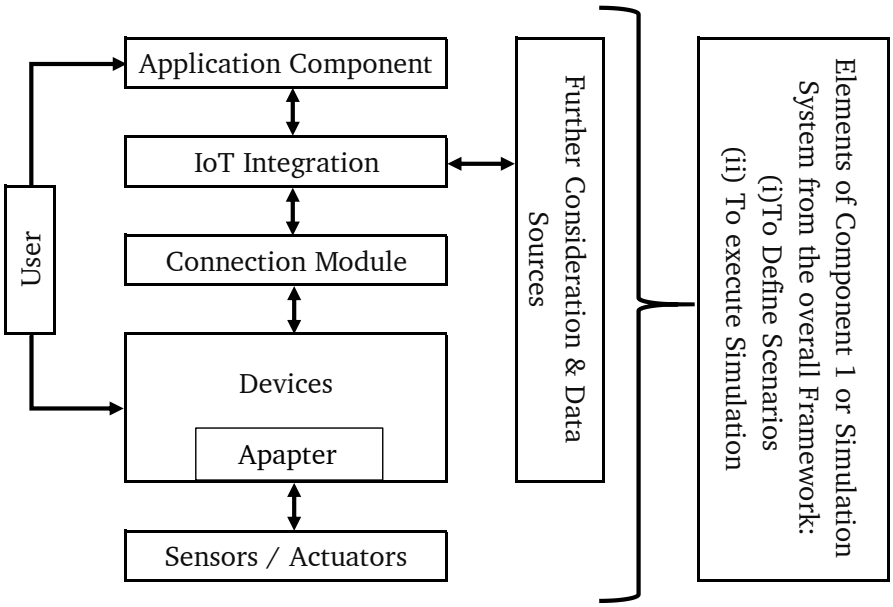

Figure 4.2: Abstract Basic Elements of the MCPS and Simulator Required for Defining the Scenarios and Executing the Simulation of the Simulation System Component

Sensors and Actuators
Sensors are lower level hardware components. They can be virtual too when the devices are virtual, e.g., the devices and the sensors of a simulator software. They gather all data of the physical space of any system. Sensors can measure any sense like the presence of a physical object in an area, temperature, humidity of a particular area, pressure on a certain space. They gather data to send them to all the connected devices of the system. To do this,

sensors do not need many logics. They convert real-life changes of a physical environment into electric signals, so that devices can read the changes and act according to it [ADS02]. Actuators are also nothing but hardware components and can be virtual too like the sensors. Where sensors can only read and gather information, actuators can perform a command. Suppose, the optimal temperature of a room is set to 25 degree Celsius. If the temperature rises to 30 degree Celsius, the actuator will turn on the Air conditioner to lower down the temperature. Whereas sensors send the gathered data to the connected devices, actuators perform commands from their connected devices based on the sensor information. They act on the converted electric signals and translate the signals to a physical action [ADS02]. After they have performed the command, they can also respond back to command sender devices. Based on the provided data of sensors and capability to act of the actuators, situations can be modeled in an MCPS. They are the backbone of the systems.

Devices and Users of MCPS
Devices are hardware components where the sensors/actuators are integrated. There can be many devices in an MCPS. In situation modeling of an MCPS, the presence of the users are often mandatory, depending on the scenario. Suppose, in MCPS for vehicular communication platform, devices like cars, smartphones, RSU, and users like car owners, RSU admins or people on the roads are the inevitable parts. Devices and users can communicate with the whole platform using transport protocols.

Connection Module
Connection modules are responsible for enabling communication among the devices to devices, devices to users and devices to the application module. Gateway in the connection module lets the devices be connected by supporting different communication technologies and transport protocols. For example, RSUs of a particular area can disseminate any available software update for vehicles which are eligible to take it. If the vehicle is an autonomous vehicle, it needs to decide whether it takes the update or not based on the current situations. Taking the update itself is a modeled situation that can only happen when RSU and vehicles are connected to each other through any communication protocol. Again, the connection among them can differ based on the distance between them. Moreover, these sort of situations also needs to be modeled based on the context information of the connection module.

IoT Integration
To model and simulate situations in MCPS every object of the system needs to be integrated together (See Figure 4.2). The smartest way to do this integration is IoT integration as it enables the connectivity of different devices in a smart environment so easily [AIM10]. In a MCPS, one single component or object can take parts into multiple roles. For example, in the MCPS of a vehicular communication platform, the car can be a source of sensor data, actuator, communication medium. So when the car is a part of IoT integration, any role or attribute of the car is available to others.

Further Consideration and Data Sources
In the overview of the whole system (See Figure 4.1), we have shown that data filtering is one of the main steps (Step 3). With the data filtering step, it can also be necessary to add external information including attributes with the data generated by the sensors. Also, when a system's components are connected by IoT, it is very possible to get data from other connected sources. For example, we define the scenarios in the very first step of the framework. These scenarios are simulated in the simulator to get virtual device's sensors data. Now, it is much likely that the generated data from the simulator output is not enough to model and simulate the situations. Therefore, enabling to add additional data from external resources is achieved within this step of the system process.

Application Component
The application component of an MCPS represents more applications connected to the IoT integration through a common application interface (API). Additionally, using the application interface users can execute, control or stop a situation recognition (Figure 4.2). Situation recognition techniques, e.g. algorithms or templates, can avail the workflow of the system in many different ways.

4.3 Relationship Analysis of the System

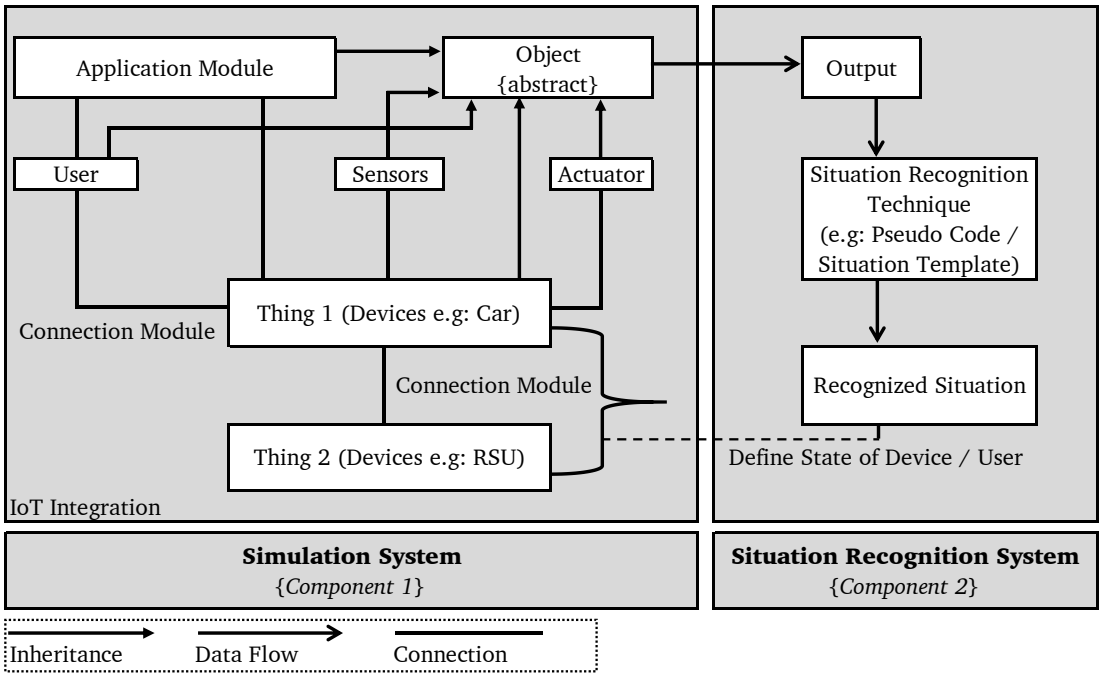

Figure 4.3: Relationship Analysis of the System including Simulation System and Situation Recognition System [MHWM17]

The analysis of the system shows how different objects are related to each other for modeling, simulation, and recognition of situations. Figure 4.3 shows the relationship among the objects of the steps of *Simulation System* and *Situation Recognition System*. To combine every object together let's take an abstract class 'Object' [MHWM17]. All other objects in the system of modeling and simulating situations are related and inherit properties from the abstract class. In the situation recognition step of the relationship analysis, a Situation Recognition Technique defines a particular situation through algorithm or pseudocode. Therefore, Recognized Situations define the states of things or users.
In the modeling and simulation steps, a thing contains many sensors and actuators. Sensors and actuators respectively observe and perform actions for a thing in situation aware MCPS. There are also users in an MCPS who maintain and manipulate things, either directly or through using the application module. All the objects of the situation modeling and simulation steps are connected to each other through the Connection Module, e.g., things to things, things to users.

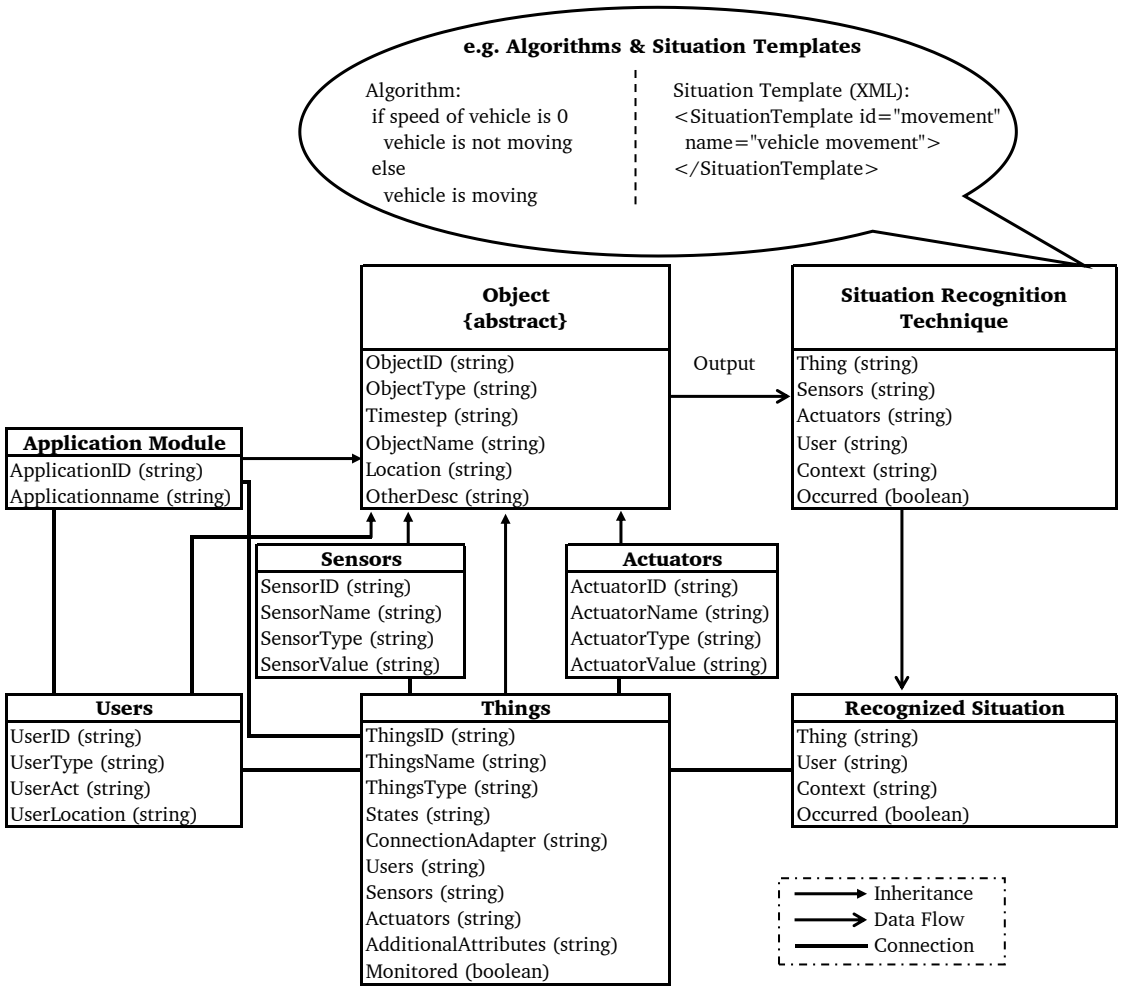

Figure 4.4: Attributes of different Objects in the System of Modeling, Simulating and Recognizing of Situations

Every object in this relationship has attributes and they are depicted in the Figure 4.4. The abstract class 'Object' has attributes like *ObjectID, ObjectType, TimeStamp, ObjectName, Location, OtherDescription* [MHWM17]. Other objects can inherit some attributes from the abstract class if needed. A situation recognition technique (e.g., pseudocodes, situation templates) which is algorithms or pseudocodes in our model, take inputs from the simulation output. The output is generated after the simulation process occurred in the simulator where Things and Users were modeled. The simulation output file can be described in a format like XML. The Situation Recognition Technique, e.g., pseudocodes or templates, define the necessary context information and condition that is needed to define and recognize a particular situation.

The conditions of the Situation Recognition step helps to model and simulate the situations through algorithms. Situation Recognition Technique must acquire all attributes from the Simulation step's output because different attributes are needed for different Situation Recognition process. Therefore, Situation Recognition Technique takes attributes from the *Simulation System's* output, e.g., *Thing, Sensors, Actuators, User, Context, Occurred*, and Recognized Situations contain the attributes like *Thing, User, Context, Occurred*. Here, context is the information that is used to characterize the situation of an object [ADB+99] where objects are the things, persons, sensors and so on. Things have to have attributes like *ThingID, ThingName, ThingType, States, ConnectionAdapter, Users, Sensors, Monitored, Actuators, AdditionalAttributes*. Connection adapter defines the connection details among objects. Things also can inherit the attributes from the abstract class 'Object'. Sensors, Actuators and Users are directly connected to Things. Sensors need to have attributes like *SensorID, SensorName, SensorType, SensorValue*. Actuators have attributes named *ActuatorID, ActuatorName, ActuatorType, ActuatorAction*. Sensors and Actuators need to be registered and integrated at the IoT platform to be connected continuously in order to transfer gathered data. As Users maintain the Things, they need to have *UserID, UserType, Act, Location* attributes to maintain the relationship with Things. Finally, the Application Module is defined with the attributes like *ApplicationID, ApplicationName*.

4.4 Modeling Scenarios for Mobile CPS (MCPS)

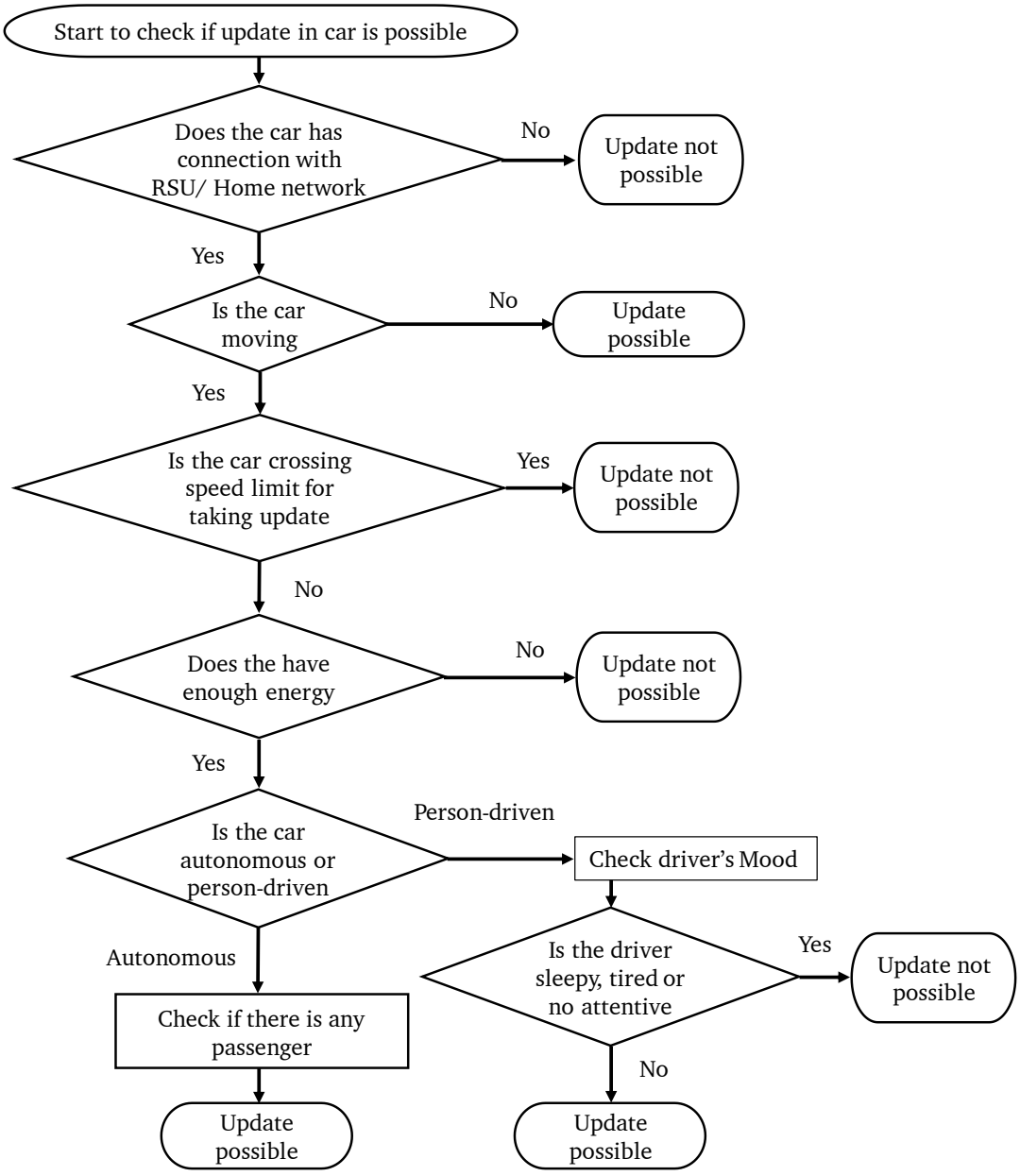

Figure 4.5: Possible flowchart to model scenarios in Simulation System (Component 1) for Vehicular Scenarios

As a simulation scenario, we take the problem of applying Software Updates in MCPS, where issues like connectivity, state of the cps (such as energy level, GPS coordinate, interaction with users) make it hard to schedule these updates. In Figure 4.5, we have modeled possible scenarios for MCPS, e.g., vehicular software update, in a flowchart. Before any vehicle starts to take software update, it will check several situations. At first, it

is needed to be checked whether the car has a connection with RSU or home network. If the car has no connection, it can be decided that the car is not capable of taking update at that moment. When the connection situation is positive, then it is needed to be checked whether the car is moving at that time or not. When at a particular time, the car has a connection with either RSU or home network and is not moving; the car can take the update as there is no major risk. Otherwise, when the car is moving in a particular time, it must be checked if the car is crossing the speed limit for taking any update. In case the car is crossing the maximum speed limit, the car is not allowed to take the update. If the car is not crossing the speed limit, the next step is checking the energy level of it. If the car has enough remained energy, it is now time to check whether the car is an autonomous car or a person-driven car. In case the car is a person-driven car, it is a must to check the driver's mood during that time. If the driver is tired, sleepy or not attentive at that time, it is advisable not to take the update. However, whereas the car is an autonomous car, the next input can be checking about the number of passengers in the car, and it can take the available update.

4.5 Modeling Situation Recognition for Mobile CPS (MCPS)

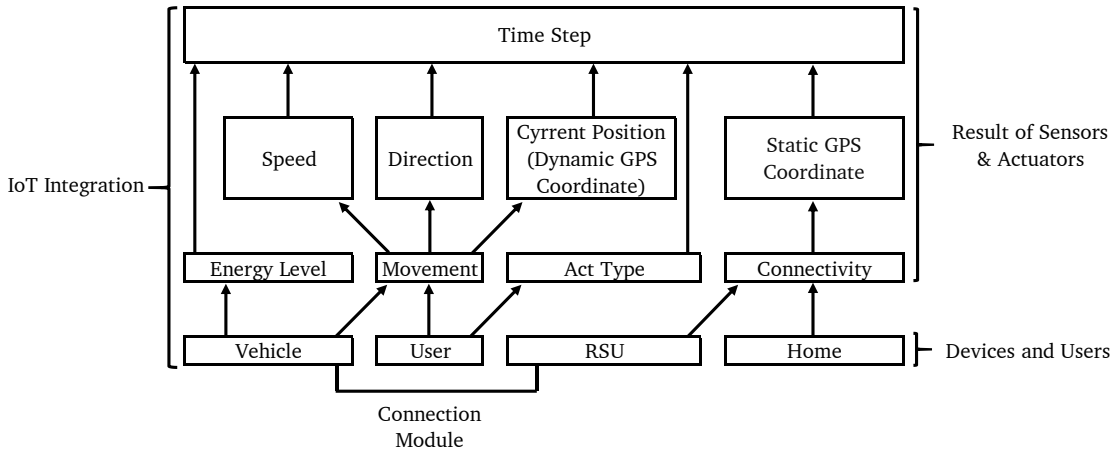

Figure 4.6: Elements of the Simulation System (Component 1) for Vehicular Scenarios

As we have modeled scenarios in MCPS for vehicular software update system, Figure 4.6 shows the elements that are explained in Section 4.2 for the Vehicular Simulation System. So in the simulator's simulation environment, the Devices or Things are vehicles, RSU, home in this case. Passengers, drivers of the vehicles and pedestrians on the streets are the Users. The sensors and actuators of the objects provide data of movement of vehicles, GPS coordinate of vehicles, RSU or Home to measure the connectivity strength among them, energy information details of vehicles, act type and movement of Users. The movement information includes the current position, direction of movement and speed of the objects. All the information is available for any time step of the simulation (Figure 4.6). All these attributes information are needed as output to recognize the situations. To finalize the

Simulation System of the modeling and simulation of situations system, the vehicular system is designed in this way.

Now, comes to the situation recognition algorithms or pseudocodes modeling as it is designed in the *Situation Recognition System*. For a vehicle to take software update, the following situations need to be recognized. The algorithms are implemented to recognize situations in Section 6.2 with more explanatory details.

If the vehicle is moving or not (Algorithm 4.1). This algorithm needs vehicles and the movement sensor data of them. The logic here is if the vehicle's speed is 0 (See Line 6 to 9), the vehicle is not moving. Otherwise, it is moving at a certain speed. The algorithm checks it for every time step.

Algorithm 4.1 Pseudocode to check if a vehicle is moving or not

```
1: for each timestep in simulation
2:     get iteration of timestep
3:     for each entity in timestep
4:         if entity is equal to vehicle
5:             get all vehicles speed, id
6:             if speed of a vehicle is more than 0.00
7:                 at timestep vehicle is moving
8:             else
9:                 at timestep vehicle is not moving
```

If the vehicle has connectivity to the internet through RSU or home WiFi or not. The logic here to check the distance (variables: A, B, C or D, here the value of the variables goes up alphabetically) among the vehicles, RSU or home. If there is connectivity, there must be a check for the strength of that using the same distance variables (Algorithm 4.2). This algorithm needs vehicles, RSU, home, the movement sensor data of vehicles, the GPS coordinates of the RSUs and home. The algorithm checks the network connectivity for every time step.

Algorithm 4.2 Pseudocode to check if a vehicle has connection with home, RSU or not and if there is connection which strength

```
for each timestep in simulation
    get iteration of timestep
    for each entity in timestep
        if entity is equal to RSU
            get all RSU id and axis point
        if entity is equal to Home
            get all Home id and axis point
        if entity is equal to vehicle
            get all vehicle id and axis point
            if vehicles axis point is within Am^2 from Home
                at timestep vehicle is at Home
            if vehicle axis point is within Bm^2 from any RSU
                at timestep vehicle has strong connection with RSU id
            if vehicle axis point is within Cm^2 from any RSU
                at timestep vehicle has medium connection with RSU id
            if vehicle axis point is within Dm^2 from any RSU
                at timestep vehicle has weak connection with RSU id
            else
                at timestep vehicle has no connection
```

Algorithm 4.3 explains a pseudocode to check if the vehicle is crossing a certain speed limit (Variable: X) or not to take the software update. As the variable here is X, if the speed limit is equal to or more than X unit, the vehicle is crossing the speed limit (See Line 6 and 7). Otherwise, the vehicle is not crossing the speed limit to take the update (See Line 8 and 9). This algorithm also needs vehicles and the movement sensor data of them. The algorithm checks the speed limit for every time step.

Algorithm 4.3 Pseudocode to check if a vehicle is crossing speed limit to take update or not

```
1: for each timestep in simulation
2:     get iteration of timestep
3:     for each entity in timestep
4:         if entity is equal to vehicle
5:             get all vehicles speed, id
6:             if speed of a vehicle is equal or more than X unit
7:                 at timestep vehicle is crossing speed limit for taking update
8:             else
9:                 at timestep vehicle is not crossing speed limit for taking update
```

From the simulation output, Algorithm 4.4 explains a pseudocode to check if any person rides any vehicle or just walks on the sidewalk street. For this algorithm, vehicles, persons, their location data, their speed data are needed as input. If any person and any vehicle have the same speed and location value in a particular time step, the person is riding the vehicle (See Line 8 to 10). If their speed and location data do not match for a particular time step, then the person is not riding any vehicle and just a person who is walking on the side street on that time step (See Line 12 and 13).

Algorithm 4.4 Pseudocode to check if person rides a vehicle or not

```
1: for each timestep in simulation
2:     for each entity1 in timestep
3:         if entity1 is person
4:             flag equal to flase
5:             for each entity2 in timestep
6:                 if entity2 is vehicle
7:                   get all timestep, vehicles id, speed, axis and persons id, speed, axis
8:                   if entity1 axis is equal to entity2 axis and entity1 speed is equal to entity2
9:                   speed
10:                      at timestep person id is in vehicle id
11:                      flag is true
12:            if flag is false
13:                at timestep person id is not in any vehicle
```

Algorithm 4.5 explains how to check if a vehicle is autonomous or a person-driven vehicle through a pseudocode. For this algorithm, vehicles, persons, their location data, their speed data are needed as input. The autonomous vehicles do not have any person as a driver (See Line 8 to 13).

Algorithm 4.5 Pseudocode to check if a vehicle is autonomous or person driven

```
1: for each timestep in simulation
2:     for each entity1 in timestep
3:         if entity1 is vehicle
4:             flag equal to false
5:             for each entity2 in timestep
6:                 if entity2 is person
7:                   get all timestep, vehicles id, speed, axis and persons id, speed, axis
8:                   if entity1 axis is equal to entity2 axis and entity1 speed is equal to entity2
9:                   speed
10:                      at timestep vehicle is not an autonomous vehicle and driven by person
11:                      flag is true
12:            if flag is false
13:                at timestep vehicle is an autonomous vehicle
```

It is also needed to check the driver's mood while driving the vehicle because it is not safe for a tired or sleepy driver to take an important update (Algorithm 4.6). For this algorithm, persons, their mood during driving is needed as inputs. The pseudocode also checks the person's mood (variable: acttype) for each time step to decide if the vehicle which is driven by this particular person, can take a software update or not.

Algorithm 4.6 Pseudocode to check person's mood

```
1: for each timestep in simulation
2:     get iteration of timestep
3:     for each entity in timestep
4:         if entity is equal to person
5:         get all persons id and acttype
6:         at timestep person is having this acttype
```

If the vehicle has enough charge in the battery (variables: X, Y) or not, a pseudocode explains how to check this situation in Algorithm 4.7. Here, in the pseudocode there are 2 types of car and variables for them are A (e.g. Volkswagen) and B (e.g. BMW). They have different maximum battery capacities. This is why the variables are different for the remaining charge of the battery. So the logic here, if the car A has actual battery capacity equal or more than X unit, the car has enough battery to take the update (See Line 6 to 8). For the car B, if it has actual battery capacity equal or more than Y unit, the car has enough battery to take the update (See Line 9 to 11). The algorithm also checks this situation for each time step. The inputs need here are vehicles, their type, their actual battery capacity for each time step.

Algorithm 4.7 Pseudocode to check if a vehicle has enough battery charge or not

```
1: for each timestep in simulation
2:     get iteration of timestep
3:     for each entity in timestep
4:         if entity is equal to vehicle
5:         get all vehicles id, type, actual battery capacity
6:         if vehicle type is equal to A and actual battery capacity is more than
7:         or equal to X unit
8:             at timestep vehicle has enough charge for update
9:         if vehicle type is equal to B and actual battery capacity is more than or
10:        equal to Y unit
11:            at timestep vehicle has enough charge for update
12:        else
13:            at timestep vehicle does not have enough charge to update
```

5 Simulator Selection

This chapter explains, in the beginning the simulator selection criteria, and there is a comparison among different vehicular simulators in Section 5.2. In Section 5.3, there is a brief description of SUMO; the simulator that is selected. Finally, some implemented projects in SUMO are highlighted in Section 5.4.

5.1 Simulator Selection Criteria

To enable the simulation of situations in MCPS scenarios, a basic simulation system for that MCPS is required. As we have taken vehicles as the MCPS to model and simulate situations and later also detect them, we need to select a simulator that can help to formulate and develop our concept into prototypical implementation. To select the simulator properly, it is very essential to know first what the selection criteria are.

As the motivation of this thesis is to detect situations of a vehicle, e.g., so that it can be said whether the vehicle can take any software update (e.g. break software update) now or not. So to model the situation we need to feed a map, moving vehicles, passengers, pedestrians to a simulator. So the simulator needs to be a multi-agent planning supported simulator. The simulator needs to be user friendly. So easy to use elements, straightforward tools or extensive documentation is required. To get a clear and detailed view of the simulation scenario, it is very important that the simulator gives high levels of details. Also after the simulation, the simulation result can be extracted in a known workable format (e.g. XML, YAML). In the output of the simulation result, we need the following data for our modeling and simulation part.

Vehicle Multiple vehicles with their IDs and types (e.g. hybrid vehicle, electric vehicle) are the first information to be available in the output.

State State refers to the current position (x-position and y-position of the node, angle, lane), speed and acceleration of the vehicle.

Time Only the state of the vehicles are not enough, we need them with a specific time step from the simulation duration.

Person The simulator also should have the ability to take different sorts of persons, e.g., pedestrians, passengers as input and later provide the person state (x-position and y-position of the node, angle, lane), speed, act type in the output.

Energy The maximum and actual battery capacity of each vehicle, consumed energy for each time step are also very necessary to be available in the extraction.

5.2 Comparison of Different Vehicle Simulators

There are many vehicular simulators available now. They are actually traffic simulator software such as SUMO, VISSIM, MITSIMlab, MATSim, Carla, CORSIM, Paramics, AIMSUN, SimTraffic, TRANSIMS etc. We already know what the selection criteria to choose the right simulator are. There are 3 different models in simulators: microscopic, mesoscopic and macroscopic. Vehicle behavior and interaction are modeled individually with the reflection of reality in microscopic simulation, whereas the macroscopic approaches focus on the complete road flow by integrating more microscopic models and mesoscopic simulation is based on a greatly simplified model[1]. The simulators can also be discrete which means variables change at regular intervals of time and continuous which means variables change continuously [SEE16]. There are open-source and commercial simulators with the visualization of 2D and 3D. In the following table 5.1, there is a detailed comparison among available popular vehicular simulators.

According to the simulation criteria from [SEE16], we have decided to use SUMO as the situation simulation system, because this meets most of our selection criteria. It can provide the option of using multiple and different sorts of vehicles. It has the option to add persons as passengers and pedestrians. It also provides the option to export the energy situation of a vehicle. After the simulation, from SUMO it is possible to extract the simulation output as XML document with time steps, vehicles state, persons state, energy level and all other details. SUMO is an open-source, microscopic, easy to use and well-documented traffic simulator too.

[1] http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/use-cases/mesoscopic-and-hybrid-simulation/

	Simulators	AIMSUN	ARCHISIM	CORSIM	MATSim	MITSIMLab	Paramics	SimTraffic	SUMO	TRANSIMS	TransModeler	VISSIM
Model	Microscopic	x	x	x	x	x	x	x	x	x	x	x
	Mesoscopic	x								x	x	
	Macroscopic	x									x	
Category	Open-source				x	x			x	x		
	Commercial	x	x	x			x	x			x	x
System	Discrete		x	x			x			x		
	Continuous	x			x				x			x
Visualization	Two-dimensional	x	x	x	x	x	x	x	x	x	x	x
	Three-dimensional	x		x			x	x	x		x	x
Infrastructure Difficulty	Easy							x				x
	Medium											
	Difficult	x	x						x			
Infrastructure Flexibility	Flexible	x					x		x			x
	Limited				x	x		x			x	
	Very Limited		x	x						x		
Vehicles and Pedestrians	Type								x			
	Dimension											
	Priority	x		x			x	x	x			x
	Pedestrian	x		x			x	x	x		x	x
	Other vehicles	x					x		x			x
Scope Area	City		x	x	x	x	x	x	x	x		x
	Region	x		x	x		x	x	x	x	x	x
	Country	x							x	x	x	
Energy Consumption	Yes	x		x					x	x		
	No		x		x	x						
Import Maps	Yes	x			x				x		x	x
	Partially											
	No		x	x		x	x	x				
Output	Yes	x		x	x	x			x	x	x	x
	No		x									

Table 5.1: Comparison among different available traffic simulators [SEE16] [SMC04]

5.3 Simulation of Urban Mobility (SUMO) - A Brief Description

SUMO is a free and open-sourced traffic simulator. It is available since 2001 to simulate a traffic road network of the size of a city[2]. It is developed by the Institute of Transportation Systems of The German Aerospace Center (DLR). Vehicles including public transports, human being need to have a described route and departure time. As the traffic flow in this simulator is microscopic, every vehicle in this simulator needs to be modeled individually. Figure 5.1 explains the simplest simulation process of SUMO.

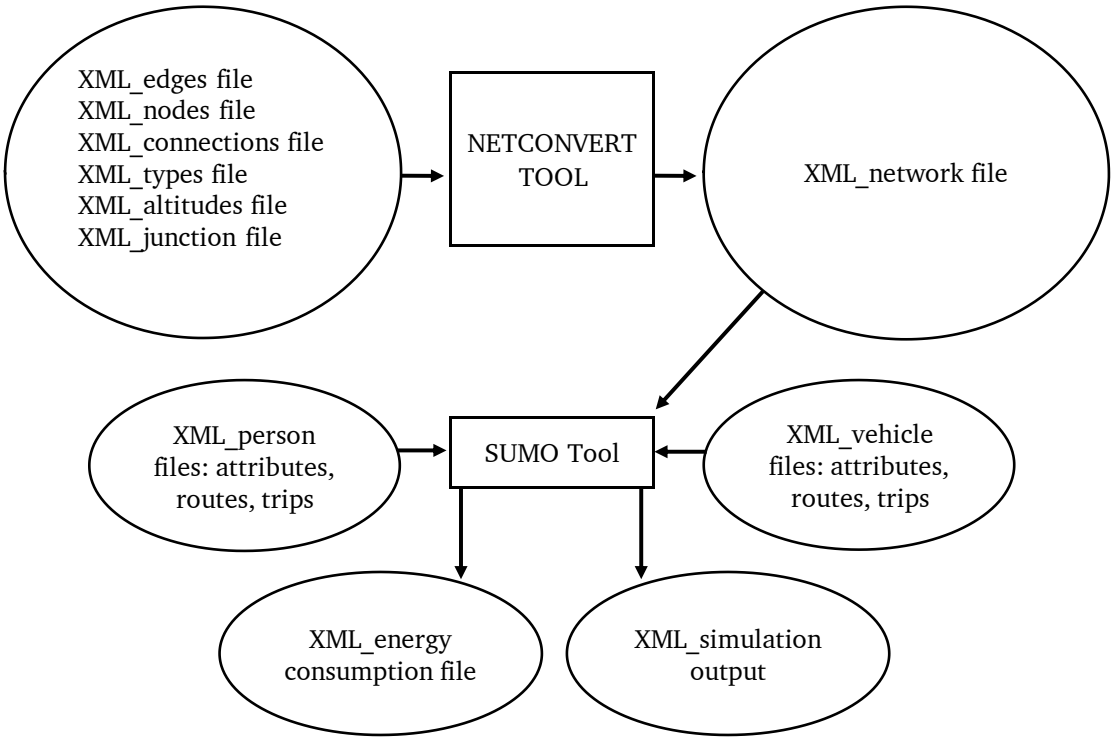

Figure 5.1: Simulation process with SUMO [MSAN11]

With the increasing time step which can be 1 second, the state of these objects change. SUMO is being used by the vehicle-to-everything (V2X) community, so that they can get realistic vehicle traces and evaluate other applications. The SUMO package contains the following applications that are also needed for our work[2]:
SUMO: command line simulation;
GUISIM: simulation with a graphical user interface;
NETCONVERT: network importer;
NETGEN: abstract networks generator;
DUAROUTER: routes generator based on a dynamic user assignment;

[2]https://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/

NETEDIT: NETEDIT is visual editor for street networks, traffic lights, detectors, and further network elements.

We have used the latest version available in August 2018 (version 0.32.0). The version contains the following features[3]:

- Supports different vehicles types
- Vehicles can change the lanes as it supports multi-lane streets
- Simulator can take inputs from XML file, e.g., input the XML-data containing nodes, edges and edge types to create a simple map
- Collision free vehicle movement
- A XML-raw-output which contains information about the state of the simulation details for every time step (Network-wide, edge-based, vehicle and detector-based outputs)
- Space-continuous and time-discrete vehicle movement
- A fast openGL graphical user interface
- Different Dynamic User Assignment algorithms
- Only standard C++ and portable libraries are used
- Fast execution speed (up to 100.000 vehicle updates/s on a 1GHz machine)
- Manages networks with several 10.000 edges (streets)
- Supports person-based inter-modal trips

5.4 Projects in SUMO

These projects show that SUMO is capable of doing situational based simulation for vehicles. Semrau et. al [SERF17] enable a situation adaptive driver behavior for lane changing merging processes in SUMO. At first, to model this in SUMO, driving behavior is designed by taking the influential critical traffic situations. Lane changing behavior also depends on the level of driving experience. Therefore, to implement it in SUMO, the emotional memory is also taken into consideration to be integrated. Some situational conditions are created for implementation. For example, a model for lateral distance keeping of vehicles was added. This means that the vehicles have to maintain a minimum gap between them to continue driving in the current lane. The result of the model is validated by using the real world data.

[3] http://sumo.dlr.de/userdoc/Sumo_at_a_Glance.html#Features

Cottignies et al. [CDNP17] aim to create a complete real-time simulation of the urban environment. The initial integration is done by using SUMO and rFpro[4]. The real-time simulation takes inputs from driver or vehicle's sensors, variables like sound, touch, motion, sight. Then based on the decision making situational algorithms driver or the autonomous vehicles take decisions. The algorithms help the drivers or the autonomous vehicles to control the vehicles by using the vehicular actuators. The result from the action is fed back to the simulators so that the vehicles can cope up for the next situation in real-time.

[4]http://www.rfpro.com

6 Simulator Environment and Prototypical Implementation

The following chapter outlines the prototypical implementation of this thesis. In section 6.1, we present the simulator environment and simulation scenarios. The next section describes the implementation for situation recognition.

6.1 Simulator Environment and Situation Modeling

INPUT	SIMULATION	OUTPUT
Map Data	Map to Network Conversion	Each Timestep +
Vehicle Details / Data	Vehicle Routes and Trips	Person Ride Details with Axis, Mood
Passenger Details / Data	Passengers Routes and Trips	Pedestrian Routes Detail
	Pedestrians Routes and Trips	Network Details
	Vehicle Energy Simulation	Vehicle Simulaton Result with Energy Details, Axis & Angles

Figure 6.1: The inputs, simulations and outputs in SUMO

This section reflects the prototypical implementation of the *Simulation System* from Section 4.1. It is explained here, how we have built the simulation scenarios in the SUMO simulator step by step. In Figure 6.1 the inputs, simulation processes and outputs for SUMO are briefly mentioned. The installation process of SUMO based on the operating system is available in[1].

The First Step of this prototypical implementation is to create a SUMO network file. The network file contains traffic related part of a map (e.g. traffic light logics referenced by junctions, junctions with their right way of regulation)[2]. The streets are a collection of lanes (edges) and junctions are the connection between the lanes (nodes). OpenStreetMap (OSM)[3] can be useful to select, download, and prepare a map where vehicles and persons can drive or walk in. The OSM file is simple to use as the map for our simulation with SUMO. OSM file should have some necessary qualities available for the simulation,

[1] http://sumo.dlr.de/wiki/Installing
[2] http://sumo.dlr.de/wiki/Networks/SUMO_Road_Networks
[3] https://www.openstreetmap.org/

such as, the type of a street, junctions, speed limit, connections, the right way of rule, marked one-way street, traffic light logic information, and the correct number of lanes to avoid unrealistic traffic congestions in the simulation[4]. We have selected the map from Stuttgart Stadmitte and Hauptbahnhof area. So the map of the are is downloaded from OSM. Then the OSM normal files are converted to the new modified OSM files (with elevation) (using osmosis[5] tool) with additional srtm plugin. The command is in Listing 6.1.

```
1  osmosis --read-xml map.osm --write-srtm tagName=ele --write-xml Stad.osm
```

Listing 6.1: Snippet of the command to convert OSM normal files to new modified OSM files

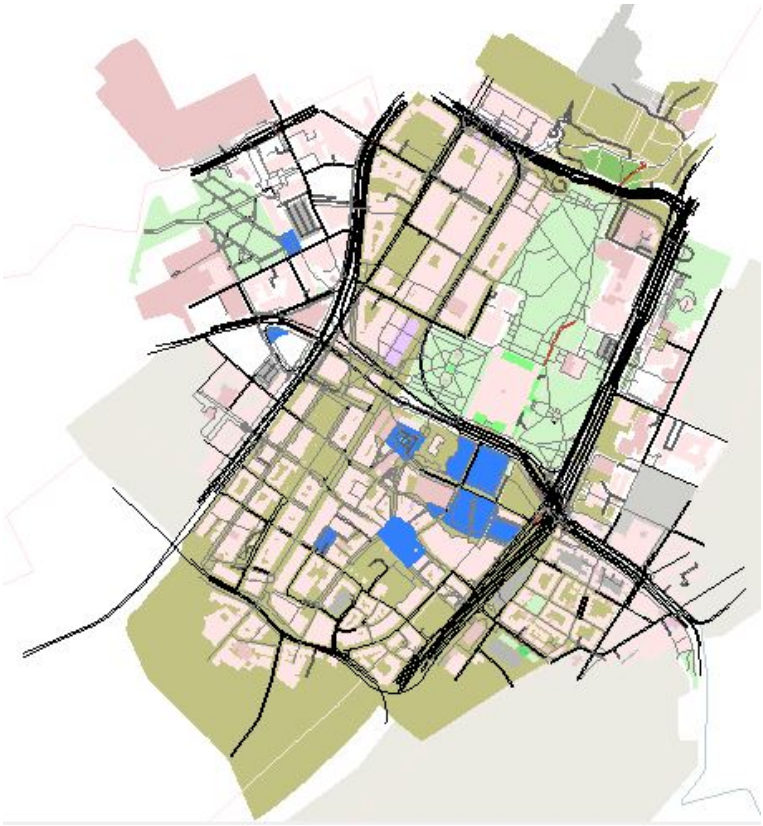

Figure 6.2: The map in SUMO-GUI, in the standard format

Here, Stad.osm is the downloaded OSM file. Now to create and import this OSM data into SUMO NETCONVERT application is used. NETCONVERT[6] extracts the simulation related information from OSM file to do the conversion and import it into the SUMO network file.

[4] http://sumo.dlr.de/wiki/Tutorials/Import_from_OpenStreetMap
[5] Osmosis is a command line Java application for processing Open Street Map data (Source and installation method: https://github.com/openstreetmap/osmosis).
[6] http://sumo.dlr.de/wiki/NETCONVERT

This network file is a simple human readable XML file. The visualization of the network area is visible in Figure 6.2.

The following command (See Listing 6.2) generates the network file (Stad.net.xml) using the netconvert file. Here, the input is the OSM file named 'Stad.osm'. Network file contains information about the edges, lanes, junctions, and connections of the selected area.

```
1  netconvert --osm-files Stad.osm -o Stad.net.xml --junctions.join --roundabouts.guess
       --osm.elevation --tls.guess
```

Listing 6.2: Snippet of command to convert OSM file to network file

All the command need to execute "cmd.exe"[7]. The network file is using Cartesian, metric coordinates. Therefore, x=0 is at the leftmost node in the network and y=0 is at the bottom node. Now we will describe lanes, junctions, and connections available in the network file we created for simulation purpose.

Edges Edges contain the definitions of lanes they consist of[8] (See Listing 6.3). An edge can be an internal edge. Internal edges are available in an intersection, so they connect incoming and outgoing normal edges. In case an edge is an internal edge, there will be an attribute named *function* next to edge id. The minimum edge length is 0.1m. Lanes can have an attribute like allow. Suppose, a sidewalk is a lane which allows only the SUMO 'vclass' pedestrians. In the following, it is shown how a 2D or 3D edge with lanes look like.

```
1  <edge id="<ID>" from="<FROM_NODE_ID>" to="<TO_NODE_ID>" priority="<PRIORITY>">
2       <lane id="<ID>_0" index="0" speed="<SPEED>" length="<LENGTH>" shape="1246.24,357.64
            1251.43,354.70"/>
3  </edge>
```

Listing 6.3: Snippet of edge with lane attributes

Now in Table 6.1, we describe the different attributes of an edge in SUMO. Edges usually have attributes like id, from, to and priority. As edges are made of lanes, they also contain information about lanes. Lanes usually have attributes like id, index, speed, length and shape. Table 6.1 explains attributes meaning and their data type of lanes[8].

[7] for example, on Windows, to start "cmd.exe" follow Start->Execute->cmd.exe. Suppose, The files of SUMO release locate at **C: > sumo-0.32.0**, then every time before any command it is necessary to write the full path

[8] http://sumo.dlr.de/wiki/Networks/SUMO_Road_Networks

Name	Type	Description
id	string	The id of the lane
index	running number (unsigned int)	A running number, starting with zero at the right-most lane
speed	float	The maximum speed allowed on this lane [m/s]
length	float	The length of this lane [m]
shape	position vector	The geometry of the lane, given by a poly-line that describes the lane's center line
function	"internal"	Always "internal" for an internal edge

Table 6.1: Table with the lanes attribute name, type and description

Junctions Different streams cross in a junction of a map[9]. Junctions have attributes like id, type, position details; an example is in Listing 6.4. In the listing, the "request"s describe with the which (index) streams have a higher priority and are in conflict. "Response" expresses the higher priority stream and "foes" the conflicted one.

```
1  <junction id="<ID>" type="<JUNCTION_TYPE>" x="<X-POSITION>" y="<Y POSITION>"
       incLanes="<INCOMING_LANES>" intLanes="<INTERNAL_LANES>" shape="<SHAPE>">
2              <request index="0" response="0000" foes="0100" cont="0"/>
3              <request index="1" response="0000" foes="1100" cont="0"/>
4  </junction>
```

Listing 6.4: Snippet of junction with attributes

Table 6.2 describes what the attributes (id, x, y, incLanes, intLanes, shape) in junctions mean (column "Description") and what data type they are (column "Type")[9].

Name	Type	Description
id	string	The id of the junction
x	x-position (real)	The x-coordinate of the intersection
y	y-position (real)	The y-coordinate of the intersection
incLanes	id list	The ids of the lanes that end at the intersection
intLanes	id list	The IDs of the lanes within the intersection
shape	position list	A polygon describing the road boundaries of the intersection

Table 6.2: Table with the junctions attribute name, type and description

[9] http://sumo.dlr.de/wiki/Networks/SUMO_Road_Networks

Connections Then come the connections. Connections provide the information of which outgoing lanes can be reached from an incoming lane[10]. Example of connections file is available in Listing 6.5.

```
1  <connection from="<FROM_EDGE_ID>" to="<TO_EDGE_ID>" fromLane="<FROM_LANE_INDEX>"
       toLane="<TO_LANE_INDEX>" dir="r" state="o"/>
```

Listing 6.5: Snippet of connections with attributes

The attributes of the connections (from, to, fromLane, toLane, intLanes, dir, state) are listed in Table 6.3. The types of the attributes and their meanings are enlisted with their names in the table[10].

Name	Type	Description
from	edge id (string)	The ID of the incoming edge at which the connection begins
to	edge id (string)	The ID of the outgoing edge at which the connection ends
fromLane	index (unsigned int)	The lane of the incoming edge at which the connection begins
toLane	index (unsigned int)	The lane of the outgoing edge at which the connection ends
intLanes	id list	The IDs of the lanes within the intersection
dir	enum ("s" = straight, "t" = turn, "l" = left, "r" = right, "L" = partially left, R = partially right, "invalid" = no direction)	The direction of the connection
state	enum ("-" = dead end, "=" = equal, "m" = minor link, "M" = major link)	The state of the connection

Table 6.3: Table with the connections attribute name, type and description

The Second Step is to import polygons from OSM data and produce OSM polygon file. POLYCONVERT[11] is able to do the import of geometrical shapes like polygons and convert them to a representation for visualization in SUMO-GUI. To do the conversion, the used command is listed in Listing 6.6.

```
1  polyconvert --net-file Stad.net.xml --osm-files Stad.osm --type-file typemap.xml -o
       Stad.poly.xml
```

Listing 6.6: Snippet of the command to import polygons from OSM data

[10] http://sumo.dlr.de/wiki/Networks/SUMO_Road_Networks
[11] http://sumo.dlr.de/wiki/POLYCONVERT

The Third Step of creating the simulation environment in SUMO is to generate the vehicular traffic demand. A vehicle in SUMO needs a vehicle type that describes the vehicle's physical properties[12]. Then it is necessary to create a route for the individual vehicle. Listing 6.7 describes the vehicle with attributes.

```
<additional>
  <vType id="Vehicle_Type_id" lenght="Length_Value" maxSpeed="Max_Speed_Value"
       sigma="float" minGap="float" color="color" probability="float">
    <param key="maximumBatteryCapacity"value="float"/>
    <param key="maximumPower" value="float"/>
    <param key="actualBatteryCapacity"
    value="float"/>
    ........
  </vType>
</additional>
```

Listing 6.7: Snippet of vehicle types with attributes (ecar.add.xml file)

Here sigma is driver imperfection (between 0 and 1), and the probability is the chance of emitting a vehicle each second. Vehicles can have more attributes as an internal moment of inertia, vehicle mass, etc. When SUMO installation file is downloaded, there is a file available named 'randomTrips.py' in the folder 'tools'. This code helps to generate routes for different objects (e.g., vehicles, persons) within the simulator network. The command to generate routes and trips for vehicles using the 'randomTrips.py' file is listed in Listing 6.8 [12].

```
python randomTrips.py --n Stad.net.xml -r ecar.rou.xml -t "type=\"ElectricalVehicle\"
    departSpeed=\"max\" departLane=\"best\"" -c passenger --additional-files
    ecar.add.xml -p 1.4 -e 1000 -l
```

Listing 6.8: Snippet of the command which produces routes and trips for vehicles in the simulator SUMO

This gives the routes file 'ecar.rou.xml' and trips file 'trips.trips' as output. The example of a routes file for the vehicles[12] is presented in Listing 6.9. The routes file contains all the vehicles information, e.g., id, type, depart, departLane, departSpeed and route edges, etc.

```
<routes>
  <vehicle id="0" type="ElectricalSedan" depart="0.00" departLane="best"
       departSpeed="max">
      <route edges="25793413#0 -25793413#0 24041046#0"/>
</vehicle>
 <vehicle id="1" type="ElectricalBus" depart="1.40" departLane="best" departSpeed="max">
      <route edges="24807643 25802534#0 25802534#1 408711843"/>
</vehicle>
    ........
</routes>
```

Listing 6.9: Snippet of route file of vehicles

[12]http://sumo.dlr.de/wiki/Definition_of_Vehicles,_Vehicle_Types,_and_Routes

Example of the trips[13] are available in the trips.trips.xml file and the following Listing 6.10 presents the general format of the file. Trips have their id, depart, from, to, type, departSpeed and departLanes attributes.

```
1 <routes>
2    <trip id="0" depart="0.00" from="25793413#0" to="9666308" type="ElectricalSedan"
         departSpeed="max" departLane="best"/>
3    <trip id="1" depart="1.40" from="24807643" to="-145313671#2" type="ElectricalBus"
         departSpeed="max" departLane="best"/>
4 </routes>
```

Listing 6.10: Snippet of trips file of vehicles

The meanings and types of the attributes from the vehicles routes and trips file are combined together in a table. They are listed in the following Table 6.4[13].

Name	Type	Description
vehicle id	id (string)	The name of the vehicle
type	id	The id of the vehicle type to use for this vehicle.
depart	float	The time step at which the vehicle shall enter the network
departLane	int or string (>=0, "random", "free", "allowed", "best", "first")	The lane on which the vehicle shall be inserted
departSpeed	float(m/s) or string (>=0, "random", "max")	The speed with which the vehicle shall enter the network
trip id	id (string)	The name of the trip
edges	id list	The edges the vehicle shall drive along, given as their ids, separated using spaces

Table 6.4: Table with the name, type and description of the vehicles route and trip attributes

The Fourth Step is to generate the pedestrians and passengers flow. Pedestrians are persons that walk[14]. They walk in the edges 'sidewalk' which are the rightmost lanes in simulator network. Whereas a passenger rides or drives a vehicle. To add the passenger in the simulator it is necessary first to create a file like the following Listing 6.11.

```
1 <additional>
2 <vType id="Vehicle_Type_id" vClass="passenger"/>
3 </additional>
```

Listing 6.11: Snippet of passengers file

[13]http://sumo.dlr.de/wiki/Definition_of_Vehicles,_Vehicle_Types,_and_Routes
[14]http://sumo.dlr.de/wiki/Simulation/Pedestrians

Then to create the routes and trips for persons in the generated map, the following command in Listing 6.12 is used[15]:

```
python randomTrips.py --n stad.net.xml -r passengers.rou.xml -o passenger.trips.xml
    --trip-attributes "modes=\"car\"" -c persontrip --persontrips -p 1.4 -e 1000 -l
```

Listing 6.12: Snippet of the command that is used to generate the routes and trips for the persons in SUMO

The generated routes file contains the information about the person who rides a vehicle and person who walks. The file also contains information about from where the ride begins and where it ends. Listing 6.13 gives an example of the generated routes file[15].

```
<routes>
   <person id="0" depart="0.00">
      <walk edges="23553288#1 23553288#0 370301707#1 "/>
   </person>
   <person id="1" depart="1.40">
      <ride from="3996880" to="10075258#1" lines="1_0"/>
   </person>
</routes>
```

Listing 6.13: Snippet of routes of pedestrians and passengers

The generated trips file provides the trip details of persons[15], e.g., from which point to which point the particular person is having the trip and the modes of the trips, see the following Listing 6.14.

```
<routes>
   <person id="0" depart="0.00">
      <personTrip from="23553288#1" to="90724434#2" modes="vehicle"/>
   </person>
   <person id="1" depart="1.40">
      <personTrip from="3996880" to="10075258#1" modes="vehicle"/>
   </person>
</routes>
```

Listing 6.14: Snippet of trips of persons

the Fifth Step in this last step, generating the output file with battery information of vehicles is needed as we have described in the *Execute Simulation (Step 2)* of Figure 4.1 in chapter 4. SUMO generates the output files in XML format by default[16]. We take the Floating Car Data (fcd) output with battery usage. FCD output provides name, position, angle, type for every vehicle and person. So, a configuration file named 'Stad.sumo.cfg' is created to combine everything together and run the simulation. We run the simulation in GUI. The configuration file example is presented in Listing 6.15. The file needs the generated net file and all the routes files as input. Additional files value needs the files

[15] http://sumo.dlr.de/wiki/Simulation/Pedestrians
[16] http://sumo.dlr.de/wiki/Simulation/Output

like "ecar.add.xml" and "stad.poly.xml". The simulation time is defined within the "time" attributes. The simulation will run for 1000 time-step as defines here and the interval between the time-steps will be 0.1.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/sumoConfiguration.xsd">
3
4     <input>
5        <net-file value="stad.net.xml"/>
6        <route-files value="ecar.rou.xml,passengers.rou.xml"/>
7     </input>
8
9        <additional-files>
10       <additional-files value="ecar.add.xml,stad.poly.xml"/>
11       </additional-files>
12
13    <time>
14       <begin value="0.1"/>
15       <end value="1000"/>
16       <step-length value="0.1"/>
17    </time>
18
19    <output>
20    <fcd-output value="SUMO_output.output.xml"/>
21       <battery-output value="battery.xml"/>
22       <battery-output.precision value="4"/>
23    <device.battery.probability value="1"/>
24 </output>
25
26 </configuration>
```

Listing 6.15: Snippet of configuration file for outputs

'SUMO_output.output.xml' file provides the FCD output and 'battery.xml' contains the energy consumption details. Later, they are mapped and merged together so that it is easy to do the situation modeling and detection.

As this version of SUMO can not provide user moods (e.g., sleepy, attentive, not attentive, tired) details, for our situation modeling purpose, we add them very randomly with persons who drive a vehicle. Moreover, to keep it simple, we have added a few Road-Side Units (RSU)s and home address (See Line 4-6) to calculate if the vehicle has enough connectivity to the internet (See Listing 6.16). This follows the *Filter Data* step of the concept chapter (See Figure 4.1).

```
<fcd-export xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/fcd_file.xsd">

       <timestep time="0.00">
             <RSU id="1" x="851.00" y="203.00" edge="24820388#0_3"/>
             <RSU id="2" x="1012.00" y="1008.00" edge="3933624#4_0"/>
             <Home id="1" x="699.31" y="1314.01"/>
       </timestep>
    <timestep time="1.40">
       <vehicle id="1" x="1358.63" y="949.82" angle="29.42" type="ElectricalSedan"
            speed="13.89" pos="5.10" lane="24807643_1" slope="0.00"
            energyConsumed="0.0000" actualBatteryCapacity="1000.0000"
            maximumBatteryCapacity="2000.0000" chargingStationId="NULL"
            energyCharged="0.0000" energyChargedInTransit="0.0000"
            energyChargedStopped="0.0000" acceleration="0.0000" timeStopped="0"/>
       <vehicle id="1_0" x="1403.27" y="1115.85" angle="201.94" type="ElectricalCar"
            speed="0.00" pos="5.10" lane="3996880_0" slope="0.00" energyConsumed="0.0000"
            actualBatteryCapacity="17500.0000" maximumBatteryCapacity="35000.0000"
            chargingStationId="NULL" energyCharged="0.0000"
            energyChargedInTransit="0.0000" energyChargedStopped="0.0000"
            acceleration="0.0000" timeStopped="0"/>
       <person id="1" acttype="Attentive" x="1403.27" y="1115.85" angle="201.94"
            speed="0.00" pos="5.10" edge="3996880" slope="0.00"/>
    </timestep>
</fcd-export>
```

Listing 6.16: Snippet of the output file with RSU, home, drivers mood and all other available details from SUMO output

6.2 Situation Recognition

This section of this chapter shows the prototypical implementation of the Section 4.4 from the concept chapter. Moreover, it shows the situations simulation and recognition in the python code snippets based on the algorithms of Section 4.5. The whole situation modeling, simulation and detection processes are shown step by step in Figure 6.3.

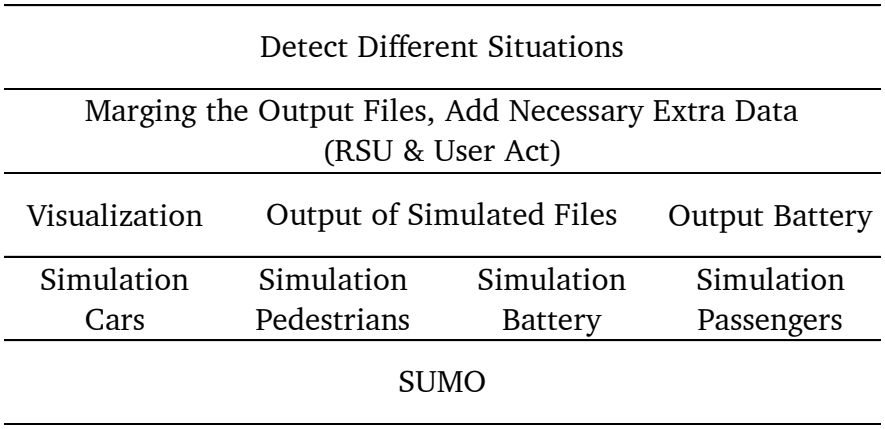

Figure 6.3: The whole situation modeling, simulation and detection

As we want to check if a vehicle is capable of taking the software update for brake at a certain time step or not, we have to detect some situation, such as, if the vehicle is moving or parked somewhere, if the vehicle is running within the certain speed limit, if the vehicle is autonomous or a person driven vehicle, if the vehicle has network connectivity with RSU or home network, even if the vehicle has internet connectivity what sort of signal strength it has (strong, medium or weak), if any person rides the vehicle, if the vehicle has enough battery power, etc. (See Listing 6.17).

```
1  In a particular time step:
2  Test if the vehicle is moving or not;
3  Test if the vehicle is crossing the speed limit or not;
4  Test if the vehicle has strong, medium or weak internet connection;\\
5  Test if the vehicle is autonomous or person driven;
6  Test if the vehicle has any riders or not;
7  Test if the vehicle's driver is not sleepy or too tired;
8  Test if the vehicle has enough battery charge;
```

Listing 6.17: Test if the vehicle can take available brake software update in a particular time period

So at first, we want to recognize from the simulation output if the vehicle is running, stopped on the road or parked somewhere (e.g., at home, parking area) (See Listing 6.18). To recognize this situation the logic is, if the speed of the vehicle is 0 then it is stopped or parked somewhere, and if the vehicle has the speed limit more than 0, the vehicle is running on a road on that particular time step (See Line 15-18). The inputs we need for this detection are vehicle id, time step, vehicle's speed.

```
1  import xml.etree.ElementTree as ET
2  tree = ET.parse('SUMO_output.output - All Finished.xml')
3  root = tree.getroot()
4
5  for timestep in root.iter('timestep'):
6    time = timestep.get('time')
7    for child in timestep:
8      if (child.tag == 'vehicle'):
9        speed = (float) (child.attrib.get('speed'))
10
11       print ("time: ", time)
12       id = (str) (child.attrib.get ('id'))
13       print ("id: ", child.attrib.get('id'))
14       print ("speed: ", child.attrib.get('speed'))
15       if (speed > 0.00):
16         print ('At time ', time, "vehicle ", id, "is moving")
17       else:
18         print ('At time ', time, "vehicle ", id, " is not moving")
```

Listing 6.18: Implemented Python code snippet to recognize if a vehicle is moving or not moving

To recognize from the simulation output if the vehicle is crossing the maximum speed limit to take a brake software update. Suppose, the maximum speed limit for a car to take the update is 13m/s. So if any vehicle is crossing this limit in a time step, it is not allowed to take the update. The input we need for this simulation and detection are id, time step, vehicle's speed. The python code snippet from Listing 6.19 shows the logical part of the speed limit recognition (See Line 8-11) in the following (as the output file importing part is always same, we skip writing that part).

```
1  for timestep in root.iter('timestep'):
2    time = timestep.get('time')
3    for child in timestep:
4      if (child.tag == 'vehicle'):
5        speed = (float) (child.attrib.get('speed'))
6        id = (str) (child.attrib.get ('id'))
7
8        if (speed >= 13.00):
9          print ('At time ', time, "vehicle ", id, "is crossing speed limit")
10       else:
11         print ('At time ', time, "vehicle ", id, " is not crossing speed limit")
```

Listing 6.19: Implemented Python code snippet to recognize if a vehicle is crossing the speed limit or not

If the vehicle does not have internet connectivity, then it is not possible for the vehicle to download and install the updated software version (See Listing 6.20).

```
for timestep in root.iter('timestep'):
  time = timestep.get('time')
  for child in timestep:

   if (child.tag == 'RSU'):
      rsu_id = (str) (child.attrib.get ('id'))

   if (child.tag == 'Home'):
      home_id = (str) (child.attrib.get ('id'))

   if (child.tag == 'vehicle'):
      vehicleypoint = (float) (child.attrib.get('y'))
      vehicle_id = (str) (child.attrib.get ('id'))

      if (vehicleypoint > 53.00 and vehicleypoint <= 353.00):
        print ('At time ', time, "vehicle ", vehicle_id, "is connected to RSU1 and the
            connection is strong (+++)")
      elif (vehicleypoint >= 353.01 and vehicleypoint <= 503.00):
        print ('At time ', time, "vehicle ", vehicle_id, "is connected to RSU1 and the
            connection is medium (++)")
      elif (vehicleypoint >= 503.01 and vehicleypoint <= 650.00):
        print ('At time ', time, "vehicle ", vehicle_id, "is connected to RSU1 and the
            connection is weak (+)")
      elif (vehicleypoint >= 858.01 and vehicleypoint <= 1158.00):
        print ('At time ', time, "vehicle ", vehicle_id, "is connected to RSU2 and the
            connection is strong (+++)")
      elif (vehicleypoint >= 708.01 and vehicleypoint <= 858.00):
        print ('At time ', time, "vehicle ", vehicle_id, "is connected to RSU2 and the
            connection is medium (++)")
      elif (vehicleypoint >= 1158.01 and vehicleypoint <= 1310.00):
        print ('At time ', time, "vehicle ", vehicle_id, "is connected to RSU2 and the
            connection is medium (++)")
      elif (vehicleypoint >= 650.01 and vehicleypoint <= 708.00):
        print ('At time ', time, "vehicle ", vehicle_id, "is connected to RSU2 and the
            connection is weak (+)")
      elif (vehicleypoint >= 1320.01 and vehicleypoint <= 1460.00):
        print ('At time ', time, "vehicle ", vehicle_id, "is connected to RSU1 and the
            connection is weak (+)")
      elif (vehicleypoint >= 1310.01 and vehicleypoint <= 1320.00):
        print ('At time ', time, "vehicle ", vehicle_id, "is connected to Home1 WiFi")
      else:
        print ('At time ', time, "vehicle ", vehicle_id, "is not connected")
```

Listing 6.20: Implemented Python code snippet to recognize if a vehicle has connection with home RSU or not and if there is connection what the level of connection strength is

A vehicle can be connected to RSU to take an update and then share it with other vehicles. If the vehicle is parked at home or office then it is also easier for the vehicles to take update as they get a strong WiFi signal from the home or office network. Suppose in the

prototypical implementation, to check if the vehicle is at home, it is checked whether the coordinate distance difference of the home and vehicle is within $10m^2$. To check if the vehicle has a connection with any RSU, the distance difference is measured. If the distance difference is within approximately $150m^2$ then the vehicle is strongly (+++) connected to that RSU, if within approximately $400m^2$ then the vehicle has a medium level (++) of connection and if within approximately $800m^2$ then the vehicle has a weak (+) connection. If this distance range does not match, the vehicle has no connection to any RSU or home network. The RSU and home have a static position, therefore, static axis point (e.g., y-axis of RSU_1 = 203.00, RSU_2 = 1008.00, Home = 1314.01 in our taken map). In Line 15 of Listing 6.20, we set the connection between vehicles and RSU1 to strong because in this condition, it is checked whether the distance between the vehicles and RSU1 is $150m^2$ (the vehicles point are set to 53.00 and 353.00 as the axis point of RSU_1 is 203.00). In Line 21 of the code snippet, we set the connection between vehicles and RSU2 to strong because in this condition, it is checked whether the distance between the vehicles and RSU2 is $150m^2$ (the vehicles point are set to 858.01. and 1158.00 as the axis point of RSU_2 is 1008.00). For this situation modeling and recognition we need the axis information of vehicles and homes, time step, id of the vehicles and homes (See Listing 6.20).

From the SUMO output, to recognize if any person rides a car or just walks on the sidewalk street (See Listing 6.21), we need to check the vehicles and persons axis value and speed. When the axis value and speed of the persons and vehicles are equal (See Line 10 and 11), the person is actually riding the car; otherwise the person is walking on the side walk street. With the same logic, it can also be checked if the vehicle is an autonomous vehicle or person driven.
To check the person's interaction with the vehicle, all the persons need to be considered first whereas to recognize if the vehicle is autonomous all the vehicles need to be considered first.

```
1  for timestep in root.iter('timestep'):
2        for child1 in timestep:
3           if child1.tag in ['personride', 'person']: #considering all persons at one
                timestep
4              flag = False
5              for child2 in timestep:
6                 if child2.tag == 'vehicle': #considering all vehicles in the same timestep
7                    tm = timestep.get('time')
8                    pr = child1.get('id')
9                    vh = child2.get('id')
10                   if child2.get('x') == child1.get('x') and child2.get('speed') ==
                         child1.get('speed'):
11                      print('At time {}, person {} is in vehicle {}.'.format(tm, pr, vh))
12                      flag = True
13             if flag is False:
14                print('At time {}, person {} is not in any vehicle.'.format(tm, pr))
```

Listing 6.21: Implemented Python code snippet to recognize if person rides a vehicle or not

To recognize if the vehicle has enough battery or not, we need the maximum battery capacity, consumed energy of the vehicle. Suppose in our prototypical implementation, we have 2 types of vehicles: electric sedan and electric car. The maximum battery capacity of the electric sedan and electric car respectively are 2000Wh and 35000Wh. So if the electric sedan has an actual remaining battery capacity of 500Wh or more than that in any time step, it is capable of taking update (See Line 14 and 15). In case of an electric car, it is equal or more than 15000Wh (See Line 16 and 17). To model and simulate this situation we need the time step, vehicle id, vehicle type, actual battery capacity attributes (See Listing 6.22).

```
1  for timestep in root.iter('timestep'):
2    time = timestep.get('time')
3    for child in timestep:
4      if (child.tag == 'vehicle'):
5        vehicle_id = (str) (child.attrib.get ('id'))
6        vehicle_type = (str) (child.attrib.get ('type'))
7        vehicle_actualBatteryCapacity = (float) (child.attrib.get('actualBatteryCapacity'))
8
9        print ("time: ", time)
10       print ("id: ", child.attrib.get('id'))
11       print ("type: ", child.attrib.get('type'))
12       print ("actualBatteryCapacity: ", child.attrib.get('actualBatteryCapacity'))
13
14       if vehicle_type == "ElectricalSedan" and vehicle_actualBatteryCapacity >= 500:
15         print ('At time ', time, "vehicle ", vehicle_id, "has",
               vehicle_actualBatteryCapacity, 'W charge. So, it has enough charge for
               update')
16       elif vehicle_type == "ElectricalCar" and vehicle_actualBatteryCapacity >= 15000:
17         print ('At time ', time, "vehicle ", vehicle_id, "has",
               vehicle_actualBatteryCapacity, 'W charge. So, it has enough charge for
               update')
18       else:
19         print ('At time ', time, "vehicle ", vehicle_id, "has",
               vehicle_actualBatteryCapacity, 'W charge. So, it does not have enough
               charge for update')
```

Listing 6.22: Implemented Python code snippet to recognize if a vehicle has enough battery charge or not

To check the person's mood at any time step, it is enough to check the person's 'acttype' from the XML output file with their id (See Listing 6.23).

```
for timestep in root.iter('timestep'):
  time = timestep.get('time')
  for child in timestep:
    if (child.tag == 'person'):
      acttype = (str) (child.attrib.get('acttype'))
      print ("time: ", time)
      id = (str) (child.attrib.get ('id'))
      print ("id: ", child.attrib.get('id'))
      print ("acttype: ", child.attrib.get('acttype'))
```

Listing 6.23: Implemented Python code snippet to recognize person's mood

7 Conclusion and Future Work

The following chapter summarizes the research, concept, and implementation of the system. Furthermore, it discusses the conclusions with further scope for the future work.

Within this thesis, initially some fundamentals like CPS, MCPS, their generic architecture, IoT, context, situation, Situation-Aware Systems, and Workflows, the method of modeling and simulation are introduced. These fundamentals lead to the related work of them. Afterwards, the situation recognition methods from different research projects are explained and analyzed. Moreover, to describe the modeling in the CPSs, some important related work are highlighted. As we choose the application of Software Update in vehicular systems as our MCPS to model and simulate situations, CPS modeling of vehicles is also illustrated.

The primary contribution of this thesis is the development of the concepts for modeling and simulation of situations in MCPS. It starts with a system overview which contains two components; *Simulation System* and *Situation Recognition System*. The system has five steps all together: defining the scenarios, executing the simulation is a selected simulator, filtering the simulation output, defining the situations through pseudocodes and finally recognizing the modeled situations by executing the algorithms using Python. Based on the overview of the system, some basic elements that are required to model and simulate situations in MCPS are derived. To model situations in pseudocode, a relationship analysis of the system including the basic elements is made. This relationship analysis with necessary attributes of the objects leads to the modeling situation recognition for the vehicular system which includes vehicles, RSU, home, persons. So, the basic elements with the attributes from the objects are mapped in the vehicular system and the situation recognition algorithms or pseudocode are modeled. The goal of the vehicular system is to check if a particular vehicle can take a software update at a specific time. Hence, some related situations (e.g., vehicle's speed, network connection, remaining energy) are modeled to make the decision.

The final contribution of this thesis is the prototypical implementation of the concept with modeled and simulated situations. To accomplish that, an available vehicular traffic simulator SUMO is selected after comparing the requirements with many available vehicular system simulators. In SUMO, the scenarios are modeled first to execute them together. The execution of the modeled scenarios provides the simulation output with necessary data. The output data are filtered and merged together to avail a single data stream for the situation recognition. Finally, from these output data, the modeled pseudocodes are implemented using python. So, the concept is validated by the prototypical implementation of the situations modeling in SUMO and recognized situations.

Future work includes building error-free situations modeling and simulation in MCPS. The prototypical implementation in this thesis does not consider the run-time recognition of situations. The simulation environment is complicated and not complete. This comprises the need of the derivation of modeling, simulation, and testing scenarios using real-time data stream from MCPSs.

Moreover, the situation recognition must be done with optimization and efficiency as real-time requirements fulfillment is very important. The quality of the data and context which later defines the quality of the situations needs to be improved. As further research may want to enable the prediction of situations.
Another possible future work would be to validate this system concept with other MCPS scenarios and simulators.

Bibliography

[ADB+99] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, P. Steggles. "Towards a Better Understanding of Context and Context-Awareness." In: *Handheld and Ubiquitous Computing*. Ed. by H.-W. Gellersen. Berlin, Heidelberg: Springer Berlin Heidelberg, (1999), pp. 304–307. ISBN: 978-3-540-48157-7 (cit. on pp. 26, 49).

[ADS02] M. Anjanappa, K. Datta, T. Song. "Introduction to Sensors and Actuators." In: The Mechatronics Handbook, (2002), pp. 16–1 (cit. on p. 46).

[AIM10] L. Atzori, A. Iera, G. Morabito. "The Internet of Things: A survey." In: *Computer Networks* 54.15 (2010), pp. 2787–2805. ISSN: 13891286. DOI: 10.1016/j.comnet.2010.05.010. arXiv: arXiv:1011.1669v3 (cit. on pp. 24, 25, 46).

[AKK13] S. H. Ahmed, G. Kim, D. Kim. "Cyber Physical System: Architecture, Applications and Research Challenges." In: *IFIP Wireless Days* (2013), pp. –4. ISSN: 2156972X. DOI: 10.1109/WD.2013.6686528 (cit. on pp. 22, 24, 37, 39).

[AM00] G. D. Abowd, E. D. Mynatt. "Charting Past, Present, and Future Research in Ubiquitous Computing." In: *ACM Trans. Comput.-Hum. Interact.* 7.1 (2000), pp. 29–58. ISSN: 1073-0516. DOI: 10.1145/344949.344988. URL: http://doi.acm.org/10.1145/344949.344988 (cit. on p. 26).

[BB09] M. C. Bujorianu, H. Barringer. "An Integrated Specification Logic for Cyber-Physical Systems." In: *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS* (2009), pp. 291–300. DOI: 10.1109/ICECCS.2009.36 (cit. on p. 38).

[BFD+15] B. Balaji, M. A. A. Faruque, N. Dutt, R. Gupta, Y. Agarwal. "Models, Abstractions, and Architectures: The Missing Links in Cyber-Physical Systems." In: *52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. (2015), pp. 1–6. DOI: 10.1145/2744769.2747936 (cit. on p. 38).

[BHK+15] U. Breitenbücher, P. Hirmer, K. Képes, O. Kopp, F. Leymann, M. Wieland. "A Situation-Aware Workflow Modelling Extension." In: *Proceedings of the 17th International Conference on Information Integration and Web-based Applications &Services - iiWAS '15* (2015), pp. 1–7. DOI: 10.1145/2837185.2837248. URL: http://dl.acm.org/citation.cfm?doid=2837185.2837248 (cit. on pp. 27, 33, 35, 36).

[CDNP17] A. Cottignies, M. Daley, E. Newton, H. Parker. “rFpro & SUMO: The Road To A Complete Real-Time Simulation Of Urban Environments for DIL, ADAS and Autonomous Testing.” In: *SUMO 2017 – Towards Simulation for Autonomous Mobility* 31 (2017). ISSN: 1866-721X (cit. on p. 62).

[CHHK18] C. C. Cheng, P. C. Hsiu, T. K. Hu, T. W. Kuo. “Oasis: A Mobile Cyber-Physical System for Accessible Location Exploration.” In: *Proceedings of the IEEE* 106.9 (2018), pp. 1744–1759. ISSN: 00189219. DOI: 10.1109/JPROC.2018.2817511 (cit. on p. 22).

[Cyb16] Cyber Physical Systems Public Working Group of U.S. Department of Commerce’s National Institute of Standards and Technology (NIST). *Framework for Cyber-Physical System*. PICASSO Project Opportunity Report Version 1.0. (2016), p. 266. DOI: https://doi.org/10.6028/NIST.SP.1500-201 (cit. on pp. 21, 25).

[CYH+17] B. Chen, Z. Yang, S. Huang, X. Du, Z. Cui, J. Bhimani, X. Xie, N. Mi. “Cyber-Physical System Enabled Nearby Traffic Flow Modelling for Autonomous Vehicles.” In: *IEEE 36th International Performance Computing and Communications Conference, IPCCC 2017* (2017), pp. 1–6. ISSN: 1097-2641. DOI: 10.1109/PCCC.2017.8280498 (cit. on p. 39).

[Dey99] A. K. Dey. “Understanding and Using Context.” In: *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing* (1999), pp. 304–307 (cit. on p. 28).

[FHWM16] A. C. Franco da Silva, P. Hirmer, M. Wieland, B. Mitschang. “SitRS XT – Towards Near Real Time Situation Recognition.” In: *Journal of Information and Data Management* 7.1 (2016), pp. 4–17 (cit. on p. 35).

[GBF+16] J. Guth, U. Breitenbücher, M. Falkenthal, F. Leymann, L. Reinfurt. “Comparison of IoT Platform Architectures: A Field Study based on a Reference Architecture.” In: *2016 Cloudification of the Internet of Things (CIoT)*. IEEE, (2016), pp. 1–6. DOI: 10.1109/CIOT.2016.7872918 (cit. on p. 45).

[GHH+17] Y. Guo, X. Hu, B. Hu, J. Cheng, M. Zhou, R. Y. Kwok. “Mobile Cyber Physical Systems: Current Challenges and Future Networking Applications.” In: *IEEE Access* 6 (2017), pp. 12360–12368. ISSN: 21693536. DOI: 10.1109/ACCESS.2017.2782881 (cit. on p. 22).

[HHM+17] M. Hüffmeyer, P. Hirmer, B. Mitschang, U. Schreier, M. Wieland. “SitAC – A System for Situation-aware Access Control - Controlling Access to Sensor Data.” In: *Proceedings of the 3rd International Conference on Information Systems Security and Privacy* Icissp (2017), pp. 113–125. DOI: 10.5220/0006186501130125. URL: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006186501130125 (cit. on p. 36).

[HPK12] D. D. Hoang, H.-Y. Paik, C.-k. Kim. “Service-Oriented Middleware Architectures for Cyber-Physical Systems.” In: *IJCSNS International Journal of Computer Science and Network Security* 12.1 (2012), p. 18 (cit. on pp. 21, 40).

[HWS+15] P. Hirmer, M. Wieland, H. Schwarz, B. Mitschang, U. Breitenbücher, F. Leymann. “SitRS – A Situation Recognition Service based on Modeling and Executing Situation Templates.” In: *The 9th Advanced Summer School on Service-Oriented Computing*. (2015), pp. 113–127 (cit. on p. 34).

[HWS+16] P. Hirmer, M. Wieland, H. Schwarz, B. Mitschang, U. Breitenbücher, S. Gómez Sáez, F. Leymann. “Situation recognition and handling based on executing situation templates and situation-aware workflows.” In: *Computing* 99 (2016). DOI: 10.1007/s00607-016-0522-9 (cit. on pp. 18, 32–34, 44).

[HZL10] K. Häussermann, O. Zweigle, P. Levi. “Understanding and Designing Situation-Aware Mobile and Ubiquitous Computing Systems.” In: *World Academy of Science, Engineering and Technology. International Journal of Computer, Electrical, Automation, Control and Information Engineering* 4.3 (2010), pp. 329–338 (cit. on pp. 28, 31, 32).

[JLW+16] D. Jia, K. Lu, J. Wang, X. Zhang, X. Shen. “A Survey on Platoon-Based Vehicular Cyber-Physical Systems.” In: *IEEE Communications Surveys and Tutorials* 18.1 (2016), pp. 263–284. ISSN: 1553877X. DOI: 10.1109/COMST.2015.2410831. arXiv: arXiv:1011.1669v3 (cit. on p. 40).

[LPW+17] Y. Liu, Y. Peng, B. Wang, S. Yao, Z. Liu. “Review on cyber-physical systems.” In: *IEEE/CAA Journal of Automatica Sinica* 4.1 (2017), pp. 27–40. ISSN: 23299274. DOI: 10.1109/JAS.2017.7510349 (cit. on pp. 22, 23).

[Luc08] D. Luckham. “The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems.” In: *Rule Representation, Interchange and Reasoning on the Web*. Springer Berlin Heidelberg, (2008), pp. 3–3. ISBN: 978-3-540-88808-6 (cit. on p. 35).

[Mar97] A. Maria. “Introduction to Modeling and Simulation.” In: *Proceedings of the 1997 Winter Simulation Conference Parry , McAneny , and Dromerhauser* (1997), pp. 933–940 (cit. on pp. 28, 29).

[MHWM17] M. Mormul, P. Hirmer, M. Wieland, B. Mitschang. “Situation model as interface between situation recognition and situation-aware applications.” In: *Computer Science - Research and Development* 32.3-4 (2017), pp. 331–342. ISSN: 18652042. DOI: 10.1007/s00450-016-0335-2 (cit. on pp. 28, 47–49).

[MSAN11] R. Maia, M. Silva, R. Ara, U. Nunes. “Electric Vehicle Simulator for Energy Consumption Studies in Electric Mobility Systems.” In: *2011 IEEE Forum on Integrated and Sustainable Transportation Systems* June (2011). DOI: 10.1109/FISTS.2011.5973655 (cit. on p. 60).

[OTV17] E. Omerdic, D. Toal, Z. Vukic. “User interface for interaction with heterogeneous vehicles for cyber-physical systems.” In: *14th International Conference on Control, Automation, Robotics and Vision* 691980 (2017), pp. 13–15. ISSN: 0003-4967. DOI: 10.1109/ICARCV.2016.7838747 (cit. on pp. 22, 23).

[Pas98] J. Pascoe. “Adding Generic Contextual Capabilities to Wearable Computers.” In: *Proceedings of the Second IEEE International Symposium on Wearable Computers (ISWC’1998)* 44 (1998) (cit. on p. 26).

[PP16] K. Patel, S. Patel. “Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges.” In: *International Journal of Engineering Science and Computing, IJESC* 6.5 (2016), pp. 6122–6131. ISSN: 00219517. DOI: 10.4010/2016.1482 (cit. on pp. 25, 26).

[PZCG14] C. Perera, A. Zaslavsky, P. Christen, D. Georgakopoulos. “Context Aware Computing for The Internet of Things: A Survey.” In: *Communications Surveys & Tutorials, IEEE* 16.1 (2014), pp. 414–454 (cit. on p. 27).

[RCDD98] T. Rodden, K. Chervest, N. Davies, A. Dix. “Exploiting Context in HCI Design for Mobile Systems.” In: *in Workshop on Human Computer Interaction with Mobile Devices*. (1998) (cit. on p. 26).

[RLSS10] R. Rajkumar, I. Lee, L. Sha, J. Stankovic. “Cyber-Physical Systems: The Next Computing Revolution.” In: *Design Automation Conference*. (2010), pp. 731–736. DOI: 10.1145/1837274.1837461 (cit. on pp. 17, 22).

[SAW94] B. Schilit, N. Adams, R. Want. “Context-Aware Computing Applications.” In: *1994 First Workshop on Mobile Computing Systems and Applications* (1994), pp. 85–90. ISSN: 15277755. DOI: 10.1109/WMCSA.1994.16. URL: http://ieeexplore.ieee.org/document/4624429/ (cit. on p. 27).

[SEE16] M. Saidallah, A. El Fergougui, A. E. Elalaoui. “A Comparative Study of Urban Road Traffic Simulators.” In: *MATEC Web of Conferences* 81 (2016). ISSN: 2261-236X. DOI: 10.1051/matecconf/20168105002. URL: http://www.matec-conferences.org/10.1051/matecconf/20168105002 (cit. on pp. 58, 59).

[SERF17] M. Semrau, J. Erdmann, J. Rieken, B. Friedrich. “Modelling and calibrating situation adaptive lane changing and merging behavior on Chinese elevated roads.” In: *SUMO 2017 – Towards Simulation for Autonomous Mobility* 31 (2017). ISSN: 1866-721X (cit. on p. 61).

[SH15] F. Salim, U. Haque. “Urban computing in the wild: A survey on large scale participation and citizen engagement with ubiquitous computing, cyber physical systems, and Internet of Things.” In: *International Journal of Human-Computer Studies* 81 (2015). Transdisciplinary Approaches to Urban Computing, pp. 31–48. ISSN: 1071-5819. DOI: https://doi.org/10.1016/j.ijhcs.2015.03.003. URL: http://www.sciencedirect.com/science/article/pii/S1071581915000488 (cit. on p. 17).

[SM12] T. Sanislav, L. Miclea. “Cyber-Physical Systems - Concept, Challenges and Research Areas.” In: *Control Engineering and Applied Informatics* 14.2 (2012), pp. 28–33. ISSN: 14548658 (cit. on pp. 22, 23).

[SMC04] A. J. Sullivan, D. Malave, N. Cheekoti. “Traffic simulation software comparison study.” In: *University Transportation Center for Alabama* (2004) (cit. on p. 59).

[SS16] S. Singh, N. Singh. “Internet of Things (IoT): Security challenges, Business Opportunities & Reference Architecture for E-commerce.” In: *Proceedings of the 2015 International Conference on Green Computing and Internet of Things, ICGCIoT 2015* (2016), pp. 1577–1581. DOI: 10.1109/ICGCIoT.2015.7380718 (cit. on pp. 24–26).

[SWYS11] J. Shi, J. Wan, H. Yan, H. Suo. “A survey of Cyber-Physical Systems.” In: *2011 International Conference on Wireless Communications and Signal Processing, WCSP 2011* (2011). ISSN: 00189286. DOI: 10.1109/WCSP.2011.6096958. arXiv: 1306.2422 (cit. on p. 22).

[Tal08] C. Talcott. “Software-Intensive Systems and New Computing Paradigms.” In: Springer-Verlag, (2008). Chap. Cyber-Physical Systems and Events, pp. 101–115. ISBN: 978-3-540-89436-0. DOI: 10.1007/978-3-540-89437-7_6. URL: http://dx.doi.org/10.1007/978-3-540-89437-7_6 (cit. on p. 37).

[TGP08] Y. Tan, S. Goddard, L. C. Pérez. “A Prototype Architecture for Cyber-physical Systems.” In: *SIGBED Rev.* 5.1 (2008), 26:1–26:2. ISSN: 1551-3688. DOI: 10.1145/1366283.1366309. URL: http://doi.acm.org/10.1145/1366283.1366309 (cit. on p. 37).

[VF13] O. Vermesan, P. Friess. “Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems.” In: River Publishers, (2013) (cit. on p. 18).

[WCG08] Y. Wang, M. C. Vuran, S. Goddard. “Cyber-Physical Systems in Industrial Process Control.” In: *ACM Sigbed Review* 5 (2008). DOI: 10.1145/1366283.1366295 (cit. on pp. 37, 38).

[WP06] R. West, G. Parmer. “A Software Architecture for Next-Generation Cyber-Physical Systems.” In: *Position paper at the NSF Cyber-Physical Systems workshop* (2006), pp. 1–3. ISSN: 17574676. DOI: 10.1002/emmm.201200244 (cit. on p. 38).

[WSBL15] M. Wieland, H. Schwarz, U. Breitenbücher, F. Leymann. “Towards situation-aware adaptive workflows: SitOPT - A general purpose situation-aware workflow management system.” In: *2015 IEEE International Conference on Pervasive Computing and Communication Workshops, PerCom Workshops 2015* (2015), pp. 32–37. DOI: 10.1109/PERCOMW.2015.7133989 (cit. on p. 28).

[WZZ+14] J. Wan, D. Zhang, S. Zhao, L. Yang, J. Lloret. “Context-aware vehicular cyber-physical systems with cloud support: Architecture, challenges, and solutions.” In: *IEEE Communications Magazine* 52.8 (2014), pp. 106–113. ISSN: 01636804. DOI: 10.1109/MCOM.2014.6871677. arXiv: arXiv:0910.2486v1 (cit. on p. 39).

All links were last followed on February 15, 2019.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature