FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

DOCTORAL THESIS

Vladimír Matěna

Integration Paradigms for Ensemble-based Smart Cyber-Physical Systems

Department of Distributed and Dependable Systems

Supervisor of the doctoral thesis:	doc. RNDr. Tomáš Bureš, Ph.D.
Study programme:	Computer science
Study branch:	Software systems

Prague 2018

Title: Integration Paradigms for Ensemble-based Smart Cyber-Physical Systems

Author: Vladimír Matěna

Department of Distributed and Dependable Systems: Department of Distributed and Dependable Systems

Supervisor: doc. RNDr. Tomáš Bureš, Ph.D., Department of Distributed and Dependable Systems

Abstract: Smart Cyber-Physical Systems (sCPS) are complex systems performing smart coordination that often require decentralized and network resilient operation. New development in the fields of the robotic systems, Industry 4.0 and autonomous vehicular system brings challenges that can be tackled with deployment of ensemble based sCPS, but require further refinement in terms of network resilience and data propagation. This thesis maps the use cases of the sCPS in the aforementioned domains, discusses requirements on the ensemble based architecture in terms of network properties, and proposes recommendations and technical means that help to design network aware ensemble based sCPS. The proposed solutions are evaluated by the means of target systems simulation using state of the art realistic network and vehicular simulators.

Keywords: smart cyber-physical systems, reliability-aware design, real-time communication, autonomic components, ensembles, simulation environment, testbed

Acknowledgments

In this text I would like to thank all who supported me through my doctoral studies and made this work possible. First, I would like to thank my advisor Tomáš Bureš for all the help, guidance, advises, ideas, and specially for keeping high spirits in every situation. I am also deeply grateful to František Plášil who helped me a lot especially in the beginning of my studies. His advises and guidance let me understand my topic in a broader context and the countless hours he spent with me writing the papers and correcting all the mistakes helped me to put this thesis together. Further, I would like to thank Petr Hnětynka for his help and specially for keeping an eye on all the deadlines and Pavel Parízek for constantly pushing me forward and providing me with advises on all the technical details.

Next, I would like to thank all my colleagues at the Department of Distributed and Dependable Systems who created a fruitful and positive work environment for me. In particular I would like to thank Rima Al Ali, Lubomír Bulej, Jakub Daniel, Vlastimil Dort, Ilias Gerostathopoulos, Vojtěch Horký, Pavel Ježek, Zbyněk Jiráček, Michał Kit, Jan Kofroň, Filip Krijt, Dominik Škoda, Petr Tůma, and Jiří Vinárek. I am also grateful to Petra Novotná for all the support with administrative tasks. Without her help we all would end up stuck somewhere in a distant country.

Also I would like to thank the institutions that financially supported my work. In particular I would like to thank Charles University Grant Agency project No. 391115.

Last but not least, I would like to thank my parents Vladimír and Jana for their generous material and mental support as well as my grandparents who together with my parents sparked my interest in science and technology. Finally, I would like to thank my fiancée Hana for her great loving support, valuable advises, and all the hours we were talking about this.

Contents

CHAPTER 1

Introduction

At the brink of the information age the computers are finding their way from processing information in offices and data centers into direct control of physical entities anywhere in the world. The recent development in robotics, automotive, logistics and aviation promises rapid emergence of autonomous and interconnected machines. These, in the form of smart fridges, security systems, entertainment systems, Computer Numerical Control (CNC) lathes, robotic vacuum cleaners, and heating systems, are slowly filling our houses and factories. In the same way the general public spaces, such as streets and air, are already containing prototypes of self driving cars and drones. In future, streets are about to be occupied by autonomous machines delivering packages, doing cleaning tasks, performing maintenance, and continuously monitoring everything.

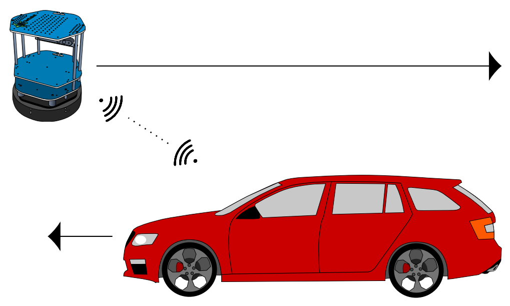
The resulting mixture of entities naturally needs to cooperate and share information in order to maintain safety and efficiency of operation. This brings interesting challenges not only in coordination and communication, but also in the area of system design and architecture.

For a human in a crowded and noisy underground it is quite an easy and natural task to coordinate with a stranger on precedence in a narrow passage. Unfortunately, it is a bit more challenging to introduce coordination into an open-ended, dynamic and large scale system composed of autonomous entities using a lossy communication. A field of sCPS is addressing these challenges.

1.1 Smart Cyber-Physical Systems

The recent advances in computer manufacturing and, in particular, mobile computers open a brand new field of applications. Each year we see introduction of general purpose computational devices that are smaller and cheaper. Apart from this, these devices are also gaining significant computational power without raising energy demands enabling both complex applications and power savings.

Those advances encourage engineers to do two things. First, to replace legacy single purpose controller hardware by mass produced programmable controllers that enable introducing additional features. i.e. replacing mechanical thermostat with temperature sensor and programmable controller that can easily maintain stable temperature and, in addition, also collect statistics or follow defined schedule. Second, to introduce systems that used to be impossible or just too complex and expensive to be build. i.e. monitoring humidity of a soil in flowerpot used to be beyond level of complexity suitable for deployment in households, but currently it can be implemented by a child as a high school project.



As already mentioned, the computational power is at hand, but without proper communication the most complex projects cannot be implemented. For a long time, the wires have been used to connect actuators and sensors to controllers in the field of the Cyber-Physical System (CPS). Even when their reliability is very hard to replace, the modern systems composed of small and possibly mobile devices rely on wireless media as means of communication. This is where the state of the art is changing really fast. The Wi-Fi and Bluetooth are around for quite some time as well as mobile internet connection, but the properties of these connection means are changing. Current Wi-Fi ac can easily replace wired connection enabling wireless high resolution video streaming. Bluetooth low energy standard can let small, battery powerd devices operate for months or even years. Performant The 4th Generation mobile networks (4G) mobile internet connection is about to cover most of the population while the The 5th Generation mobile networks (5G) is set to significantly increase its capacity to cover much more devices and remove necesity for local Wi-Fi hotspots or even erradicate cables in some places [8].

Powerful, small and connected controllers are only one part of the new CPS. Sensors are developing in a similar way as processing units do. Becoming cheaper, smaller and more performant devices like Light Detection And Ranging (LIDAR) or depth camera provide sensor data of quality that used to be hard and expensive to achieve.

Actually, already deployed devices such as smart phones, security systems and connected cars are already equipped with high quality sensors and necessary processing power [9]. Most smart phones include Global Position System (GPS), magnetic field sensor, light sensor, microphone, gyroscope and camera. The potential of these devices is currently used mostly locally. Assuming possibilities of current and future communication, it is going to be possible to build complex CPS composed of many, possibly small, devices.

As the availability of communication means and smart devices started era of massive interconnection among devices called Internet of Things (IoT), it is presumable that those connections will be used also for remote or even distributed control. As a result CPS composed of a large number of interconnected devices are, at last in some literature, called sCPS.

Keeping the purpose of CPS the sCPS maintain interconnection between cyber and physical world. The only difference is shift from rather static connection between sensor, controller and actuator to a dynamic, open, and distributed system composed of heterogeneous devices. Where a relation between cyber and physical world used to be determined by cables connecting sensor and actuators to controller, now dynamic groups of devices are in place. The resulting system has possibly much higher capabilities, but these come at the cost overall system complexity. Due to this sCPS introduce new challenges in architecture design and

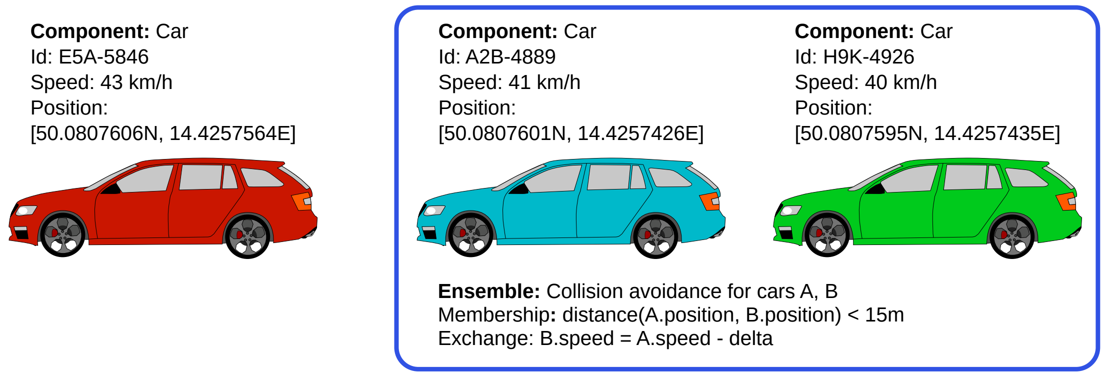

Component: Car
Id: E5A-5846
Speed: 43 km/h
Position:
[50.0807606N, 14.4257564E]
Component: Car
Id: A2B-4889
Speed: 41 km/h
Position:
[50.0807601N, 14.4257426E]
Component: Car
Id: H9K-4926
Speed: 40 km/h
Position:
[50.0807595N, 14.4257435E]
Ensemble: Collision avoidance for cars A, B
Membership**:** distance(A.position, B.position) < 15m
Exchange: B.speed = A.speed - delta

Figure 1.1: An example of vehicular EBCS.

communication that needs to be addressed in order to take a full advantage of those systems.

1.2 Motivation

As stated earlier, the current development in the field of sCPS brings dynamic systems composed of many, possibly small, nodes that closely cooperate together in order to fulfill shared goals. The complexity of the problem requires a systematic approach backed by a proper system architecture in order to implement the system properly. An Ensemble Based Component Systems (EBCS) [10] is one of the an architectural styles suitable for designing those systems.

An EBCS is build on top of ensembles of executable components [11]. The component encapsulates business logic implemented by processes and a knowledge that represent the state of the component and defines its external interface. The ensemble is a dynamic group of components defined by a membership condition expressed as a function of member knowledge. Once the ensemble is established the knowledge exchange, that is a part of ensemble definition, is performed. The exchange is a function that changes the knowledge of particular ensemble member based on the combined knowledge of all members of the ensemble. An example of EBCS is a collision avoidance system where cars close to each other form an ensemble and limit speed of cars in the back in order to maintain safe distance as displayed in Figure 1.1.

Even though the abstractions work well in theory, the real implementation of these systems is not so straightforward. In particular, implementing the system on top of a poor wireless network is really a challenging task that needs to be addressed at design time and reflected in the system architecture.

Possible applications that require mentioned integration of network properties at design time include robotic systems, Industry 4.0, and and smart cars. All these fields are at the brink of massive expansion in the near future. Even though the current experimental implementations are not required to address network related design time challenges as those use simplified network model or simply work around the problem on a lower level, the future pressure on cost reduction, reliability, and massive integration with existing systems will force designers to address those challenges.

The applications of dynamic grouping implemented as an EBCS are many.

In the field of robotic systems the envisioned usage ranges from coordination of automated search and rescue missions through the coordination of military units towards the coordination of household cleaning machines and general coordination of IoT systems. Regarding the Industry 4.0, EBCS can offer safety enforcement through coordination between humans and robots at workplace as well as coordination on manufacturing particular products that translate into planning and scheduling task. Finally, the dynamic groups offer many applications in the area of automated cars. Applications in this field include collision avoidance, traffic optimization, movement coordination, and formation of platoons.

On of the common base of all these, and most modern systems in general, is strong usage of network communication. The general availability of the network interfaces, their low price and power consumption together with customer demand for well integrated and remotely managed systems fuels demand for networked sCPS and motivates usage of EBCS.

With the new features available, from possibly large volumes of new devices connected, also comes the hassle of managing the communication. Managing connections among thousands of devices spread across the city or even tens of devices populating a single household is challenging in terms of network infrastructure and deciding what data should be send where. The challenge is even more demanding if a distributed system, operating on top of possibly restricted and unreliable wireless network, is taken into consideration. Such kind of a system is required in applications where reliable network is not available by design, such as military or search and rescue applications, or in cases where the provided network cannot be relied on due to excessive usage or poor coverage. Because of this even, applications designed to operate in peaceful and stable environments need to be ready for network related troubles due to congestion in populated areas and poor coverage in countryside. Finally, systems that needs to be reliable, such as autonomous driving, vehicle or robot coordination, and such similar systems needs to be aware of the underlying network in order to adjust their behavior.

In all the fields mentioned above usage of EBCS faces network related problems, preventing efficient and straightforward development, that needs to be reflected in the architecture in order to design such systems that are naturally aware of the network related problems. In general these can be summarized as (i) formation of cooperation groups in real network environment; (ii) testing of the real behavior of the proposed system.

1.3 Overall Goals

The high level goals of this thesis, based on the motivation, are aiming on the extension of the EBCS architecture concepts with network awareness and overall optimization of the architecture towards realistic network properties. The main pitfall of the current EBCS architectures, with respect to the network awareness, seem to be lack of expressiveness for the network distance between nodes hosting the components that form the system. Moreover, the network unaware architecture may bring an false assumption of all data being available everywhere at a very low cost. In order to mitigate these deficiencies the high-level goals are outlined for this thesis in the following paragraphs.

Distributed group formation and operation with network limits The first goal aims on design of an architecture that supports group formation while being aware of the network and its realistic limits. This translates into decision on where the data, necessary to form the groups, should be spread and how to balance network usage and utility produced by the system. Moreover, the operation of the already formed group needs to be supported including data exchange and coordination of group members under the realistic network conditions. This includes coping with data loss and latency issues in coordination, especially in case when a real-time system is considered.

Evaluation of networked sCPS applications Evaluation is necessary in order to verify achievement of the previous goal as well as to ease development of new applications using the proposed architecture. The following steps needs to be taken in order to verify correct design of the proposed tools and to let the other scientists in the field conduct experiments. First, the simulation of the target environment needs to be created that specially focus on realistic network. Second, a test-bed needs to be present that is easy to run and contains all the necessary tools to enable widespread usage of it.

The goals of this thesis, as well as matching challenges and research questions are detailed in Chapter 3 after introducing necessary vocabulary.

1.4 Document Structure

This thesis is structured as follows. Chapter 2 presents state of the art in the field of interest while specially focusing on sCPS related architectures, applications, networking, and simulations. Thesis goals are introduced in Chapter 3 where challenges and research questions are formulated. In Chapter 4 desired use cases in the field of sCPS are described while the requirements on the network that follow from mapping of the use cases to the EBCS are described in Chapter 5.

Chapter 6 outlines approach towards addressing goals of this thesis. First different ensemble formation techniques are described including their implications towards network communication requirements in Chapter 7. Next the specifics of designing hard real-time networked sCPS based on the ensembles are presented in Chapter 10. The real-time analysis is followed by network traffic optimization conducted using communication groups and adaptations in communication as described in Chapters 8 and 9 respectively. Later, the means of evaluation of the sCPS are addressed by listing of frameworks implemented or modified as part of this thesis, Chapter 11, and, finally, a test-bed based on the frameworks is presented in Chapter 12.

The ideas presented in previous chapters were evaluated by means of simulation and the results of these are displayed and described in Chapter 13. In the Chapter 14 the work related to the topic of this thesis is listed according to the challenges of this thesis. Finally, Chapter 15 concludes the work and discusses possible new directions of research in the field.

1.5 Used Conventions

Part of the text of this thesis, located mostly in the Chapters 8, 9, 10, 12, and 13, was reused from publications co-authored by the author of this thesis. Mentioned publications are listed in Chapter 3. Reused text, marked with horizontal black line following the page margin such as in the case of this paragraph, was included verbatim with a few exceptions.

- The text parts that used to include description of terms already mentioned somewhere else in the text were replaced by references to the description.
- Part of the text was rephrased and some of the terms replaced with synonyms in order to keep consistent vocabulary and language across the whole document.
- Some of the figures that are referenced from the reused text were not taken verbatim but regenerated from original data in order to improve their visual appearance and precision.
- In case of Chapter 9 the reused results were extended with previously unpublished data using different method of simulation. These extra data are described and a brief comparison with the original data is included.

Apart from these the reused text was taken verbatim. In general the meaning is consistent with version of the text that was originally published.

CHAPTER 2

Background

In this chapter the background in the field is described that form a base of the research presented in this thesis. Included topics range from description of current trends in the target domains; through the description of the EBCS and the communication techniques that are required to support the concept outlined by the architecture; towards the means of evaluation such as different simulation frameworks.

First, the target application domains are described in Section 2.1. Second the EBCS software architecture, closely related to the sCPS, are presented in Section 2.2. Then, in the Section 2.3 the network technologies suitable for target systems are described. Following is an explanation of a declarative group membership description in Section 2.4, while the last Section 2.5 serves as an overview of suitable simulation frameworks.

2.1 Application Domains

The following text presents state of the art systems that can benefit from network aware EBCS architecture. Applications in the field of robotic systems, Industry 4.0, and smart cars are described. Later, smart cars are analyzed in detail as those can be used to generalize problems and solutions common to all the mentioned fields of interest.

2.1.1 Robotic Systems

In this section possible applications in the domain of robotic systems are described. As of today many tasks are already handled by various kinds of robots, moreover in near future the set of such tasks is going to increase. Currently, most of the robots are still stationary and require no or just simple coordination. But, in future, the advance of IoT will make the amount of mobile robots requiring complex dynamic coordination higher. The requirement for complex coordination comes both from the increasing number of robots that may cooperate and rougher, less predictable environments these robots are going to be exposed to.

One of the fields that is already gaining interest in coordination of robotic systems is surveillance. Assuming a request to scout a terrain for intruders of traces of dangerous chemicals, a group of unmanned vehicles is deployed in the area. It is not important whenever these are Unmanned Aerial Vehicle (UAV) or Unmanned Ground Vehicle (UGV) or even submarines. All of these will need to coordinate on scattering the area and collision avoidance. Assuming the terrain is not known in advance the vehicles will need to perform some sort of Simultaneous Localization and Mapping (SLAM). Thus a new subarea, that needs to be scanned, can be discovered during the process of scouting. Also, some of the vehicles can be temporary, or even permanently, unavailable due to malfunction, maintenance or necessity to charge their batteries. Due to these facts, the coordination of the overall operation needs to deal with dynamic and open-ended system.

From the surveillance it is only a small step to the field of search and rescue operations. Actually, the search and rescue operation can be decomposed into the search part, which is equivalent to already described surveillance application, while the rescue part introduces even bigger challenge for dynamic coordination as new victims can be discovered and status of already known victims can change. Moreover, the robotic rescuers can break or stuck when dealing with rough terrain and the human rescue workers can easily become new victims due to accidents or sudden changes of the environment.

Another field which is going to receive massive help from robotic systems is the military. Recently, a widespread adoption of various kinds of the UAV have been seen in the armies of developed countries. This trend is expected to continue and new kinds of robotic warriors are about to enter the service. Once autonomous units enter the service the already existent pressure on more agile coordination of battle operations will even increase. A sign of this is the existence of Battle Management Language (BML) [12] which is supposed to be processed both by human warriors and autonomous robotic forces. It is not necessary to argue that a battle scene is a dynamic environment where even the numbers, locations and overall status of allied units is hard to predict.

Switching back to the peaceful applications, coordination of robotic vacuum cleaners in a large household of office building poses also an interesting challenge. Again this problem encompasses dynamic groups of cleaners and humans that need to coordinate in order to clean the house effectively and, if possible, not to collide with both humans and each other. Dynamic coordination introduces possibility to change both the area to clean, number of cleaning robots and humans at the runtime. Even though the benefit of this application may not be as significant as the others, low requirements on safety and reliability of the system make it quite easy to be implemented.

The vacuum cleaners are just a small step from a full home automation. Even when the rest of the devices in household are mostly stationary objects, even these devices can form dynamic groups. For instance security, ventilation and multimedia systems can pair rooms and humans in order to provide the best possible service.

One step further from smart household is the general IoT. It is easy to imagine dynamic groups of devices beyond the household. Actually, interaction in open-ended system composed of mobile devices heavily relies on dynamic coordination, unless extensive planning is performed in advance. In this category devices ranging from fridges to street lamps are expected to form dynamic groups with humans and other actors in the system in order to serve their purpose efficiently.

Yet another example of a robotic system with necessity for dynamic coordination is Industry 4.0. Due to its broad impact in the field, Industry 4.0 is discussed separately in the following Section 2.1.2.

2.1.2 Industry 4.0

Called digitized industry, Industry 4.0 or advanced manufacturing, there is an obvious trend in using robots and other means of automated manufacturing where possible. Usage of mostly computer controlled hardware brings new opportunities in manufacturing process design. Among others, this is ability to produce highly customized products on customer demand and advanced standardization that enable high predictability in manufacturing process.

With these new abilities also new challenges arise. One of the challenges is the coordination necessary to maintain safety at a workplace. Current robots present in industry are mostly static, mounted on pillars made of steel and concrete, as those sometimes multiple ton weighting devices are used mainly for manipulation, welding and binding of objects that humans cannot handle due to their excessive weight or dimensions. Whereas in the future, smaller mobile robots will be deployed to handle tasks currently performed by humans. These robots will hardly replace all humans in the factory as some humans are still needed for: (i) supervision; (ii) tasks that require common sense decisions; (iii) activities so rare that creation of specialized machine for them does not pay off. Due to the fact that those humans will need to closely cooperate with robots current safety approaches, such as complete separation of robots and humans at the workplace, are not applicable. Instead, the robots and the humans will need to dynamically coordinate on mutual avoidance in space and time while allowing all workers to operate at workplace without statically set restrictions. Due to the fact that a collision with robot can seriously harm or even kill the colliding human, the coordination mechanism and control processes used to perform the avoidance needs to be reliable with respect to both failures and hard real-time processing. These two requirements make dynamic coordination, in a possibly open-ended system, performed on top of wireless network a very challenging task. Moreover, it is necessary to note that in the open and dynamic system there are no guarantees on number of workers entering the factory or accumulating in one place. Thus, the desired system needs to handle situations where available network bandwidth and processing power on the nodes cannot handle excessive number of workers occupying particular area while keeping the same Quality of Service (QoS).

Together with the safety related coordination, it is important to coordinate workers performing the same task or even coordinate on task assignment to the workers. Assuming the production of factory is rapidly changing based on current demand, it is necessary to decompose high level tasks set by customers, allocate resources to sub-tasks and coordinate workers and resources assigned to particular task if necessary.

Even when the theory behind planning and scheduling is quite different from dynamic coordination, it might be beneficial to solve both problems together due to the fact that once the workers are assigned to a task it is necessary to maintain the coordination of such group. Otherwise, the dynamic reconfiguration performed by the planner and the task scheduler needs to be reflected in coordination. Even when this is technically possible, it would add complexity and possibly break the clarity of the design.

2.1.3 Smart Cars

Fully autonomous self driving cars, or just cars with advanced driving assistance, able to maintain speed and stick to lane, are often referred as smart cars. This area contains quite comprehensive set of cases that can be used to motivate research in the field of dynamic coordination.

First, the utmost important feature of all cars, collision avoidance. Modern, fully and partially autonomous cars rely on sensors such as LIDAR, Radio Detection and Ranging (RADAR), depth cameras to build-up virtual map of a surrounding environment that is later used to avoid collisions with static objects, pedistrians, and other cars. Even when those senors are, in some ways, more advanced than human senses and can even build the map in complete darkness the processing of the data is still not perfect. Actually, those systems are barely able to match human drivers. Due to this, presence of accidents caused by hard to predict factors such as broken truck behind the corner, unexpected breaking of preceding car, or trees that suddenly fell onto the road cannnot be neglicted. Even when the risk of such events cannot be ruled out, proper communication of such events can improve the situation a lot. Assuming radio communication links between cars such as Wireless Access in Vehicular Environments (WAVE) [13], it is possible to coordinate on rapid breaking and emergency stopping in such a way that the risk of accident is significantly reduced [14]. Also the cars can share the information on objects blocking the road and other obstructions in the traffic.

This idea extends from the safety to a general optimization of the traffic. Assuming the communication among cars described above, the cars can easily share safety critical information about the obstacles on the road, but also add information related to traffic density and other traffic related conditions such as weather and road surface quality. Naturally, this information is about to be transported with accordingly lower priority than the safety critical data mentioned above. Availability of such information can be utilized to perform distributed optimization of traffic in a city or a highway section in order to maximize capacity of the available roads.

Another field where dynamic coordination is stressed in vehicular systems is a movement coordination. There are many situation when drivers need to coordinate in order to drive safely. These situations ranges from mandatory to informal rules. First, there are infrastructure supported coordination signals such as traffic lights, traffic signs and horizontal traffic signs. These are followed by rules not backed by signs such as priority-to-the-right that let drivers coordinate on crossings where no coordination infrastructure is present. Last, there are informal rules for coordination, not backed by any law, which are not intended to enforce safety, but to make traffic more fluent. These coordination rules are quite many ranging from stalemate resolution at unregulated crossings to favors to other drivers. An example of such coordination is changing a lane on a highway so that a car joining the highway proceed without restrictions.

The informal rules are easy to implement using dynamic coordination means implemented on top of necessary data spread by wireless broadcasts. In autonomous vehicles this approach naturally replaces hand gestures and guessing of intends on the other human drivers. Implementing this kind of voluntary coordination, when broad into massive scale, can easily improve traffic situation and reduce traffic jams. For example cars can form ad-hoc platoon and speed up together at traffic lights while improving throughput of the crossing a lot. Furthermore, implementation of informal rules doesn't need to be safety critical in most cases. For instance, when changing lines in order to let another car join a highway, the joining car can move into a gap created by car already on highway. The presence of the gap is guaranteed by on-board sensors of the joining car, thus the safety does not need to rely on gap negotiation procedure.

Implementing compulsory coordination rules is a bit more chalenging as it reuqies a safety critical system to be implemented. On the other hand, proper implementation of dynamic coordination for these situations unlocks potential for optimization and removes limitations of static coordination means. For instance classic traffic lights are a rather static mean of coordination. On one hand, these are simple to implement and quite secure. On the other hand, smart crossing [15] with dynamic coordination of cars coming from multiple directions can achieve much higher throughput. Unfortunately also the implementation complexity is much higher.

Yet another field of applications for dynamic coordination is ad-hoc formation of platoons. This technique, also called platooning, enables fuel savings due to reduction of air drag and allows the drivers to take a rest when the car is part of the platoon. Regarding the dynamic coordination there are actually two challenges in this field: (i) is to coordinate cars in the platoon in order to maintain the platoon in operational and safe state. This includes negotiation of braking capacity, calculation of platoon speed, maintenance of gaps between cars, ordering cars in the platoon, and coordination in case of emergency breaking events; (ii) is to coordinate cars that are intending to join the platoon. This includes grouping of cars sharing the same or similar destination in order to form a new platoon and choosing the right platoon for lone cars. The main difference between those challenges are the different conditions for communication. While the challenge (i) relies mostly on the local low latency communication the challenge (ii) expects usage of long range communication, possibly the mobile Internet Protocol (IP) network connection.

Use Cases

Further motivation can be found in the use cases presented in the Chapter 4. The automotive use cases are discuses in detail including samples of dynamic group specification and detailed analysis of the problem. The use cases are presented in two groups.

First, the applications that require mostly local coordination and data sharing are presented. These include mostly safety critical application ensuring safety of operation and local coordination of cars in the street that can be handled using direct broadcast. i.e. precedence negotiation, low level platoon management and obstacle discovery.

Second, the applications that build on a long range IP network. These include high level management applications that ensure coordination of potentially large groups of cars using infrastructure based networks such as Long-Term evolution (LTE) or 5G. i.e parking place management and platoon building.

These serve to show the general problems that are similar to problems from the domains mentioned above. In fact, it is not important whenever the group of cars is being coordinated in the street, or group of robotic vacuum cleaners is coordinated in an office corridor, or even a bunch of mobile manufacturing machines are coordinated in a factory. In all these use cases the ideas, requirements, and also the solutions are all almost the same.

2.2 Ensemble Based Component Systems

Ensemble Based Component Systems [10] are class of systems combining component design, agent-oriented soft real-time execution, and ensemble-based communication. These bases give EBCS properties suitable to satisfy demands of the sCPS.

The system architecture of EBCS is based on the components. As such EBCS are dynamic systems enabling emergence of the components at runtime. Components are packages of functionality bound to different entities in the system. A component contains its internal data and a belief of internal state of the other components. Each component operate in an isolation and base its actions on its internal data and a belief of a state of the other components. The interaction with the other components, based on a belief is described using, a design time specified, ensembles that feature a dynamic groups of components.

Different EBCS based architectures exists. These include Handling massively distributed systems with ELaborate ENsemble Architectures (Helena) [16], Service Component Ensemble Language (SCEL) [11] with Java Run-time Environment for SCEL Programs (jRESP) [17], and Dependable Emergent Ensembles of Components (DEECo) [10].

2.2.1 Dependable Emergent Ensembles of Components

Dependable Emergent Ensembles of Components model [10] is an instance of the EBCS. It bases on the EBCS and gives components and ensembles concrete semantics that enable development and deployment of real DEECo based systems.

```java
@Component
public class ExampleComponent {
  // Component knowledge definition
  public String id;
  public Position position;
  public State state;

  @Process
  @PeriodicScheduling(period = 500)
  public static void exampleProcess(@In("position") ParamHolder<Position> pos) {
    if(pos.distanceTo(TARGET_POSITION)) {
      ...
    }
  }
}
```

Listing 2.1: Example of JDEECo component and ensemble.

DEECo architecture is based on concepts of a component, knowledge and ensemble. The component wraps processes that encode business logic of the system and knowledge that captures the state of the component. The processes are scheduled periodically or triggered by knowledge change. They can read and write knowledge as well as execute arbitrary code. The ensemble is a dynamic group of components described by a membership condition and knowledge exchange function. The condition is periodically evaluated, when it succeeds the knowledge exchange is executed on top of the knowledge of the ensemble members.

```
@Ensemble
@PeriodicScheduling(period=1000)
public class ExampleEnsemble {
 @Membership
 public static boolean membership(
    @In("member.id") String memberId,
    @In("coord.id") String coordId,
    @In("member.position") Position memberPos,
    @In("coord.position") Position coordPos) {
  return (! memberId.equals(coordId)) && ...
 }

 @KnowledgeExchange
 public static void map(@Out("member.state") ParamHolder<state> memberState) {
     memberState.value = State.InEnsemble;
 }
}
```

Listing 2.2: Example of JDEECo ensemble.

The ensemble formation process, that encompass membership evaluation and initiation of the knowledge exchange can be either centralized or distributed. Distributed ensemble formation is usually necessary in order to meet requirements on resilience to hardware failures and real-time guarantees. The nature of the ensemble membership and knowledge depends on the implementation. The most feature complete implementation is called JDEECo. Examples of the JDEECo component and ensemble definition are given in Listing 2.1 and 2.2 respectively. There are other frameworks that support DEECo components and ensembles discussed in Chapter 11. The details of two different ways of specifying ensembles and components with respect to their network properties are described in Chapter 7.

2.3 Network Technologies Used in the Smart Cyber-Physical Systems

The network, in many cases, a bottleneck of the whole system is discussed in this section. The relevant network technologies with respect to the envisioned usage are grouped in two sections. In detail, these deal with (i) long range IP based networks that feature high throughput and global range; and (ii) mostly wireless local area networks that provide ad-hoc operation, higher degree of resiliency, but lack sophisticated infrastructure and therefore offer only limited range. In the following listing the network protocols and physical network technologies are mixed together as both provide some sort of connectivity yet the structured systems, such as IP, forge the services provided by a low level IEEE 802.3 Ethernet, Wi-Fi, and such into a long range network.

2.3.1 Short Range

Local area network technologies differ from long range networking, represented by IP network, in range and predictability. In general, the range is limited by lack of network level structure and advanced routing. Even when some sort of routing mechanism is present in some cases, it is usually not able to handle global operation. Typical device in such a network leverage broadcast with limits set by the laws of physics. This mode of operation enables natural spreading of data to nearby devices with predictable latency. When using the wired media or wireless in a controlled environment, the latency is predictable enough to provide even a hard real-time guarantees.

Apart from a bit less common technologies discussed later, a standard networking technologies such as Wi-Fi and IEEE 802.3 Ethernet can be also considered a short range communication means. These are not specially tailored for usage in CPS but can be used for fast prototyping due to their general availability and very high speed that can, in some cases, balance the lack of real-time guarantees.

Current versions of the IEEE 802.3 Ethernet usually operates on twisted pair cables, and optical fibers. The basic speed of Ethernet is $100MiB/s$, but it slowly moves to default of $1GiB/s$. The high performance systems can operate at $10GiB/s$ or even more when using optical fibers, but common hardware used in CPS usually does not support that.

Wi-Fi operates in $2.4GHz$ and $5GHz$ band with throughput of $150-300Mbit/s$ when IEEE 802.11n radio is used or $433Mbit/s - 1.69Gbit/s$ when using IEEE 802.11ac radio. Aside from the high throughput the radio band of the Wi-Fi is heavily used and the connectivity cannot be relied on. Wi-Fi is not so easy to use when guaranteed delivery is needed in a prototype system. Although interference with other traffic cannot be prevented in reality, in simulation this limitation is not present. Moreover, the general spread of the Wi-Fi make its implementation in simulation frameworks more mature. Thus usage of the Wi-Fi in simulation can be an alternative to similar, but less supported technologies such as IEEE 802.11p.

IEEE 802.15.4

An IEEE 802.15.4 is a wireless link layer technology focusing on low throughput, short range, low cost, and low power. It is mostly used in Wireless Sensor Network (WSN) and control networks. Low implementation complexity of the protocol enables usage in small, embedded, and experimental devices as the one displayed in Figure 2.1. Availability of radio broadcasts that are received by all devices in range with almost zero configuration necessary makes this technology a good start for experiments.

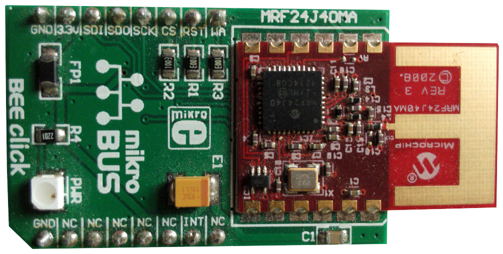


Figure 2.1: MRF24J40MA, an IEEE 802.15.4 transceiver with ZigBEE support on an extension board called click for a STM32F4 based embedded development board.

IEEE 802.15.4 stadard [18], defines multiple physical interfaces operating in multiple bands ranging from $161MHz$ to $6GHz$. The unlicensed band of $2.4GHz$ is the one where the most generally available devices operate. Data rates depend on the frequency band and modulation used. Standard data rates are defined from $19.2Kb/s$ to $2Mb/s$ as of the amendment [19] to the standard. When a beacon source is available, the devices can operate with a media access method that uses time slots to reduce collisions.

On top of IEEE 802.15.4 a ZigBee network can be build. ZigBee adds network layer with routing and dynamic reconfiguration that supports mobility [20]. A network can operate in typologies of a star, cluster tree, and mesh. Devices in a network feature single coordinator, multiple full-function devices with routing ability, and multiple reduced-function devices without routing capability.

Bluetooth

Bluetooth is a versatile technology offering wireless communication profiles ranging from virtual serial ports to audio streaming. From the point of view of the local or even personal area networking the Personal Area Network (PAN) profile is the most interesting. It uses Bluetooth Network Encapsulation Protocol (BNEP) to transmit IP packets. This is primary used to share internet connection of a cell phone with a personal computer. On top of this stack a short ranged IP network can be build. Moreover standard devices such as smartphones and laptops can connect to such network in order to monitor possible prototype system using this technology.

WAVE - IEEE 802.11p, IEEE 1609

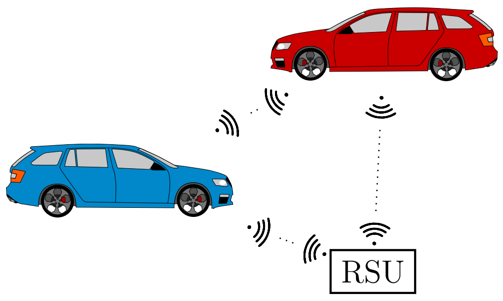


Figure 2.2: WAVE deployment, two cars with On-Board Units (OBUs) and a Road Side Unit (RSU).

WAVE is a technology base on a modified Wi-Fi radio described by the IEEE 802.11p standard. The communication model, protocol, security and physical access are described in the IEEE 1609 standard. As its name suggest WAVE focuses on vehicular systems and intelligent transportation systems. Its purpose is to provide communication among OBUs, present in moving vehicles, and between a vehicle and a RSUs, forming a road side infrastructure.

The physical layer of the IEEE 802.11p differs from the Wi-Fi (IEEE 802.11a) in used frequency band and modulation. IEEE 802.11p does suffer from much less interference thanks to the usage of a licensed band and has a bit higher throughput due to more efficient modulation.

According to [21], dealing with performance of mentioned protocols, the radio frequency range of $5.885GHz$ to $5.925GHz$ is divided into 7 channels serving different purposes and having different transmit powers and expected range. The applications running on top of these channels encompass short-range safety enforcement, long-range safety enforcement, and traffic optimization. Four access categories are defined for each channel. Each of them defines different priority. Packets belonging to a particular access category are kept in a First In First Out (FIFO) queue while packets from different queues use different back-off time when accessing the media.

Mobile ad-hoc network

A Mobile ad-hoc network (MANET) is a general term used to name an infrastructure-less network created ad-hoc by mobile and possibly wireless nodes. The typical node in a MANET employs a wireless radio technology in order to establish links to its neighbors. On top of the local links a self-healing, self-configuring routing algorithm is usually deployed. A set of nodes deployed in a MANET is visualized in Figure 2.3. Usage of particular link-layer technology and routing algorithm depends on expected mobility of the nodes, throughput requirements and other application specific constraints. In a sense, most of the local area networks mentioned in this section can be called MANETs.

Applications of MANET are many [22], [23]. Often, specific network technology and routing algorithm has to be used in particular domain. Sometimes MANETs are called differently in particular field. i.e. MANETs in vehicular domain are called Vehicular ad-hoc networks (VANETs).

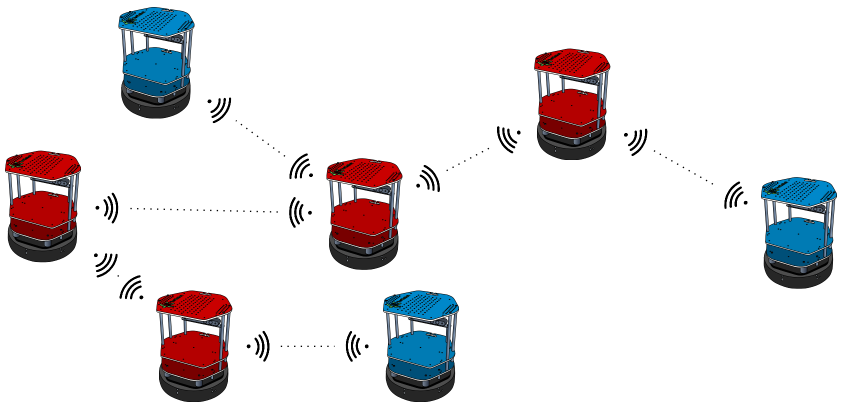


Figure 2.3: Nodes deployed in a MANET. Red nodes are auto-configured as routers.

Distributed Congestion Control

Apart from, strictly speaking, routing protocols there are protocols used for network layer management that is also very important when building the custom network. One of these is Distributed congestion control. As described for usage in vehicular networks [24] it is a protocol designed to keep media usage below a specified threshold. In particular this is important for vehicular systems to exchange periodic messages describing position and status of the cars to their neighbors. As the messages are important to maintina a safety of operation a Decentralized congestion control mechanism [25] is deployed in order to maintain some channel capacity for emergency messages. The mechanism relies on controlling transmission power, bit-rate, sensitivity, packet rate and transmission access control.

2.3.2 Long Range

Long range communication, in the context of the sCPS, provides reliable routing across a larger area that cannot be serviced by the short range communication means directly. In a sense such connectivity is provided generally by any technology that supports IP and connects to the Internet. This way almost all the sort range technologies can be used to distribute Internet connection and thus enable long range communication. This section lists some of the technologies that provide some sort of structured network on their own. These encompass mobile broadband that enables internet connection, but also introduces structured network that can offer some of the long range communication features on its own.

4G - LTE

Long-Term evolution is a high speed communication standard intended to be used with mobile devices. It roughly matches goals set for the 4th generation of mobile networks. The 4G is the most advanced mobile standard deployed, but it is set to be replaced by 5G in a near future.

Compared to the 3rd generation of mobile networks the LTE differs in its core network which is IP based and no longer relies on packet or circuit switching. The new design simplifies the network, provides some extra QoS guarantees, and enables low latency access.

The physical interface can handle fast moving clients. It uses wide range of frequencies and bandwidths that differ country to country. Also the media access method differs based on the region. In some regions Frequency Division Duplexing (FDD) is used, while in the other ones the Time Division Duplexing (TDD) is deployed. In general the bandwidths are in range $1 - 20$ Mhz while the expected throughput depends on the bandwidth and link quality. According to the [26], the maximum bandwidth is considered 300 Mibit/s and 75 Mibit/s for download and upload respectively. The lowest expected latency is around 5 ms.

5G

A 5G is a new communication standard for mobile networks. The most interesting features, with respect to the sCPS, are a very low latency, support for many simultaneous connections, device to device communication, and low power consumption. Currently there are some experimental networks deployed and some of the bands to be used are already assigned by the regulators, but the technology has not yet made it to the production deployment.

The 5G, as it is designed, represents a significant change compared to the current 4G networks. It promises very low latency in communication at the level of 1 ms. The throughput of the network should be as high as 1 Gibit/s, while the number of connected devices is supposed to increase significantly as many different sensors and IoT devices will connect. Many of the connected devices are expected to maintain an active connection. Apart from the device to infrastructure connections, it will be also possible to create device to device channels that will further reduce latency and throughput of the communication when concidering devices engaged in a direct communication.

The aforementioned challenges described above are about to be tackled using multiple innovative techniques described in [8]. The key is to increase spectral efficiency of the radio. The 5G radio will use spatial multiplexing, Multiple Input Multiple Output (MIMO), with tenths of simultaneous channels. The range of the cells in the network will be limited in dense areas where many femptocells or picocells, utilizing short range communication, will be deployed in order to maximize throughput. The network will use more advanced modulation that will enable higher bit rate in low noise environments. Finally new frequency bands will be introduced. The traditional mobile frequency bands will be fully utilized and some new frequencies around 30 GHz, offering extremely wide bands, will be used to boost short range communication.

2.4 Declarative Ensemble Specification

One of the ways of articulating ensemble definition is to declaratively prescribe the ensemble structure. An Ensemble Definition Language (EDL) is used to describe an ensemble declaratively as described in [27]. An example of EDL usage is displayed in Listing 2.3. The code shows simple leader-follower scenario with cars. The group specification, captured in EDL, is compiled into a Satisfiability Modulo Theories (SMT) problem and, at runtime, a SMT solver is used to group set of available components into ensembles. The approach described in [28] and [27] is well suited for centralized ensemble formation in a small group of components where the execution time of the SMT solver is not a limiting factor.

```
package Demo
data contract Car
  canLead: bool
  autonomous: bool
  posX : int
  posY : int
  goX: int
  goY: int
end

ensemble LeaderFollower
  id parking : Car
  membership
    roles
      leader : Car
      follower: Car
    constraints
      constraint leader.canLead or follower.autonomous
    fitness - sum distance(leader.posX, leader.posY, follower.posX, follower.posY)
  knowledge exchange
    follower.goX = leader.posX
    follower.goY = leader.posY
```

Listing 2.3: Example of system specification using an EDL.

2.5 Simulation Frameworks

2.5.1 ROS

A Robot Operating System (ROS) is a set of libraries and tools that help developers to design, deploy and monitor robotic applications. Despite its name the ROS is not an alternative operating system in sense of a new kernel or a new Linux distribution. It is rather a collection of user-space tools running on top of a standard Linux distributions.

The ROS architecture is based on passing messages between nodes using the publish-subscribe model. The runtime provides a network aware registry of topics while the nodes, running as separate processes, subscribe to the topics and post new messages. The robot controller consists of multiple nodes that communicate through the message passing. The nodes periodically publish their state and react to state changes from nodes of interest. In order to simplify deployment of nodes it is possible to define so called *launch files* that capture structure of the controller in XML. An example of a launch file that instantiates a Turtlebot controller is giver in Listing 2.4. The nodes initialized using the exemplified launch file encompass physical model of the robot necessary for visualization, movement related nodes, nodes forming navigation stack of the robot, and a node implementing Adaptive Monte-Carlo Localization (AMCL).

The message based interaction between nodes cab be exemplified on the interaction between wheel node and movement node. A node in charge of the wheels of the robot periodically send a message containing current state of the odometer. The movement controller node receives messages from wheel odometer topic, and messages from another node in charge of navigation. Based on the messages received the movement node sends messages to the wheels control node containing the desired state of the odometer.

The publish subscribe mechanism with periodic messages make the coupling between modules low. Moreover, the API is non-blocking thus the system can operate in a near real-time manner. The full real-time is not supported by the mainstream implementation, but custom proprietary implementations that support real-time are said to exist.

The ROS installation comes with many nodes implementing various functionality. Actually it is possible to put together a robot controller just by composition of the existing nodes. Also, many message types are already defined that cover most standard and even some obscure situations. As the community is encouraged to submit their custom message types described in a Domain Specific language (DSL) to the central repository the compatibility of different nodes is very high. When it comes to the implementation of a custom node the ROS offers bindings to C++, Python and Java.

The ROS comes with some graphical tools. The most interesting one is probably ROSViz that enable visualization of the robot including its surroundings. In particular the tool can be configured to display the model of the robot in 3D, positions of all its components, data from the sensors including LIDAR and depthcamera, environment map, output of the path planner, particles used by localization node, and many more different types of messages present in the system.

```
<launch>
  <arg name="name"/>
  <arg name="initial_pose_x" default="2.0"/>
  <arg name="initial_pose_y" default="2.0"/>
  <arg name="initial_pose_a" default="0.0"/>

  <group ns="$(arg␣name)">
    <param name="tf_prefix" value="$(arg␣name)"/>
    <remap from="map" to="/map"/>

    <!-- Robot Model -->
    <include file="\$(find␣turtlebot_bringup)/launch/includes/robot.launch.xml">
      <arg name="base" value="kobuki" />
      <arg name="stacks" value="hexagons" />
      <arg name="3d_sensor" value="asus_xtion_pro" />
    </include>
    <node name="joint_state_publisher" pkg="joint_state_publisher" type="
        joint_state_publisher">
      <param name="use_gui" value="false"/>
    </node>

    <!-- Move -->
    <node pkg="move_base" type="move_base" respawn="false" name="move_base">
      <rosparam file="params/local_costmap_params.yaml" command="load" />
      <rosparam file="params/global_costmap_params.yaml" command="load" />
      <rosparam file="params/dwa_local_planner_params.yaml" command="load" />
      <rosparam file="params/move_base_params.yaml" command="load" />
      <rosparam file="params/global_planner_params.yaml" command="load" />
      <rosparam file="params/navfn_global_planner_params.yaml" command="load" />
      <param name="global_costmap/global_frame" value="/map"/>
      <param name="global_costmap/robot_base_frame" value="/$(arg␣name)/base"/>
      <param name="local_costmap/global_frame" value="\$(arg␣name)/odom"/>
      <param name="local_costmap/robot_base_frame" value="$(arg␣name)/base"/>
      <param name="DWAPlannerROS/global_frame_id" value="/map"/>
      <remap from="scan" to="base_scan"/>
    </node>

    <!-- AMCL -->
    <node pkg="amcl" type="amcl" name="amcl">
      <param name="use_map_topic" value="true"/>
      <param name="odom_frame_id" value="/$(arg␣name)/odom"/>
      <param name="base_frame_id" value="/$(arg␣name)/base_link"/>
      <param name="global_frame_id" value="/map"/>
      <param name="initial_pose_x" value="$(arg␣initial_pose_x)"/>
      <param name="initial_pose_y" value="$(arg␣initial_pose_y)"/>
      <param name="initial_pose_a" value="$(arg␣initial_pose_a)"/>
      <remap from="scan" to="base_scan"/>
    </node>
  </group>
</launch>
```

Listing 2.4: A ROS launch script that launches a Turtlebot controller. The controller encompasses robot model, movement controller, path planner, and AMCL.

2.5.2 Stage

Stage[1] is a simple simulator of multiple robots in a 2D environment. If offers simulation of a basic robot movement dynamics and sensor readings for odometer and laser scanner. The simulated environment is a maze provided to the simulator in a form of a image. The simulator uses interface called Player to interface with the robot controller logic. There are ROS nodes that integrate with the Stage and Player in such a way that the robot controller logic can be implemented using ROS and the Stage provides the environment simulation.

The Stage simulator is rather old and simple, but still it can offer very good performance in cases where 3D simulation with advanced dynamics is not necessary. It is possible to simulate tenths or even hundreds of robots on a laptop computer.

2.5.3 Gazebo

Gazebo[2] is a modern robot simulator that offers many advanced features. Compared to the Stage it offers simulation of a 3D environment including many physical interactions and dynamics. It can simulate variety of sensors and different robots using provided sensor and robot databases.

The simulation of the robot takes in account the the physics of the robot and

[1]http://playerstage.sourceforge.net
[2]http://gazebosim.org

uses OpenDE[3], Bullet[4], Simbody[5], or DART[6] to simulate the dynamics of the robot. The simulation of the sensors includes noise and offers plugins that can implement new sensors. Gazebo also integrates with ROS so that it is possible to deploy a ROS based robot controller connected to the simulated robot.

The Gazebo simulator is much more advanced than the Stage one, but the advance comes at the cost of performance. Fortunately, it is possible to run the simulation in the cloud or on a dedicated server and access the simulated robots via the network.

2.5.4 MATSim

The Multi-Agent Transport Simulation (MATSim)[7] is a traffic simulator focusing on a large scale traffic simulations. It simulates movements of cars on a graph consisting of nodes representing intersections and edges representing roads. The MATSim is a discrete event simulator while the core of the simulation are events related to car entering and car leaving particular edge. On one hand, this simplification makes an exact position of a car within an edge an unknown. On the other hand this enables simulation of a very large systems consisting of thousands of cars covering whole cities.

One of the downsides of the MATSim is the assumption that the path of a car would not change. When dealing with local scope simulation of car to car interaction, sometimes, it is necessary to reroute a car or change its goal. Unfortunately the MATSim does not handle this well every time. Lack of stability when changing car paths and lack of control over car at the edge level makes MATSim a bit less usable for small scale simulations focusing on a precision of movement in lanes.

The MATSim is quite easy to setup using Maven artifact. In order to run the simulation a configuration file, exemplified in Listing 2.5 and a road network graph is needed. The OpenStreetMap can be used to obtain realistic data that can be easily converted to form such a graph.

```xml
<config>
  <module name="network">
    <param name="inputNetworkFile" value="input/prague.xml" />
  </module>
  <module name="controler">
    <param name="writeEventsInterval" value="1000" />
    <param name="writePlansInterval" value="1000" />
    <param name="eventsFileFormat" value="xml" />
    <param name="outputDirectory" value="matsim"/>
    <param name="firstIteration" value="0" />
    <param name="lastIteration" value="0" />
    <param name="mobsim" value="qsim" />
  </module>
  <module name="qsim" >
    <param name="startTime" value="00:00:00" />
    <param name="endTime" value="00:10:00" />
    <param name="flowCapacityFactor" value="1.00" />
    <param name="storageCapacityFactor" value="1.00" />
    <param name="numberOfThreads" value="1" />
    <param name="snapshotperiod" value="00:00:01"/>
```

[3]http://opende.sourceforge.net
[4]http://bulletphysics.org
[5]https://simtk.org/projects/simbody
[6]http://dartsim.github.io
[7]http://www.matsim.org

```
    <param name="timeStepSize" value="00:00:0.2" />
    <param name="trafficDynamics" value="queue" />
  </module>
</config>
```

Listing 2.5: Example of a MATSim configuration

2.5.5 SUMO

The Simulation of Urban MObility (SUMO)[8] [29] is traffic a simulator focusing on a microscopic scale traffic interactions. Similarly to MATSim it uses a graph or roads and intersections as an input of the simulation, but the edges are detailed down to the level of lanes and intersection segments. The execution model of SUMO is continuous, thus the car to car interaction is simulated while taking car acceleration and break capacity into account. Particular car types can be defined and used in the simulation as seen in the example given in Listing 2.6. Due to the continuous mode, also the exact position of every car is available at any moment.

```
<routes>
  <vType id="citigo" accel="2.14" decel="4.23" sigma="0.5" length="3.56" minGap="
      2.5" maxSpeed="13.88" color="0,0,1" guiShape="passenger"/>

  <flow id="chotkova" type="citigo" from="8135" to="16234" probability="0.2" />
  <flow id="vitezna" type="citigo" to="8135" from="16234" probability="0.2" />
  <flow id="uvoz" type="citigo" to="25178" from="21357" probability="0.2" />
  <flow id="vlasska" type="citigo" from="27020" to="5178" probability="0.06" />
</routes>
```

Listing 2.6: Example of a SUMO car flow specification. Note details of a car type specification.

The nature of the simulator enables control of a simulated vehicles without interfering with the simulator logic. The SUMO implements Traffic Control Interface (TraCI) that enables on-line interaction with the simulation using a Transmission Control Protocol (TCP) connection to the simulation server. The precision of the simulation comes at the cost of higher demand on the computation power compared to the MATSim. Regarding the scenarios necessary evaluate sCPS this limitation is not a serious problem.

In order to run SUMO simulation, it is necessary to have a graph representing a street network, SUMO configuration, and a description of car flows. The street graph is a large XML file that can be generated using provided tools from data extracted from the OpenStreetMap. The SUMO configuration file is exemplified in Listing 2.7 and the car flow description is exemplified in Listing 2.6.

```
<configuration>
  <input>
    <net-file value="prague.net.xml"/>
    <route-files value="prague.rou.xml"/>
    <additional-files value="prague.poly.xml"/>
  </input>
  <time>
    <begin value="0"/>
    <end value="100000"/>
    <step-length value="0.1"/>
  </time>
  <gui_only>
    <start value="true"/>
  </gui_only>
```

[8] http://www.sumo.dlr.de

```
</configuration>
```

Listing 2.7: Example of a SUMO configuration

2.5.6 OMNeT++

OMNeT++ is a discrete event network simulator written in C++. Its modular design enables easy composition of different modules, implementing parts of the virtual network infrastructure and simulated application, into a complex simulation. OMNeT++ uses NED language, exemplified in Listing 2.8, to define modules the simulation consists of. Also NED language is used to bind multiple modules into a network being simulated. Modules, the basic components of the OMNeT++ simulation are declared using the NED and defined by C++ classes. The modules exchange messages some of which simulate packets being delivered. The messages representing packets are subject to packet loss, congestion, and interference simulated by various modules. Each module can describe parameters and gates (connectors) to the other modules that can be set by the NED or specified in the simulation configuration file. The OMNeT++ offers very little functionality on its own. The true power of it comes with libraries that provide implementation of different network types.

```
package cz.cuni.d3s.parkingplacescanning.modules.application;
import org.car2x.veins.modules.application.ieee80211p.BaseWaveApplLayer;

simple ParkingPlaceScanningApp
{
    @class(ParkingPlaceScanningApp);
    @display("i=block/app2");

    gates:
        output toDecisionMaker;
        input  fromDecisionMaker;
}
```

Listing 2.8: Example of a simple OMNeT++ module description in a DSL

At runtime the OMNeT++ offers multiple user interfaces that encompass a command line interface suitable for batch processing of the simulations and a graphical user interface suitable for debugging and detailed packet inspection. The graphical user interface used to be based on the Tcl/Tk[9], but the latest version offers also Qt[10] based version.

2.5.7 INET

INET[11] framework is an OMNeT++ extension that implements the most common network technologies. It offers support for basic wired and wireless physical layers, as well as most common link layer and network layer protocols.

As of the INET version 3.4.0, according to the modules defined in the source code, the following are supported:

Radio interfaces

- IEEE 802.15.4

[9]https://www.tcl.tk

[10]https://www.qt.io

[11]https://inet.omnetpp.org

- IEEE 802.11

Link layer protocols

- IEEE 802.3 Ethernet
- IEEE 802.15.4
- IEEE 802.11
- IEEE 802.11d
- Point-to-Point tunnels
- Berkeley MAC (B-MAC)
- Lightweight Medium Access Protocol for Wireless Sensor Networks (L-MAC)
- Tunnel interface (tun)

Network layer protocols

- ARP
- ICMP
- IPv4
- IPv6
- Probabilistic broadcast
- WiseRoute

2.5.8 Veins

Veins[12][30] in an OMNeT++ extension that provides precise simulation of the car to car communication. It employs models of the low level network that enable accurate simulation of interference and shadowing by objects. It also provides extension to the OMNeT++ graphical interface that shows current position of cars, packets, and static obstacles. A screenshot of the Grahical User Interface (GUI) is displayed in Figure 2.4. The simulation of the car movement is provided by SUMO, thus the road network can be generated from the OpenStreetMap data. The OpenStreetMap data can also be used to define static obstacles based on building polygons. Thanks to this compatibility the scenario setup is quite a simple task.

Veins LTE[13][31] combines original Veins project with another OMNeT++ extension called SIMUlte[14]. Using Veins LTE it is possible to combine car to car communication with LTE mobile broadband connections in one simulation. Thus simulation of heterogeneous systems combining low latency car to car communication links with long range mobile broadband connections is possible as depicted in the screenshot provided as Figure 2.4.

[12]`http://veins.car2x.org`
[13]`http://veins-lte.car2x.org`
[14]`http://simulte.com`

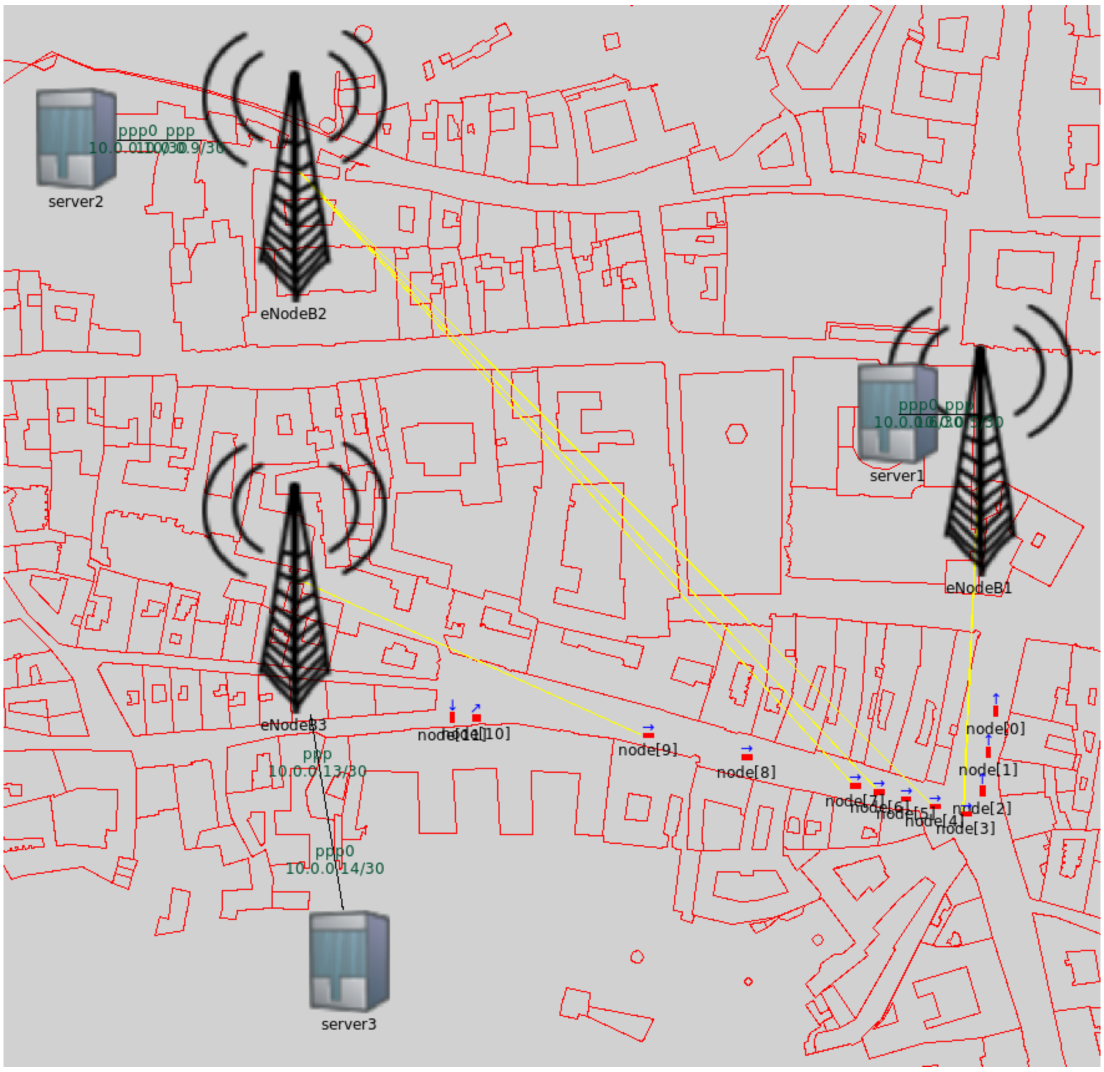


Figure 2.4: An example of a graphical output of the Veins simulation. Cars cooperate using LTE connection in the streets of the Lesser Town of Prague. Red rectangles with blue arrow represent cars and their direction of movement. Yellow beams represent ongoing wireless transmission.

CHAPTER 3

Goals

3.1 Challenges

This section describes technical and design challenges following from motivation examples and use cases described earlier in the text. It is necessary to mention that most of these challenges are easy to solve in a centralized system with a good network connection to all its parts. Unfortunately, motivation examples given above are supposed to work in a quite different environment. Open-ended, dynamic systems with loose, mostly wireless, connections and unpredictable number of actors, which can fail at any moment, represent the biggest challenge that needs to be addressed. Related challenges fall into four fields described below. In particular, these are: (i) challenges related to dynamic formation of a group of entities on top of which the coordination and data sharing shall be executed; (ii) data exchange performed on top of a dynamic group while using unreliable communication media; (iii) coordination of loosely connected entities; and (iv) the challenge of experiment design and evaluation.

3.1.1 Membership Decision

When dealing with dynamic coordination and data sharing in distributed systems, the basic question is how to define a group of entities that is supposed to cooperate or share the data. The options are many, for instance it is possible to statically define a table containing identifiers of entities that are supposed to be in a particular group. That would be practical from the implementation point of view, but almost useless in a dynamic system where new entitles in need of coordination can emerge at any moment. It is much more practical to describe a membership condition for the group and than check whenever the condition is met. i.e that two robots coordinate when they are close, if they have the same color, or posses tools necessary to perform a requested operation.

This approach expects that some information about the entities is already present in the system. i.e. when the condition is based on a mutual distance of the robots, then the system needs to share the information containing the location of all the robots in order to evaluate the condition. In the case of a centralized system, this can be implemented by sending such data to a well chosen central server which evaluates the condition. Due to network latency, low throughput, and reliability constraints it might be necessary to adopt a distributed operation scheme, where the condition is evaluated on multiple cooperating nodes. In such a case a proper dissemination of data necessary to evaluate the condition is required.

Such requirement on data propagation brings an interesting challenge. As the number of entities in the system is open, the number of nodes that needs to have data necessary for membership decision of every other node may grow beyond manageable limits. In fact it is impossible to share the information everywhere in any non trivial system. Instead, it is necessary to find a good approximation of the membership condition and propagate information on particular entity in the system only where necessary. For instance, if the condition describes pairs with distance less than threshold, then it makes no sense to propagate positions of entities further than the threshold distance is. Even though it seems simple in the exemplified case, a more general or even completely universal approach is a significant challenge.

Another challenge is the quality of the resulting coordination of data sharing groups. As the propagation of the data necessary to evaluate membership is limited, some groups are much easier to manage than others. Regarding the distance example, the group that has a condition of mutual spatial distance is much easier to manage than a group the requires its members to be scattered across the largest area possible. The challenge is to overcome those problems or at last identify conditions that cannot be used due to data propagation limits.

These limits are caused by a lack of connectivity that tempers with ensemble formation. This is even a bigger issue when dealing with coordination that requires some time to be performed. The system in question needs not only to evaluate the condition, but also to check whenever the condition necessary to maintain the coordination effort is met. Moreover, a deficiency in a connectivity of the entities that are about to be coordinated can influence the coordination quality or make the effort much less efficient. In such situations it may be necessary to brake the already established group and form a new one that exhibit better functionality. Detection and resolution of such conditions is the final challenge in membership decision.

3.1.2 Data Exchange in a Loosely Connected Ensemble

Once the group of cooperating entities is established it is necessary to perform the coordination or the data exchange among the group members. Considering the data exchange, again the network is the source of problems. If a wireless connection is considered a consistency in data delivery cannot be guaranteed. Therefore hence, the system needs to deal with lost packets and lack of bandwidth. The membership decision process, as described in Section 3.1.1, is supposed to break the data exchange group that is malfunctioning, but until this happens the data exchange mechanism is required to continue and optimize itself for the new network conditions.

On one hand, when dealing with centralized data exchange, where data are first transported to a central node that does all the processing, the consistency is not an issue. On the other hand, once the data resulting from centralized data sharing are used on distributed nodes, or when data exchange is performed completely as a distributed operation, then the consistency is a significant challenge. Even when consistency is not required every time, most of the motivating examples of this thesis require at last some degree of data exchange consistency.

3.1.3 In Ensemble Coordination with Packet Loss

The same problem as the one in data exchange consistency arises in coordination. Again, when considering action coordination using an unreliable media, then a packet loss is a significant issue. The challenge is to design an architecture of the system in such a way that the resulting system is resilient to network problems or at last it exhibits a gradual tear-down where possible.

Basically, there are two paths to follow. First is to find a suitable way how to centralize the coordination. Once the coordination decisions are made on one node it is possible to enforce consistency.

Second is to maintain fully distributed operation but employ proper algorithms to ensure consistency. Then the proper choice of abstraction is critical at architecture level so that the abstractions provide enough details for algorithms to work, but hide all the unnecessary complexity from the system designer.

3.1.4 System Evaluation Challenges

When dealing with a open-ended distributed and dynamic systems, such as those described in the motivation, it is necessary to develop a proper evaluation strategy. Dealing with the systems that are yet about to be implemented, it is almost impossible to judge whenever the usage of the particular architecture in reality was a success or failure. Even in cases where that would be possible, there is usually no similar legacy system to compere to, as the technology enabling creation of such systems emerged recently.

Natural option is to implement a pilot that would show benefits of the system in a small scale, but the true nature of those systems unfolds only if deployed in large numbers. In example, the true nature of a distributed open-ended system unfolds when a large number of entities join the system and possibly overload the network or expose too much data so that a coordinator of a group cooperation is congested. This condition is never met when the system, build in a laboratory conditions, encompasses only a pair of robots. Due to this a challenge arises to implement a suitable simulation of the system that enable evaluation of the architectural and other approaches taken during the system design with respect to a massive scale deployment.

A related challenge is to properly simulate the environment. Actually, the unpredictability and the complexity of the environment is the limiting factor of the system. Once network latency is not simulated or effects of physics are omitted, the system that would break in the reality can function flawlessly in the simulation. This issue needs to be addressed with detailed enough simulation of both the networking and the physics.

3.2 Problem Statement

With regard to the motivation described in Chapter 1 one of the unanswered challenges is applicability of the current architectural approaches to the sCPS with respect to a realistic network layer. In particular, the systematic solution is necessary for dealing with dynamic groups formed on top of unreliable and low capacity network such as Wi-Fi or ZigBee. More problems arise when the application requires real-time reaction times or distributed operation is required. In the latter case, when the dynamic group cannot rely on centralized components, the system architecture needs to let the system tailor itself towards the new properties of the network, overall system state, and mutual positions of the group members in order to provide sustainable performance and reliability.

When focusing on the EBCS and, in particular, the DEECo, the architectures currently used to capture sCPS, the main shortcoming is a lack of optimization towards the network properties such as routing, latency and range. While these architectures provide very nice abstractions for systems with good connectivity the abstractions need to be tailored towards the system that rely on an ad-hoc networking and a distributed operation. Moreover the challenge of developing architectures tailored towards complex networks requires also capability to properly evaluate the resulting design. This is even more challenging with respect to the fact that the system operation depends on the number of components, while the real systems can encompass very large numbers of components (automotive, robot coordination). This brings necessity to properly simulate both the target system and the network used to run it. The following paragraphs summarize the challenges in more detailed and structured way.

C1: *Data propagation limits.* Limiting data propagation is necessary to maintain reasonable network traffic and requires to introduce an ability to decide where to spread the data and how to do it with respect to the properties of the underlying network. This includes choice of a proper network structure and some sort of approximation of the membership function used to decide where the data are needed to establish ensembles. In particular, it is important to decide which routing protocols will be used to spread the data.

C2: *Networking in sCPS.* With respect to the requirements of the sCPS it is important to decide on network layer(s) used to propagate the knowledge necessary to decide ensemble membership and perform in-ensemble coordination or knowledge sharing. Also it is important to decide what technologies will be used to share the data, ranging from distributed knowledge sharing to centralized tupple-spaces. The whole approach needs to take in account realistic properties of the network such as a latency, congestion, and throughput.

C3: *Ensemble quality with respect to limited data.* It is important to evaluate quality of ensembles with respect to the underlying networking technology and their membership function. The properties of realistic network such as limited throughput, latency and packet loss may fail attempts to establish the ensemble, due to limited propagation of the knowledge, or make ensemble decision process inconsistent due to delays and losses in distributed ensemble formation.

C4: *In ensemble coordination and knowledge sharing consistency.* Similar to consistency of membership condition, defined in Challenge **C3**, the consistency of knowledge exchange and, in particular, in-ensemble coordination is a great challenge as the distributed operation may be enforced by requirements on real-time processing and system resiliency.

C5: *Proper evaluation of sCPS design.* Evaluation of the architectures used design dynamic and open-ended systems is a challenging task. It is necessary to ensure that the desired system work not only in abstract simulation of its interaction, but also to make the testing conditions as much realistic as possible with respect to the deployment scale and realistic simulation of the environment. Moreover, it is necessary to introduce tools, easy to use by other researches, that enable further integration of developed technologies and approaches.

3.3 Research Questions and Goals

With respect to the architectures described earlier in the text, namely EBCS and, in particular, DEECo based systems, this thesis asks the following research questions and sets goals in order to address the challenges defined in Section 3.2.

Q1: How can data propagation in sCPS be limited in such a way that network congestion is prevented and the system maintains high utility?

G1: *Smart limits in knowledge propagation.* Analyse knowledge propagation requirements in domain of sCPS and rise related properties of the network to architectural level in order capture those requirement at design time. This addresses challenge **C1**.

Q2: What are the effects of network limitations on distributed ensemble formation? How can the design of ensemble formation be changed in order to avoid negative effects?

G2: *Distributed ensemble decision.* Design methods of ensemble formation in distributed environment with respect to realistic network properties and requirements native to the target domain of sCPS such as reaction time, resiliency and dynamicity. This addresses challenges **C2**, **C3**, and **C4**.

Q3: How can the design of sCPS be evaluated in terms of network awareness?

G3: *Smart Cyber-Physical Systems test-bed.* Deliver an easy to use test-bed for experimenting with sCPS systems that can be reused by other researches in the field. The test-bed will include precise simulation of the network and physical environment so that the algorithms can be tuned in it and later used in reality with just a little effort. This addresses challenge **C5**.

3.4 Contributions

The work presented in this thesis can be divided into several areas that loosely match goals described earlier in the text. These areas cover work on DEECo architecture, experiments with simulations, and creation of a test-bed.

First, the data propagation was tackled in terms of replacing the limited propagation of knowledge data with a new concept of groupers that enable smarter coordination over long distances. This work is presented in [1]. The data propagation work was also addressed in [4] that accesses security and trust in DEECo based system and focuses on data propagation safety. The work on low level networking was concluded by creation of CDEECo++ framework and publishing of a paper on application in safety critical systems [5].

Later the concept of intelligent ensembles [28] was considered when dealing with network stack. The data propagation and ensemble formation challenges arising from more structured intelligent ensembles were tackled in a work published in [7].

Finally in [2] the QoS and safety was considered in a EBCS based Industry 4.0 system and performance modeling in the EBCS was briefly discussed in [3].

Reviewed Articles

[1] M. Kit, F. Plasil, V. Matena, T. Bures, and O. Kovac, "Employing domain knowledge for optimizing component communication," in *Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering*, ser. CBSE '15, Montréal, QC, Canada: ACM, 2015, pp. 59–64, ISBN: 978-1-4503-3471-6. DOI: 10.1145/2737166.2737172. [Online]. Available: http://doi.acm.org/10.1145/2737166.2737172.

[2] V. Matena, A. Masrur, and T. Bures, "An ensemble-based approach for scalable QoS in highly dynamic CPS," in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Aug. 2017, pp. 234–238, ISBN: 978-1-5386-2141-7. DOI: 10.1109/SEAA.2017.62. [Online]. Available: https://dx.doi.org/10.1109/SEAA.2017.62.

[3] T. Bures, V. Matena, R. Mirandola, L. Pagliari, and C. Trubiani, "Performance modelling of smart cyber-physical systems," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18, Berlin, Germany: ACM, 2018, pp. 37–40, ISBN: 978-1-4503-5629-9. DOI: 10.1145/3185768.3186306. [Online]. Available: http://doi.acm.org/10.1145/3185768.3186306.

[4] O. Štumpf, T. Bureš, and V. Matěna, "Security and trust in data sharing smart cyber-physical systems," in *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ser. ECSAW '15, Dubrovnik, Cavtat, Croatia: ACM, 2015, 18:1–18:4, ISBN: 978-1-4503-3393-1. DOI: 10.1145/2797433.2797451. [Online]. Available: http://doi.acm.org/10.1145/2797433.2797451.

[5] A. Masrur, M. Kit, V. Matěna, T. Bureš, and W. Hardt, "Component-based design of cyber-physical applications with safety-critical requirements," *Microprocessors and Microsystems*, vol. 42, pp. 70–86, 2016, ISSN: 0141-9331. DOI: 10.1016/j.micpro.2016.01.007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933116000107,

Impact Factor: 1.025, CiteScore: 1.11, SCImago Journal Rank: 0.238

Statistics captured on 05/09/2018 from https://www.journals.elsevier.com/microprocessors-and-microsystems.

[6] V. Matena, T. Bures, I. Gerostathopoulos, and P. Hnetynka, "Model problem and testbed for experiments with adaptation in smart cyber-physical systems," in *Proceedings of the 11th International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '16, Austin, Texas: ACM, 2016, pp. 82–88, ISBN: 978-1-4503-4187-5. DOI: 10.1145/2897053.2897065. [Online]. Available: http://doi.acm.org/10.1145/2897053.2897065.

[7] T. Bures, P. Hnetynka, F. Krijt, V. Matena, and F. Plasil, "Smart coordination of autonomic component ensembles in the context of ad-hoc communication," in *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques: 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10–14, 2016, Proceedings, Part I*, T. Margaria and B. Steffen, Eds. Cham: Springer International Publishing, 2016, pp. 642–656, ISBN: 978-3-319-47166-2. DOI: 10.1007/978-3-319-47166-2_45. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-47166-2_45.

Implemented Frameworks and Test-beds

In order to perform experiments and validate overall usability of the approach described in this thesis several tools and frameworks were created or extended. First, the JDEECo framework was significantly extended in terms of the network support and integration with the network and car traffic simulators. The work covers implementation of a flexible scheduler and software part of a network stack for the IEEE 802.15.4. Second, the CDEECo++ embedded runtime was created in order to verify applicability of DEECo to hard real-time systems with network implemented by the IEEE 802.15.4 radio. Third, the PyDEECo framework was implemented to enable quick drafting of voting (and other) protocols used to establish and maintain ensembles. Finally, a test-bed was created using the JDEECo framework that enables experiments with robotic vacuum cleaner coordination and their adaptation. The software implemented to support this thesis is further discussed in Chapter 11.

CHAPTER 4

Use cases

In order to base this thesis on realistic foundations, first the envisioned use cases are analyzed in this chapter. As described earlier in the motivation, this work sources its use cases from multiple domains. These are robotics, Industry 4.0, and semi-autonomous car or fully-autonomous car cooperation problems. Choice of the use cases is motivated by recent research projects connected with EBCS and in particular DEECo such as (i) EU 7th Framework Programme Ascens[1] [32] where robotic systems and smart navigation of cars was the topic of interest; (ii) industry oriented project with Volkswagen AG focusing on smart navigation; and (iii) projects with industrial partners focusing on IoT and Industry 4.0 systems: Establish[2] and Trust[3].

Analysis of the use cases focuses on the coordination and cooperation using dynamic groups based on ensembles. A special focus is applied on ensemble membership and knowledge exchange with respect to the data propagation between the components.

This chapter is organized as follows. First the use cases arising from robotic swarm coordination are briefly discussed. Then the coordination of robots in Industry 4.0 environment is mentioned. Finally, broad autonomous driving use cases are given. The reason to focus specially on autonomous driving is twofold.

First the patterns found in Industry 4.0 and robotic coordination are very similar to the ones found in the autonomous driving domain. At the level of software architecture it makes almost no difference to coordinate a swarm of flying drones, a group of mobile industrial robots, or just cars in a street. In all these cases the focus is on real-time communication and data propagation ensuring collision avoidance and cooperation on a common goal in the group. Looking at the tasks such as scouting an area, optimizing part delivery, or preventing street congestion these are in fact very similar. All these share dynamic identification of a target group, spreading information among its members and some sort of coordination based on the group shared knowledge.

Second reason to focus on autonomous driving and traffic related problems is the availability of tools and input data. There are tools for dealing with robots such as ROS, Gazebo, and Stage. The problem is that these are focusing mostly on single robot simulation and simulation of low level robot controls. Even when the tools for multi robot simulation are getting available the realistic data for the

[1]`http://www.ascens-ist.eu`
[2]`https://www.vtt.fi/sites/ESTABLISH`
[3]`http://trust40.ipd.kit.edu`

environment and realistic task settings are rare. On contrary, in the autonomous driving and traffic optimization domain the tools are ready to be used. Simulators such as SUMO or MATSim are easy to run and provide sufficient level of precision to analyze properties of dynamic groups of cars. Moreover, the source data such as road maps and traffic density are available. Road maps can be imported from OpenStreetMap and processed as necessary. The traffic density data are easy to obtain. For example even real-time data can be obtained from Ito World[4]. Sometimes data are available for free directly from the government of a particular region. This is the case of United Kingdom where the Department for Transport provides traffic counts[5].

Motivated by the similarity of the problems, availability of practical tools, and data for traffic analysis the autonomous driving use cases were chosen as representative for the whole group. Thus the autonomous driving use cases are presented in much more detail. The use cases described in this chapter are then analyzed in the following Chapter 5.

4.1 Robotic Swarm

An interesting use case of a dynamic group in the world of robots is creation and maintenance of a particular swarm shape in a space. This is the behavior that the UAVs learned from birds, UGVs from pack hunters, and Unmanned Underwater Vehicle (UUV) from fish. In difference from the most of animals the unmanned vehicles usually has some other interest in maintaining the swarm than defense against predators. A usual task for a swarm is to scout an area in an efficient way as described in following scenario.

4.1.1 Search and Rescue

Scenario RS1 The scenario is a search and rescue mission in a city after an earthquake. The mission is conducted by hundreds of UAVs equipped with infrared cameras searching for survivors to be extracted from the debris. Each UAV is equipped with a packet radio device with limited range and a localization device, i.e. GPS. The target search area is initially known to all the UAVs.

The task is to spread UAVs across the area in such a way that the area scan is completed in shortest possible time. Static assignment of subareas to particular UAV is not a good option as different areas differ in scan difficulty and UAVs can fail or new ones can be introduced. Instead the UAVs can be seen as an ensemble of components that interact in order to dynamically resolve the situation.

The components needs to coordinate locally on collision avoidance and fine grained subarea assignment. The global coordination encounter at last rough area assignment to components necessary for top level planning. In order to perform the global coordination some sort of overlay network that ensure global data delivery needs to be employed.

[4]`http://www.itoworld.com`
[5]`http://www.dft.gov.uk/traffic-counts`

4.2 Industry 4.0 Robot Coordination

Another interesting use cases emerge in domain of Industry 4.0. Dynamic grouping using ensembles make very good sense in a dynamic factory operated by both robots and human workers at the same time. From a high level point of view a factory is a set of workers, robots, tools and skills that needs to be combined together in order to manufacture a product. At last two problems in a Industry 4.0 factory can be tackled using ensembles. The first and most important is to enforce safety by avoiding collision among workers and non-stationary robots. The second one is a dynamic grouping of assets and tasks in order to form a group of workers able to complete particular task.

4.2.1 Collision Avoidance

Scenario FS1 The scenario for collision avoidance is a large factory hall occupied by both mobile robots and human workers. There is an indoor localization system installed in the factory that provides location and time base to the robots. The workers wear suits with embedded controller that senses its location and maintain synchronized clock. All the robots and the worker suits are also equipped with a packet radio such as Wi-Fi or IEEE 802.15.4.

Objective is to maintain safety at workplace by limiting robot movement and speed limits so that, possible tragic, collisions do not occur. The system needs to dynamically react to new robots and human workers as well as react to congestion, limited radio range and packet loss. The system also needs to react in real-time as the dynamics of robots and humans does not allow delays in evaluation of the situation.

4.2.2 Task Group Formation

Scenario FS2 The second Industry 4.0 scenario differs from the first one in being oriented on a complex dynamic group structure rather than on real-time processing of data coming through a possibly unreliable channel. The scenario also encompasses robots and humans, but now instead of a precise position an ability to perform a particular task is of the utmost importance.

The factory receives tasks that needs to be completed. The tasks needs to be matched against schedule and ability set of workers and robots and combined into a feasible schedule. In language of ensembles robots and workers capable of performing particular task meet the ensemble membership condition. The group serve two purposes, first it defines feasible group of workers able to complete the task, second it is a platform for coordination of these workers in order to actually perform the sub-tasks necessary to finish the assigned task.

4.3 Autonomous Driving

Autonomous or semi-autonomous driving is the main source of scenarios for this thesis. Autonomous driving problems related to dynamic grouping of cars and road side infrastructure encompass coordination and information sharing in common traffic. Surprisingly these problems seem to encompass situations similar to the aforementioned robotic swarms and Industry 4.0 problems. As swarm coordination seems to be quite similar to coordination of group of cars on a highway and collision avoidance system in Industry 4.0 factory is quite close on street collision avoidance.

The use case scenarios are organized into two groups. First one encompasses scenarios that are based on a local communication. These are inspired mostly by coordination performed by human drivers, thus in this scenarios the sCPS are learning from human drivers. The second group is formed by scenarios that at last partially rely on a long range communication. These scenarios go beyond standard situations in the traffic and open new possibilities for traffic improvements.

4.3.1 Environment

The common setup for vehicular scenarios is as follows. Semi-autonomous cars and fully-autonomous cars operate together with standard cars in city traffic. Naturally the cooperation between standard and autonomous cars is limited to the bare minimum defined by the traffic rules. The autonomous cars are expected to have the following equipment:

- GPS providing approximate location and precise time source.
- Sensors and controllers implementing self-driving ability conformant to traffic rules. The self driving controller needs to have an interface that enable obtaining its state and intended actions as well as providing advises on how to solve a particular situation.
- Short range low latency radio such as IEEE 802.15.4 or WAVE intended for local data sharing and coordination.
- Long range communication device providing Internet connectivity such as LTE or 5G.
- Controller capable of maintaining ensembles and performing related tasks as described later in the text.

Providing that the autonomous cars are equipped with the aforementioned devices and systems the common task of all scenarios described later is to help improve the traffic and solve traffic related problems. The envisioned systems are supposed to optimize behavior of cars, but do not guarantee safety. Safety of operation needs to be handled independently by the self-driving controller that only takes advises from the sCPS based on components and ensembles running on an independent controller.

4.4 Local Vehicle Coordination

Scenarios is this group are based on a coordination that is or could be performed by human drivers. The motivation is that a human driver even when it is usually not required by general traffic rules, also takes coordination hints from other drivers. These hints are not only based on movements of cars that can be read by on-board sensors of autonomous vehicles. In fact, human drivers use gestures, signaling by lights and horns, guesses on the intentions of the other drivers and even judge the other driver based on way he or she drives.

Of course the autonomous cars cannot take part in a hand gesture argument at the unregulated crossing. But it is possible to take some of the principles and implement them on top of a radio communication. In general the cars can publish their intends like changing lane, parking, turning and others using radio and the other cars in the area can adjust their behavior in order to optimize the traffic. The communication can go beyond this and other cars can also respond to the request and for instance agree to let the car intending to change a lane actually do that.

As the human drivers are still assumed to remain in control in some/most of the cases, some of the coordination performed automatically by autonomous cars can be simplified back to the actions like blinking lights, so that the human drivers can take place in this kind of coordination. Of course the more complex scenarios do not allow this, but in fact the less complex ones are already implemented. For instance the turn signals can be seen as a simple sharing of the intend to turn that is unfortunately occasionally not followed by mutual agreement between drivers.

4.4.1 Scanning for a Parking Place

Scenario VS1 A car is about to reach its destination and needs to find a parking place. As there is no global parking place registry nor there are organized parking houses nearby it has to rely on a surroundings scan in order to find a free parking spot. This is inefficient in terms of fuel consumption, time spend, final parking position location, and travel speed during the search.

In fact this can be improved if other cars in the area that are just passing by can help with the effort. The parking car can signal its intend to the other cars in the area and the other cars can initiate the surroundings scan and report possible positive results back to the car in the need of parking. This way the cars can not only cover larger area, thus being able to find a parking place quickly, but also the location of the parking place can be better as the other cars cover also streets different from the one chosen by the parking car to approach its target location. Providing that the sensor range of the cars is short the parking car cannot travel at full speed as it needs to be able to stop at suddenly discovered free spot. If there are enough cars capable of scanning in front of it these can take role of a long range sensors, thus the parking car can continue at the full speed. Optionally, some cars can be re-routed to cover the less frequent streets and thus scan a larger area.

Such system requires dynamic grouping of parking and passing by cars based on their location, capabilities and directions. The system needs to react to connection quality among the cars and current street coverage in order to maintain quality of the search service.

4.4.2 Precedence Negotiation

Scenario VS2 A scenario for precedence negotiation is a narrow passage on a bidirectional road where only one car can fit. Another possible setup is the unregulated crossing. In both these cases the cars need to decide which one goes first. The decision is based on traffic rules and in case of human drivers also on the gestures. Gestures are defined by traffic rules to be used when a stalemate is reached. For instance this happens when 4 cars meet at the x-shaped crossing one in each direction. The gestures are also sometimes used to optimize the traffic. Some drivers pass precedence to others that would otherwise starve if the traffic rules would have been applied literally. Finally the gestures can be used to resolve emergency situations where no standard rules apply. This happens when cars needs to move across the sidewalk to avoid collapsed road.

All these situations require some sort of coordination among cars that resolves the precedence order. This is where dynamic groups or ensembles can be used. The system needs to be dynamic to respond to new cars emerging, resilient to network failures and latency as some cars can leave or get far from others, and distributed as the situation might take place somewhere where the infrastructure is not available.

The cars needs to resolve the precedence order and signal the self driving controller with the resulting advise. The controller is responsible for performing the action safely, it cannot rely on the negotiated precedence being honored by others. The advise given by precedence resolution system is intended to optimize traffic by reducing necessity to break and starve.

4.4.3 Traffic Lane Change Negotiation

Scenario VS3 Lane change scenario consists of cars going on a highway with multiple lanes. One car decides to change the lane due to necessity to exit the highway or just to overtake a slowly moving truck. If there are no cars around it is not a problem, but it often happens that there is little or no space in the target lane. Performing the lane change can be difficult if the car in need of a lane change is restricted in movement until the current lane ends or it misses an exit.

Human driven cars use signaling, like turn signals, and guesses to realize there is a driver in need to change the lane. Once the drivers are aware of the situation they can take an action to improve the fluency of the traffic such as making a space in a lane so that the other can can fit into it. Similarly the autonomous cars can form a dynamic group in form of an ensemble with components taking roles of cars. The component representing a car that is preventing another car from changing lane can slow down to create a gap and signal the component representing the lane changing car that is will be allowed to change the lane. Finally the self driving controller in the blocking car can be asked to slow down while the one in the lane changing car is advised to move into the resulting gap.

Moreover this scenario can be extended to the situations such as the zip maneuver where two lanes merge into one and the cars are expected to alternate. With the ensemble containing logic to implement advises on a final ordering of the cars the maneuver itself can be much smoother and safer as the order does not change at the very last moment.

4.4.4 Platooning on Highways

Scenario VS4 This scenario focuses on the creation and management of highway platoons on a local level. It expects a highway occupied by cars where some of them are capable of autonomous movement in a platoon. The platoon is basically a train of cars that maintain a constant speed and distances between neighboring cars in order to achieve some benefits. The benefits include air drag reduction leading to lower fuel consumption and an ability to drive autonomously for cars that are not capable of doing so in general traffic. Also, the creation of platoon simplifies traffic management as a platoon can be considered a single object both in the logical evaluation of the traffic and the maneuver planning such as coordinated acceleration.

The cars that are about to form a platoon need to group together based on the rules captured by the ensemble membership. Once the platoon is established the members of the platoon need to group again in order to coordinate on the platoon management such as the speed and mutual distances. Both grouping steps needs to be flexible and aware of the dynamics of both the situation and the environment such as cars leaving/entering the area and a road surface quality changes that needs to be reflected in the platoon organization.

Apart from the scenarios described earlier, implementation of this one cannot rely on advises to the self driving controller. Instead the platoon management needs to be hard real-time and dependable as it needs to guarantee safety of the operation. The cars cannot drive close enough to each other without continuous coordination as there is not enough time for the car in the back to react to an arbitrary move of the car in the front.

4.4.5 Optimized Speed-up in Traffic Jams

Scenario VS5 The coordinated speedup scenario is similar to the platoon one. In a sense it is some sort of a short lived platoon that is established only for the purpose of coordinated speedup at the traffic lights. The setup is the queue of cars waiting at the traffic lights. The problem is the situation when the go signal is given. The first car can accelerate without restrictions, but the following cars need to wait for a gap to emerge in front of them thus limiting the throughput of the crossing.

The task is to coordinate the cars waiting to form blocks of neighboring cars that can coordinate the speedup. The block need to dynamically adjust to the new cars joining the block during the wait phase and changing conditions during the speedup phase. The group of cars that is supposed to described by the ensemble membership condition needs to be established in such a way that only coordinated speedup aware autonomous cars are part of it and no incompatible cars are placed in between.

Again the requirement on the coordination system is to be hard real-time and reliable. As failure to coordinate during the speedup phase can lead to crashes. On contrary, during the wait phase the cars can decide to break the group due to deficiency in communication, acceleration, or break capacity without putting anyone at risk.

4.4.6 Joining the Roundabout

Scenario VS6 The scenario is similar to the lane change Scenario VS3. There are cars passing through a roundabout, some of which are autonomous enough to regulate their speed on demand. Another fully-autonomous car is about to enter the roundabout and the cars on the roundabout take precedence. The car that is entering may struggle to enter as there may be just limited gaps in between the neighboring cars already present on the roundabout.

Encouraged by the generosity of some human drivers this scenario envisions a dynamic grouping algorithm that groups together a car starving to enter the roundabout and the cars already on the roundabout that prevent it from entering. Once the group is established the traffic situation can be optimized by making the cars on the roundabout move in such a way that one extra car can fit in there. The position of the, about to emerge, gap is then given as an advise to the car entering the roundabout so that it can schedule its arrival an minimize necessity to break.

The envisioned dynamic grouping algorithm composes of the two parts called *group creation* and *speed adjustment*. The *group creation* part maps to the ensemble formation and the *speed adjustment* part can be realized as a knowledge exchange in the just formed ensemble.

This scenario does not expect the advise system is real-time or reliable as filing to negotiate the entry or misunderstanding between cars just results in extra breaking and acceleration. Thus failure of the system can be tolerated.

4.4.7 Optimized Traffic Lights

Scenario VS7 This scenario envisions an ensemble grouping together a traffic light and cars that are approach to it. Considering the standard traffic light there are many limitations that arise from the fact that data exchange between cars and the traffic light is limited to several light signals and occasional perception of incoming cars by the traffic light controller. Apart from the game changer approaches such as Autonomous Intersection Management [33]. There are works such as an machine consciousness approach to urban traffic control [34] that aim to improve existing systems. Hence, the dynamic grouping of cars and the crossroad controller is discussed here as the opportunity to optimize the traffic.

Grouping together the traffic light controller and the approaching cars enable the system to adjust light intervals according to the cars quantities and also optimize a speed of the approaching cars so that sudden speed changes are reduced or even eliminated.

Considering an ensemble that aims only on advising the traffic lights and cars, the system can be seen as non-critical as failure can be tolerated. If the ensemble is used to coordinate cars and also to send go/stop signals to them, then the system has to react in hard real-time and possible failure may lead to crashes. Therefore, based on the degree of optimization this system can be seen as critical or non-critical.

4.4.8 Obstacle Discovery

Scenario VS8 Considering autonomous cars this scenario encompasses cars passing through a road that is at risk of containing obstacles such as broken cars, fallen trees, and accumulated snow or ice. Taking in the account low visibility or frequent turns the cars need to significantly slow down in order to maintain safety and guarantee enough break distance.

Dynamic grouping of nearby cars in ensembles based on their relative position and sharing a view of the surrounding in the ensemble can provide the missing confidence in sensors for most of the cars letting them to increase their speed and lower relative distance.

Considering the system as giving advises about the on-road conditions it is required to operate only in soft real-time as missing advise can be tolerated and just results in a fallback to the low speed. However giving false advises can result in significant troubles. The system needs be dependable. If the system further limits distances between cars to the level that fallback is no longer possible than the system has to be implemented as hard real-time. Such system is very close to collision avoidance system or platoon management covered in Scenario VS4.

4.4.9 Clearing Path for Emergency Vehicles

Scenario VS9 This scenario encompasses cars navigating through a street at presence of an emergency vehicle. Sometimes it is complex for cars to coordinate on letting the emergency vehicle pass as fast as possible. This is especially true when the space is limited such as in the city centers or the environment is harsh such as in the case of heavy snow fall. Smart exchange of data among the cars and the emergency vehicle may improve the situation.

Dynamic grouping of cars and emergency vehicles using ensembles envisions matching cars that need to react to the emergency vehicle approaching and coordinating those cars and the vehicle in order to let the vehicle pass as fast as possible.

Considering the requirement on the system it can be seen either as pure advisory system without necessity to provide non-functional guarantees or a strict coordination among cars that in case of failure or delay causes an accident.

4.4.10 Identification of Dangerous Conditions

Scenario VS10 This scenario encompasses random cars and envisions data sharing among them. Here, the ensembles define groups of cars focusing on different aspects of the traffic. Some cars can be grouped in an ensemble that provides coordination on safety distance maintenance. Different ensemble can be used to share data about road quality and obstacles. Yet another one can focus on identification of offenses against traffic rules and possibly let the autonomous cars keep safe distance from cars driven by non-conformant drivers.

The requirements on the system in sense of reliability and real-time processing depends solely on the property of the data the ensemble is sharing. In general it can be said that the ensembles are non-critical with exception of the safety distance maintenance one.

4.5 Global Vehicle Coordination

4.5.1 Route Optimization

Scenario VS11 The route optimization scenario encompasses cars passing through a road network composed of segments. The usage of ensembles is to form groups such that each one encompasses one segment, one time interval and all the cars that plan to pass through the segment during the time interval. Such grouping enable dynamic analysis of future road segment usage and optimization of car routes in such a way that segment capacity is not exceeded and, possibly, the combined travel time of all the cars is minimized.

One of the possible optimization algorithms that can be deployed on top of the dynamic grouping provided by ensembles is a naive algorithm that just reschedule cars that suffer from passing though busy segments to an alternative path that contain less busy segments. Assuming true dynamic assignment of cars to groups even such a simple algorithm might help to improve the situation. In order to reach an optimum a more advanced algorithm with bouncing protection, possibly based on the net flows, needs to be used.

4.5.2 Parking Place Registry

Scenario VS12 This scenario is similar to the Scenario VS1 but differs in the scope of the operation. While the Scenario VS1 focuses on a local area and peer to peer data exchange this scenario focuses on the global view of the parking places. This scenario composes of cars and parking places in a larger area. Dynamic groups are created that contain a car and its planed parking place including a time interval. Such system of groups defines a global parking place registry.

In the language of ensembles the assignment of a new parking place to the car is membership condition while the data exchange is the acknowledge of the parking place by the registry system. Such acknowledgement can be provided by a centralized registry or by the parking place itself. The parking place can either be equipped with a controller that is part of the system and hosts the parking place component that is part of the ensemble or the parking place components can reside on the server and be included as a Beyond Control Entity.

The membership condition check ensures that a particular parking place can be occupied only by a single car at a give time interval.

4.5.3 Long Distance Platoon Formation

Scenario VS13 This scenario extends the local Scenario VS4. The local version focuses on a platoon coordination in terms of the car control. This scenario focuses on formation of stable platoons while choosing cars that share the same destination. This scenario encompasses all cars in the region and the highway infrastructure in that region. Each car in the system has its destination independently set. If all the cars just start to form platoons together, the resulting platoon would be unstable as cars will be permanently leaving and entering the platoon due to the different destinations. This behavior introduces risks and lowers efficiency of the platoon. Instead the cars are suggested to form groups that share the same destination or at last significant portion of the track. Such groups needs to be negotiated on a possibly long distance and some of the the cars needs to be a bit re-routed in order to take part in the platoon.

The envisioned groups of the cars are formed based on the shared destination and route similarities. Such groups translate to platoons described by an ensemble membership condition that matches common segments of the planned route of the assumed member cars.

4.5.4 Car Sharing

Scenario VS14 The scenario is a car sharing system. There are people who use a car to get to their destination. The objective is to match cars and passengers in such a way that they can share a ride with someone and thus reduce total number of cars used. Possibly some of the original passengers does not need to own a car and rely only on a shared ride in order to get to their destination. There might be necessary to slightly alternate some of the routes as otherwise there would be just limited number of matches.

The ensembles are used in this scenario to describe a shared ride as a group composed of a car and its passengers. Membership of such an ensemble is based solely on its fitness. Fitness of a shared ride ensemble with given passengers and car is expressed as utility (reduction in car usage) divided by a price (extra distance and time traveled by the car used). The system has to be able to handle possibly many passengers and cars.

4.5.5 Car Sharing Management

Scenario VS15 Car sharing management scenario is very similar to the car sharing with global data described in Scenario VS14. In a sense it can be considered a taxi management service. It differs from shared ride groups in a strict separation of taxi cars and passengers. The taxi drivers has no other objectives than refueling the cars and providing rides to the passengers. This differs from shared cars where a car driver also has a destination.

The grouping in this scenario puts together an empty time slot of a taxi driver and a suitable passenger transport from A to B that needs to fit inside this time slot. The grouping does not take only time slots but also entry and leave locations of the passengers in order to include taxi in only such ensembles that are feasible. Possibly the system can be optimized by expressing an ensemble fitness in order to achieve shortest wait times for taxi and passengers and/or to minimize taxi car traveled distance.

4.5.6 Charge Station Assignment

Scenario VS16 Charging station allocation, as described in this scenario, is a similar to parking place registry Scenario VS12, yet more important. Assuming charging a car takes nontrivial time, it is necessary to either have many spare charging slots or make a schedule.

Regarding the schedule, this is the place where dynamic grouping using ensembles can help. Similar to car sharing mentioned earlier in the text the ensembles are constructed from cars in needs of charging and free time-slots on charging station slots. Assignment to ensembles may take in account a schedule of the car or just urgency of the charging request.

The operation of this system, even when it does not seem to be life critical needs to be performed in such a way that cars do not starve to charge. Compared to parking, where it is possible to send parking cars to more distant parking places or just let them roam around for a while, a car in need of charging may not make it to different charging station, nor it can wait in the traffic. Thus the system should minimize the occurrences of completely discharged cars to bare minimum.

4.5.7 Road Billing System, Toll Collection

Scenario VS17 This scenario encompasses payed road segments and cars that needs to pay for passing the segments. The envisioned ensembles have membership defined in such a way that a car that needs to pay for a road segment is grouped with such road segment. The ensemble knowledge exchange phase maps to the payment. This way the dynamic groups implemented by ensembles form a road toll collection system.

This scenario imposes some uncommon requirements on the system. First the payment needs to be implemented by a transaction, ensuring single payment per car per segment. Second taking part in the ensemble is not mutually beneficial for both the car and the toll collection system. The system needs to ensure that cars that are supposed to enter ensembles really do so.

Possible occasional failure of the system does not poses a threat to the security as it only leads to minor financial loss. Although major and lengthy failures should be avoided.

4.5.8 Street Lane Optimization

Scenario VS18 The scenario on long distance lane and traffic optimization encompasses cars and a road infrastructure configurable in terms of lane assignment and traffic sign content. The cars present their schedule in form of a list of road segments they plan to visit. The dynamic groups are formed between cars and their planed segments. Based on those groups described by ensembles, load and traffic type of particular road segment is determined. From the predicted load follow the parameters of the road segment such as lane assignment, speed limits, and other properties.

Assuming the lanes can be configured to a safe default, the system itself is not critical as a failure leads only to traffic complications. The reliability of the system should minimize failures of such scale that significantly affect the traffic fluency.

4.5.9 Emergency Situation Management

Scenario VS19 Global emergency situation management scenario is similar to the Scenario VS9. It differs in its global reach enabling not only path cleaning but also assigning emergency response vehicles to crashed cars and possibly endangered persons. Moreover it enables regular cars not only to clear path, but also to completely skip entering the crash site location due to possibility to re-route traffic elsewhere. Interaction of the system, with traffic control can result in assigning traffic light signs in such a way that emergency cars can proceed at maximum efficiency.

The implementation of the system relies on multiple ensembles that solve different aspects mentioned above in separation. It can be though of dynamic groups scheduling emergency vehicle arrivals to accident sites. next there can be dynamic group that encompasses cars that should rather avoid particular location in order to easy traffic for emergency response vehicles. Finally, there can be an ensemble grouping together traffic lights and emergency vehicles that ensures emergency vehicles have precedence at traffic lights.

CHAPTER 5

Modeling the use case scenarios with ensembles

This chapter captures functionality required by the use cases presented in Chapter 4 using the concept of an ensemble. Based on the analysis of ensemble usage and draft of the ensemble implementation, requirements on the ensemble functionality are summarized. The requirements range from knowledge specification, ensemble membership decision, knowledge exchange, communication requirements, and non-functional requirements such as reliability, resilience and real-time processing.

5.1 Ensembles of Autonomous Cars

Regarding the DEECo ensembles, the first thing to discuss is the component. When dealing with vehicular systems the natural components are cars. These form a logical units of functionality and provide an encapsulation. Such a component can be hosted on an on-board computer present in each car. The two main features of a component are processes implementing its business logic and component knowledge that defines its public interface.

Car as a Component

The first that needs to be decided when thinking of a car as a component is the knowledge. It represents the public interface of the component and holds the data the processes and ensembles work with. The DEECo relies on periodically scheduled and triggered processes. These seems to be natural choice also for the vehicular systems. The process of the car component is an implementation of the business logic of the car. With respect to the use cases the processes can be split into three categories. These are together with a knowledge discusses in separate sections.

Car Periodic Processes

Periodic processes are the most common processes used for multiple tasks. One of these is sensor reading. Sensors that are not capable of generating events or provide continuous data needs to be polled periodically. Sensors such as GPS or LIDAR are examples of sensors that either needs to be polled or generate data stream that needs to be periodically processed. Another usage of the periodic processes is the control logic of the car. Usually this is running in loops that pass through multiple stages of analysis, planning, and execution. These stages needs to be periodically repeated in order to keep the system running.

Processes in a Car Triggered by an External Event

External event triggered processes are an alternative to polling performed by periodic processes. If the sensor monitored by the process supports event generation (i.e. obstacles is detected, or a threshold value is reached) the process execution is linked to occurrence of such event. This limits resource wasting and reduces delay in between event occurrence and system response.

Processes in a Car Triggered by a Knowledge Change

Knowledge triggered processes are remote version of the event triggered processes. In this case the process is linked to change in the knowledge which can be caused by another local process, or an ensemble knowledge exchange execution. This way the process can react to remote event with reduced latency. An example of such event is change of availability of a parking place announced by a remote car.

Knowledge of a Car

Knowledge is a public interface and depends mostly on the use case, but some common parts can be identified to be shared by most vehicular use cases presented. The knowledge of a DEECo component is formed by data fields that describe current state of the component. The knowledge fields for the car component as required by the use cases follow:

- Id (General)
- Timestamp (General)
- GPS position (Scenarios VS1, VS12, VS4, VS7, VS8, VS11, VS13, VS15, VS17, VS14, and VS16)
- Road segment Id (Scenarios VS17, VS7, and VS18)
- Remaining battery capacity/fuel (Scenarios VS16 and VS11)
- State/Emergency (Scenarios VS19 and VS10)
- Sensed neighbors
  - Id (General)
  - Relative position (Scenarios VS2, VS4, VS3, VS6, and VS9)
  - Blocking cars (Scenarios VS2, VS3, VS9, and VS10)
  - Velocity (Scenarios VS3 and VS4)
  - Platoon Id (Scenario VS4)
  - Break / Acceleration capacity (Scenarios VS4, VS5, VS3, and VS6)
  - Intended move (Scenarios VS3, VS2, VS7, and VS9)
  - Obstacles (Scenarios VS8 and VS10)

- Communicated neighbors
    - Id (General)
    - GPS position (Scenarios VS2, VS3, VS6, VS9, and VS19)
    - Sensing capabilities (Scenario VS1)
    - Blocking cars (Scenario VS10)
    - Platoon Id (Scenario VS13)
    - Planned route (Scenarios VS11, VS13, VS14, VS15, and VS18)
    - Sensed parking (Scenarios VS12 and VS1)
    - Obstacles (Scenario VS10)

5.2 Use Case Feature Analysis

Aiming to analyze requirement on ensemble formation techniques this section presents a draft of ensemble membership specification. The draft of the membership condition uses various features that are later summarized. The membership expression is based on the knowledge fields. In order to improve readability the mapping knowledge fields drafted in preceding section is loose and some details might not be an exact match. The point of the implementation draft is to pinpoint features of the formation system and not to focus on syntactic details.

5.2.1 Implementation Draft

Parking Place Scanning

The membership condition, displayed in Listing 5.1 relates to the Scenario VS1. The condition is quite simple. There is a parking car that is supposed to be in state when it is about to park somewhere and a scanning car that is supposed to provide parking place surveillance. The role of both the components is *shared* as one car can provide surveillance for multiple parking cars and a parking car can have multiple scan providers.

```
membership(parkingCar: shared, scanningCar: shared) {
  return parkingCar.wantToPark == True && scanningCar.canScan == true
}
```

Listing 5.1: An ensemble grouping together a scanning and parking car.

A matching *fitness* condition is a bit more complicated. It is supposed to optimize geographical location of the scanning car with respect to the parking car position. As displayed in Listing 5.2 the parking car can provide a heat map showing the best places where to scan. The problem is that such a map may be not easy to spread using a limited network connectivity between the cars. Possible alternative is to rank scanning cars based on the distance to an optimal scanning position while omitting all cars that are more distant than some reasonable threshold. This way the map is replaced with a single position coordinate.

```
fitness(parkingCar, scanningCar) {
  return parkingCar.areaOfInterestHeatmap[scanningCar.position]
}
```

Listing 5.2: Fitness of an ensemble of a scanning and parking car.

Another approach to the parking place management is the parking place registry described in the Scenario VS12. Then the ensemble is supposed to book a parking place in the registry. The special requirement on the membership is to provide role exclusivity in time denoted by the *exclusive-in-time* keyword. This enables to match parking place and time slot to a car. The membership and fitness condition code is displayed in the Listing 5.3.

```
membership(car exclusive, place: exclusive-in-time) {
  return car.state == parking, place.state == free
}
fitness(car, place) {
  return 1 / distance(car.pos, place.pos)
}
```

Listing 5.3: Implementation draft of the fitness and membership condition of the Parking place assignment ensemble. This implementation uses parking place registry.

Narrow Passage Blockers Selection

Motivated by Scenario VS2 the membership condition displayed in Listing 5.4 matches two cars that mutually block each other. The required roles of the cars are exclusive as the resolution of the situation is supposed to resolve only two cars passing through a narrow passage in opposite directions. The exemplified simple ensemble can be extended to cover more complex situations such as a chain of cars blocking each other in a loop.

```
membership(carA: exclusive, carB: exclusive) {
  return carA.blockedBy.contains(carB.id) && carB.blockedBy.contains(carA.id)
}
```

Listing 5.4: Membership condition of an ensemble used to resolve two cars mutually blocked in a narrow passage.

Universal Blocker Selection

Another two similar membership condition definitions are presented here, motivated by Scenarios VS3 and VS6, solving the blocking cars. The idea is to form multiple ensembles for a particular situation where the car that is performing an action, i.e. changing lanes or joining a roundabout, is *exclusive*. Thus only one ensemble instance exists per such a car. The role of the blocking car is *shared* as a car can block multiple cars performing an action.

The difference between the use cases is expressed in the condition itself. The lane change condition displayed in Listing 5.5 matches cars that have conflicting lane interests and are driving side-by-side thus the blocking car is preventing the other car from the changing lane.

```
membership(changingCar exclusive, blockingCar: shared) {
  return neignbouringLanes(changingCar.lane, blocking.lane) && sidebyside(
    changinCar, blockingCar)
}
```

Listing 5.5: Membership condition for ensemble binding together lane changing car and a car that blocks the target lane.

While the roundabout join condition displayed in Listing 5.6 groups the cars that share the roundabout. These are further filtered by entry location of the entering car and projected location of the blocking car. Further, the roundabout join condition contains *fitness* function that evaluates which of possibly multiple blocking cars on the roundabout should let the entering car in. Based on the linear function of a required slowdown of the blocking car and a wait time of the entering car the most suitable blocking car is selected.

```
membership(joiningCar: exclusive, blockingCar: shared)  {
  return joiningCar.wantToEnterId = blockingCar.roundabout.id && blockingCar.
    roundaboutPos < joiningCar.roundaboutEntryPos
}
fitness(joiningCar, blockingCar) {
  return X * waitTime(joiningCar, blockingCar) + Y * slowdown(joiningCar,
    blockingCar)
}
```

Listing 5.6: Membership and fitness condition of an ensemble containing car that is about to join the roundabout and a car present at the roundabout.

Platooning

Regarding the local platoon, depicted in Scenario VS4, there are multiple ways how to design the ensembles that serve for the platoon coordination. Drafts of membership functions of the two options are presented here.

One way of defining the platoon coordination group is the linked approach. This way there are ensembles that contain pairs of cars. The first car in the pair is leading the second one. This way the whole platoon is composed of links as displayed in Figure 5.1. A draft of the ensemble membership function is presented in Listing 5.7. This approach to the platoon definition requires a special feature *exclusive-in-role*. The motivation for this feature is that the platoon links should form a single chain, thus the components must be used exclusively in the links, but exclusivity should be bound to a car in a particular role. It should be allowed for a single car component to take par in at most one link as a leader and at most one link as a follower.

```
membership(carA: exclusive-in-role, carB: exclusive-in-role) {
  return inFrontOf(carA, carB) && carA.canLead
}
fitness(carA, carB) {
  return distance(carA.position, carB.position)
}
```

Listing 5.7: Ensemble membership for linked platoon, coordination between pairs of cars.

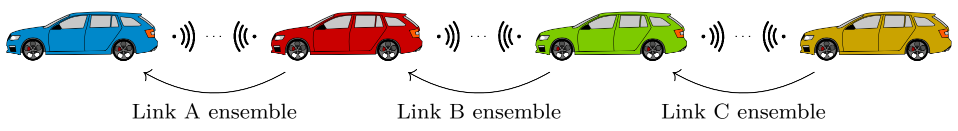


Figure 5.1: Communication links in the linked platoon. The four member platoon is linked by three leader-follower links A, B, and C.

Another way of defining the platoon ensemble membership is to have one group per platoon as displayed in Listing 5.8. Although it is easier to define, the downside might be a bit more complex code necessary for coordination.

```
membership(cars[]: exclusive) {
  cars = orderByPos(cars)
  return noGaps(cars -> map x: x.position)
}
fitness(cars[]) {
  cars = orderByPos(cars)
  return 1 / sum(cars -> map x: x.distToNext)
}
```

Listing 5.8: Membership condition of the overall-platoon ensemble.

The global view of the platoon is different. The platoon management, VS13, is supposed to organize cars into a platoons, thus the view of physical car positions is not so important. The key to group cars that should share a platoon together is the destination. The ensemble membership of such group is described in Listing 5.9. Such membership can also be interpreted as template that builds an ensemble group by addition of individual components.

```
membership(cars[]: shared) {
  return forall cars, car.destination == cars[0].destination
}
```

Listing 5.9: Platoon management membership condition.

Coordinated speedup at the traffic light, Scenario VS5, is similar to the platoon. The speedup group can be seen as temporally platoon formed just for the purpose of speedup. Such group is described by ensemble membership condition depicted in Listing 5.10. The ensemble can be in this case seen as a group that grows each time a new car satisfies a membership condition.

```
membership(cars[]: shared) {
  return cars -> reduce x,y -> directlyInFrontOf(x, y)
}
fitness(cars[]) {
  return min(cars -> filter x -> x.onTheBorder -> map x -> x.speedGainRestriction)
}
```

Listing 5.10: Membership function of the speedup coordination ensemble.

Traffic Optimization

The traffic optimization described in Scenarios VS7, VS11, and VS18 can be translated into membership conditions that group together cars and road infrastructure in order to identify overloads, re-route cars, and reconfigure infrastructure.

First the membership condition, inspired by Scenario VS7, that defines an ensemble of cars that can pass through the traffic lights together is discussed. The *fitness* of this ensemble depends on the throughput of the particular traffic light under the conditions defined by the cars that are supposed to pass through it as described in a draft implementation presented in Listing 5.11. Therefore each the ensemble instance formed contains the optimal group of cars to pass through the particular traffic light.

```
membership(lights: exclusive, cars[]: shared) {
  return cars -> map x -> atlightsId == lights.id
}
fitness(lights, cars[]) {
  return throughtput(lights, cars)
}
```

Listing 5.11: Draft of the traffic light interval optimization ensemble.

The route and lane optimization ensembles intended to cover Scenarios VS11 and VS18 are similar in the membership condition displayed in Listing 5.12 and 5.13. Both of them map cars to road segments. The difference is in the purpose of the ensemble. The route optimization ensemble is supposed to diverge cars from overloaded roads while the lane optimization ensemble is used to add capacity to heavily used segments by assigning an extra lanes.

```
memebrship(segment: exclusive, cars[]: shared) {
  return forall cars, car -> car.route.contains(segment.id)
}
```

Listing 5.12: Draft of the route optimization ensemble membership condition.

```
membership(car: exclusive, roadsegment: shared) {
  return within(car.position, roadsegment.area)
}
```

Listing 5.13: Lane optimization ensemble assigning cars to segments. Draft of the membership condition.

Emergency and Safety

Emergency and safety use cases presented in Scenarios VS8, VS19, and VS19 group cars together in order to prevent emergency situations or at last properly respond to them happening.

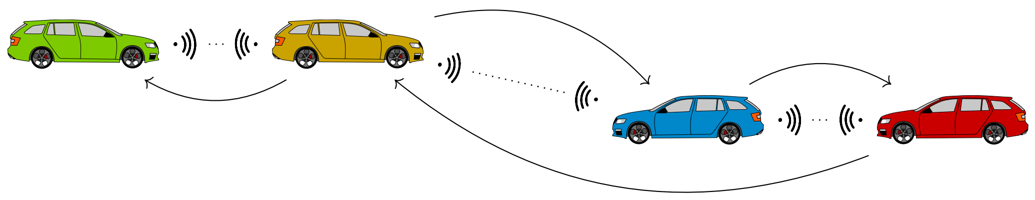


Figure 5.2: Communication in the obstacle detection and clear path guarantee ensemble. The green and the yellow car are meeting the blue and red car coming from the opposite direction. Dotted links symbolize wireless communication, solid arrows display sensor data flow in ensembles. The yellow car is sensing for the green one and the blue one. The blue one is providing data to the red car. The red car is sending sensor data to the yellow one.

The ensemble implementing Scenario VS8 encompasses cars that follow each other on a street as defined in membership function draft displayed in Listing 5.14. The purpose of the ensemble is to exchange information about possible obstacles on the road or to provide guarantee there are no obstacles ahead. The expected instantiation of the ensembles is displayed in Figure 5.2. Due to the real-time nature of the guarantee the ensemble only contains cars with sufficient mutual distance for reporting obstacles and optimizes towards optimal distance as displayed in Listing 5.14.

```
membership(leadingCar: shared, followingCar(s): shared) {
  return infront(leadingCar, followingCar) && distance(leadingCar, followingCar) /
     followingCar.speed < MAX_SAFE_REPORT_DELAY
}
fitness(leadingCar, followingCar) {
  1 / abs(distance(leadingCar, followingCar) / followingCar.speed -
    OPTIMUM_REPORT_DELAY)
}
```
Listing 5.14: Ensemble membership and fitness draft that group cars together in such a way that following cars are guaranteed to meet only known obstacles.

Emergency related ensemble, defined by the membership function displayed in Listing 5.15, is motivated by Scenario VS9. The purpose of the ensemble is to group together emergency vehicles and cars on their path in order to clear the path for the emergency vehicles.

```
membership(emergencyCar: exclusive, cars[]: shared) {
  return forall cars -> onPath(car, emergencyCar.destination)
}
```
Listing 5.15: Draft of ensemble membership function. The resulting ensemble is supposed to distribute information about approaching emergency vehicles in order to let other cars out of their way.

Another emergency ensemble encompasses a car in need of assistance, call center, and assigned emergency response vehicle. The membership condition and fitness function draft of the emergency assistance ensemble is displayed in Listing 5.16. The purpose of such ensemble is to group together car in need of assistance with matching assistance vehicle while optimizing arrival time.

```
membership(car: exclusive, callCenter: shared, ambulance: excluisve) {
  return car.emergency.state == True && car.emergency.assistance == Medical && !
    callCenter.busy
}
fitness(car, callCenter, ambulance) {
  return regionalFitness(car.position, callCenter.region) + 1 / distance(car.
    position, ambulance.position)
}
```
Listing 5.16: Draft of ensemble membership and fitness functions that group together crashed cars and emergency services.

Car Sharing

The use cases presented in Scenarios VS14 and VS15 are both related to the car sharing. The purpose of the ensembles in this case is to group together available car and a passenger.

The car sharing ensemble membership condition draft is presented in Listing 5.17. The passengers are grouped with car owner that share the origin and destination with tolerance of maximum allowed walk distance. The fitness function further optimize the total walk distance per ensemble.

```
membership(people[]: exclusive, carOwner: exclusive) {
  return  forall people, distance(person.position, carOwner.position) <
    WALK_DISTANCE && forall people, distance(person.destination.position, carOwner
    .destination.position) < WALK_DISTANCE
}
fitness(people[], carOwner) {
  return sum -> map people -> distance(person.position, carOwner.position) +
    distance(person.destination.position, carOwner.destination.position)
}
```
Listing 5.17: Draft of the car sharing ensemble membership and fitness function.

Taxi management as described in Scenario VS15 has a bit simplified ensemble definition as the taxi driver has no intended origin and destination. The drafted membership and fitness code displayed in Listing 5.18 groups together taxi cabs free at the time of journey of the passenger while the fitness function minimizes taxi car traveling overhead.

```
membership(taxi:: shared, passenger: exclusive) {
  return freeInTime(passeger.journey.time, taxi.schedule)
}
fitness(taxi, passenger) {
  return 1 / distance(taxi.expectedPositionInTime(passenger.journey.time),
    passenger.journey.start)
}
```

Listing 5.18: Taxi allocation ensemble membership and fitness function draft.

Charge Station Assignment

Another use case, where the distance optimization is required, is the charge station allocation Scenario VS16. In this case the ensemble contains a charging station and cars that can charge from it. The membership ensures the charging schedule is feasible, while the fitness function optimizes travel distance. The draft of the implementation is displayed in Listing 5.19.

```
membership(cars[]: shared, station: exclusive) {
  return  forall cars, suitable(car.estimatedArival, station.schedule)
}
fitness(cars[], station) {
  return 1 / total_travel_distance(cars, station)
}
```

Listing 5.19: Draft of ensemble membership and fitness function that implement charge station assignment.

Toll Collection

The toll collection ensemble, based on Scenario VS17, groups together a heavy weight car and a road segment it is passing through. A draft of the matching membership condition is displayed in Listing 5.20. The interesting feature of this ensemble is that it is not mutually beneficial for both the car and the road segment components. The mechanism of the ensemble formation should not allow the car to cheat.

```
membership(car: exclusive, roadSegment: shared) {
  return within(car.position, roadSegment.area) && car.type == Truck
}
```

Listing 5.20: Draft of the toll collection ensemble membership condition.

5.2.2 Membership Features

Based on the draft of the implementation presented in the Section 5.2.1 the required features are discussed here. The common base features encompass deciding *membership function* based on the knowledge. The differences are mostly in the selection of the ensembles that are eligible to join. Some minor changes to *membership* semantics are required for growing ensemble specification.

Shared Component

Specifying the components as *shared* in the membership function is the basic concept of DEECo ensemble that supports two components with a shared role. A *shared* component can be part of arbitrary number of ensemble instances of any type. The *shared* components are used across in many drafted examples. The *shared* membership seems to be practical even in combination with ensembles with complex structure..

Exclusive Component

A more restrictive than *shared* is the *exclusive* membership in ensemble. This is used main for ensembles that have complex structure due to the different usage of ensembles with more advanced structure. Ensembles with simple structure are usually used for data exchange where overlaps are not a problem while more structured ensembles, used for coordination, may malfunction if a particular component is member of multiple ensembles.

An *exclusive* component can be part of only one instance of particular ensemble. *Exclusive* membership is handy for defining unique roles in the ensemble, i.e. platoon leader, but comes at cost of high runtime complexity in ensemble formation algorithm. The *exclusive* is also used in many examples drafted in Section 5.2.1.

Exclusive Component in Role

In some cases the *exclusive* is too restricting. For example in Scenario VS4 where a platoon is described using the leader-follower links. The problem is that each component needs to be part of two links, but exactly in one of the leader and follower roles. In order to describe this a feature to mark component membership as *exclusive-in-role* is needed.

Exclusive-in-role behaves like *exclusive*. The difference is that a component is split into multiple virtual components, each having one of the roles of the original component. The virtual components are then assigned exclusively to ensemble instances.

Exclusive on Domain

In some of the Scenarios, such as Scenarios VS15 and VS16, where the feasibility of a schedule is one of the constraints, it would be nice if such constraint could be expressed at the level of the ensemble. One of the possible solutions is to define a domain of possible values of the schedule in the component and membership function while enforcing exclusive values in the member components.

Again this feature can be implemented by splitting each component into virtual ones. In this case the split is defined by different values that belong to the schedule domain. Ensemble instances than contain particular virtual component. i.e an ensemble instance containing a virtual component *taxi-car-from-8:23-to-8:56*.

Growing Ensemble

In case of the ensembles that contain components meeting a special condition that is supposed to hold once ensemble is established the membership condition can be formulated as a transaction that modifies existing ensemble by adding a new component into it. This feature is relevant to Scenarios VS4 and VS5. In this case the membership condition would have the form of a guard that checks whenever the new car is close enough to existing cars in the platoon or acceleration group.

5.2.3 Ensemble Purpose

The nature of the knowledge exchange depends on the purpose of the ensemble. The ensembles that arise from the discussed Scenarios are used for data exchange, coordination, and maintenance of a shared state. The purposes of the ensembles in different scenarios are captured in Table 5.1.

Use case	Data exchange	Coordination	Shared state
VS1	stream of frames	✗	✗
VS2	✗	order at crossing	✓
VS3	commands to car	lane change	✗
VS4	✗	speed and distances	speed
VS5	✗	speed, distances	acceleration
VS6	✗	precedence	✗
VS7	incoming cars	✗	✗
VS8	✓	✗	✗
VS9	emergency path	✓	✗
VS10	collect threats	✗	✗
VS11	collect road usage	optimize route	✗
VS12	✗	spot usage	spot allocation
VS13	✗	road, position	platoon data
VS14	available cars	stops and route	✗
VS15	schedule exchange	taxi schedule	✗
VS16	plan exchange	station slots	✗
VS17	collect payments	✗	✗
VS18	collect lane usage	lane assignment	✗
VS19	collect accidents	car assignment	✗

Table 5.1: Ensemble purpose of the vehicular scenarios. "✗" means the scenario does not use an ensemble such a way, "✓" mark or clarification text means that the scenario uses particular ensemble for the purpose.

5.3 Implications for Communication

The formal ensemble definition and formation is only a part of the problem. Functionality of the ensemble depends also on (i) the ability to propagate the knowledge of the component to the computational node that actually forms the ensemble and (ii) propagation of the resulting data back to the components. The key question in communication part of the ensemble formation is where the ensemble is formed and what communication media is used. Answers to both questions have serious impact on throughput latency, reliability, and overall ability to create a real-time system.

Regarding the vehicular, robotic, and Industry 4.0 scenarios the ensemble formation options encompass distributed operation, centralized formation on a server, edge cloud server based formation, and formation performed per ensemble on a mobile network node. The communication media used encompass short range packet radio, long range mobile network with the Internet connection, and an application specific networks build on top of a short range packet radios. The ensemble formation and networking options are detailed in the following sections.

5.3.1 Centralized Formation

Centralized formation of the ensembles depends on a server capable of establishing and maintaining the ensembles. The knowledge of all components, possibly hosted directly on mobile nodes, cars, needs to be send to the server. The server maintains ensemble instances and send possible knowledge changes arising from the ensemble operation back to the components.

On one hand this approach bring benefits such as consistency, efficiency in membership decision, and availability of a global view of the system. On the other hand the necessity to transfer the knowledge and maintain the permanent connection limits the scalability, resilience, and reliability of the system. The mentioned problems can be mitigated by placing the server to the edge cloud or other location close to the nodes hosting components. Then the communication latency and throughput disadvantages are limited. Further if the ensemble formation server is part of the mobile network it may have access to a raw connection to the network nodes hosting the components, hence providing guarantees on the network link properties.

5.3.2 Distributed Formation

Distributed formation keeps the ensemble membership decision and maintenance of the ensemble on the nodes hosting the components or other dedicated network nodes that are dispensed in the environment. In case of the simple, structure less, ensembles the formation of ensemble is fully distributed and conducted independently on all nodes hosting a component. A replica of the knowledge of a remote component and the local knowledge are tested for membership condition, once the test succeeds the knowledge exchange is performed in such a way that it only updates knowledge of the local component.

The bright side of distributed operation is the resilience to network failures, reduced latency, and scalability. The obvious downside is a loss of consistency and possibly limited options for ensemble membership and knowledge exchange specification.

5.3.3 Instance Based Centralization

This is a possible middle ground in the ensemble formation. Keeping the formation of a single ensemble instance on a single computation node, thus keeping the consistency, but allowing any node in the system to take function of an ensemble former. At the cost of increased complexity in the communication protocol and runtime design the ensemble formation can be deployed to mobile nodes hosting the components. Then each node tries to establish an instance of the ensemble and once it succeeds it obtains acknowledgment from all the member components. The acknowledged ensemble instance is than hosted on the node where it was created and the knowledge changes are propagated to nearby nodes hosting the member components. This way the ensemble operation would be consistent and resilient at the same time. Moreover the ensemble forming node might run on the central server and manage the components remotely, thus effectively providing smooth movement between the distributed and centralized operation.

5.3.4 Mobile Broadband

Mobile data connection such as 4G and future 5G provide long range connection and connectivity to the Internet. Sharing the knowledge using these technologies can be realized using an IP connection to a central server, or between mobile nodes hosting the components. Alternatively the network might allow clients to talk directly to each other while removing communication overhead and improving latency. Also the ensemble formation infrastructure might be part of the mobile network.

The advantage of the mobile broadband is the high throughput and IP connectivity enabling global range connections. On the other hand, the mobile broadband availability cannot be relied on in remote places. Moreover, the latency and reliability can be influenced by other clients using the same network. Also, in general the mobile broadband cannot be relied on to provide real-time communication media.

5.3.5 Packet Radio

MANETs such as IEEE 802.15.4 or WAVE provide short range communication. Using this media to share the knowledge requires broadcasting the knowledge of mobile nodes hosting the components to their direct neighbors. As the knowledge does not reach the whole network it may be necessary to use this media in combination with distributed ensemble formation and knowledge rebroadcasting mechanisms.

The benefit of the packet radio systems is the removal of dependency on the infrastructure resulting in resilient communication system. Moreover the WAVE operates in the dedicated radio band, thus the interference with other systems is reduced to minimum. Usage of the dedicated radio band and employment of a suitable Media Access Control (MAC) enables creation of a real-time EBCS.

The limited range of the packet radios can be extended by adding the infrastructure that provides a global IP connectivity or by employing an application specific routing protocol. Unfortunately both these options partially remove benefits of packet radio systems.

5.3.6 Communication Overview

In earlier sections the options for ensemble formation and knowledge sharing were discussed. An overview of the communication options in context of the vehicular

Use case	Formation			Communication		
	Centralized	Hybrid	Distributed	4G/5G	MANET	Real-time
VS1	✓	✗	✗	✓	✗	✗
VS2	✗	✓	✗	✗	✓	✗
VS3	✗	✓	✗	✗	✓	✗
VS4	✗	✓	✗	✗	✓	✓
VS5	✗	✓	✓	✗	✓	✓
VS6	✗	✓	✗	✗	✓	✓
VS7	✓	✗	✓	✓	✓	✗
VS8	✗	✗	✓	✓	✓	✓
VS9	✓	✓	✗	✓	✓	✗
VS10	✗	✓	✗	✓	✓	✗
VS11	✓	✗	✗	✓	✗	✗
VS12	✓	✓	✗	✓	✗	✗
VS13	✓	✗	✗	✓	✗	✗
VS14	✓	✓	✗	✓	✗	✗
VS15	✓	✗	✗	✓	✗	✗
VS16	✓	✓	✗	✓	✗	✗
VS17	✗	✗	✓	✗	✓	✗
VS18	✓	✗	✗	✓	✗	✗
VS19	✓	✗	✓	✓	✓	✗

Table 5.2: Ensemble formation and knowledge spreading features used by example scenarios.

scenarios is given in Table 5.2.

5.4 Network Reliability and Availability

In dependency on the scenario, impact of the network failures and outages, limiting the knowledge propagation, on the ensemble operation and whole system may be in some cases significant. A network shortage leads to removing of a components from an ensemble and possibly also to destruction of an ensemble instance.

Some of the scenarios, VS4, VS5, VS8, and VS9 do not allow for ensemble instance being broken in the middle of the operation. i.e. the platoon needs further coordination to tear down once it is established. Therefore a sudden loss of the connectivity may lead to a catastrophic failure. In case of the obstacle discovery and emergency path cleanup the loss of communication and subsequent loss of established ensembles does not directly lead to a dangerous condition, but seriously limit the operation of the system. As a consequence all cars may need to slow down and the cleared path for emergency vehicle is no longer guaranteed.

Another failure mode applies to Scenarios VS3, VS2, and VS6. These scenarios include negotiation that enables a car to make a move while the other cars make necessary space for it. The nature of the protocol is that no move is performed until the gap is established and detected by the sensors. Thus the unavailability or limited reliability of the network do not put the system into a dangerous condition. But the lack of network connectivity locally degrade the service and may limit the ability of the cars to perform the moves that were supposed to be negotiated.

The remaining scenarios do not pose a threat to system safety and granular degrade with decreasing network coverage. For instance the VS15 dealing with taxi management will suffer only minor disruptions if the connectivity is not provided in some areas.

5.5 Real-time Communication

Achieving soft real-time or even hard real-time processing is a challenging task even when a network is not used. The system needs to be analyzed down to the interrupt timing and the memory allocation is better to be completely avoided at runtime. These restrictions make even implementation of a network driver a challenging task. As the target vehicular systems are mobile it is necessary to think of a wireless network to be used for communication.

Once the wireless network is part of the system, it is necessary, apart from the usual problems, to deal with network latency, packet loss, and network congestion. The "air" is by nature a shared media where the network nodes compete for a media access. Moreover the same media may be used by multiple systems. I.e. the Wi-Fi provides almost no guarantees on MAC. Anyone can run a Wi-Fi enabled device sharing a frequency band with a system of interest. Fortunately the vehicular wireless systems such as WAVE use a dedicated frequencies, thus the others should not interfere. But still some random interference and faults in the MAC may cause occasional interference resulting in a packet loss.

In order to provide desired level of reliability the target system needs to run on a dedicated frequency band, employ MAC that prevents congestion or tolerate it, and rely on maximum of N subsequent packets being lost. The resulting probability of a failure together with a cost of the failure determine the system safety level.

The evaluation of the packet loss and failure rate gets complex once a packet is supposed to be delivered using multiple hops in the network. Usually the probability of the information being delivered on time drops dramatically once multiple hops are introduced. Fortunately the scenarios requiring the real-time communication do not use it for the long range communication. Naturally the tight coordination is required only on processes taking place close to each other in space.

As captured in the Table 5.2 the scenarios like VS6, VS8, VS5, and VS4 require some sort of real-time networking. The roundabout join scenario requires real-time communication to ensure that the gap where the car is supposed to join still exists, but the joining car can back off if too high latency is discovered or a message is missed. Similarly the obstacle discovery scenario can deal with variable latencies at the cost of reduce speed of cars. These scenarios belongs to soft real-time category where a missed deadline in delivery does not result in a system failure, but rather the system degrades its utility gradually while maintaining its safety. On contrary, the platoon and speedup coordination are hard real-time systems where the system fails terribly once the communication deadlines are missed.

Even if the imminent crash can be avoided by taking preliminary measures such as sorting cars form beginning to the end of the platoon by descending break capacity, missing a deadline in platoon coordination can be considered a hard system failure.

CHAPTER 6

Solution Strategy

This chapter outlines an approach taken in order to address challenges and fulfill goals of this thesis. Based on the motivation presented earlier in the text the use cases that motivate this work were described in Chapter 4. Keeping in mind the split of the use cases into groups based on the required communication distance and other estimated network related properties the use cases were analyzed in Chapter 5 while the interesting parts of their logic were captured in notion of ensembles. Using the drafted ensembles the required features on the ensemble specification and operation with respect to the original use cases were identified.

Based on the use cases, and in particular their analysis and mapping to ensembles, the following chapters describe the answers to the research challenges presented in Chapter 3. The results presented in Chapters 8, 9, and 12 were published in conference proceedings of CBSE 2015 [1], ISOLA 2016 [7], and SEAMS 2016 [6] respectively. The work mentioned in Chapter 10 was published in Journal of Microprocessors and Microsystems [5]. The simulation test-bed for experiments with sCPS, covered in Chapter 12, was also published as an artifact and made available to other scientists in the sCPS community and beyond.

First, different ensemble formation techniques are presented with their network related behavior explained as observed during experiments described in Chapter 13.

The classic DEECo bipartite ensembles, forming a baseline DEECo implementation, are described in separate section of Chapter 7. These are evaluated in terms of membership condition locality, partitioning, and knowledge exchange. In Chapter 7, also the intelligent ensembles, an extension that introduces roles and declarative ensemble specification, are described in terms of possible distributed operation and network related properties of their formation.

Next, the real-time operation of distributed sCPS is explained in Chapter 10. Exemplified on the Intelligent Crossroad System (ICS) the common issues of hard real-time system operating on top of the wireless network are explained including a possible way how to deal with the problems using the DEECo ensembles and a proper timing analysis.

Regarding the operation of EBCS on a realistic wireless network two distinctive methods for optimization of the communication were outlined in Chapters 8 and 9. First, a technique that allows to save messages and even improve quality of formed ensembles by partitioning a distributed system is described in Chapter 8. Then a general way of adaptation based optimization of the EBCS based sCPS is described in Chapter 9. Aiming on the autonomous optimization of the runtime towards current network quality and throughput the proposed approach identifies influential parameters and extend an architecture of the system in such a way that these can be set by the designer as well as automatically optimized at the run-time.

In order to properly evaluate the approach described earlier in the text an experimental implementation of the techniques was crated. Several frameworks were implemented allowing for deployment of components and ensembles. Features, usage examples, and benefits of these are described in Chapter 11.

Based on the one of the provided implementations JDEECo a test-bed was crated that enable easy experiments with adaptation of robotic sCPS. The test-bed described in Chapter 12 is composed of multiple simulation engines that take care of the precise simulation of robot movement, sensor readings, and network communication.

Measurements of various qualities of the draft implementations of the systems outlined in Chapters 10, 8, and 9 are presented in Chapter 13 in order to prove applicability of described approaches.

CHAPTER 7

Effects of Network on Ensemble Formation

This chapter describes bipartite ensemble and intelligent ensemble in Sections 7.1 and 7.2 respectively. The reason behind two different ensemble specifications is twofold. First, the distinction between the two is evolutionary. The concept of bipartite ensemble was initially created and the intelligent ensemble started as its extension that addresses lack of support for complex group description. Second, difference is in the operation and implementation. Different formation specification, used in intelligent ensemble, brings more expressiveness, but comes at the cost of additional restrictions on synchronization and requires a constraint solver to for the groups. Due to the operational differences, that sometimes prevent intelligent ensemble from being used, both the approaches remain relevant to the topic of this thesis. These are analyzed in this chapter in terms of requirements on the communication, resilience, and possibilities for centralized and distributed operation. The analysis is required in the following chapters to understand the underlying group formation techniques and their properties that are subject to various optimization and improvements in the following chapters. By providing the initial group formation analysis this chapter helps in addressing goals G1 and G2 of this thesis.

7.1 Bipartite Ensembles

This section deals with classical DEECo ensembles as outlined in [10] and implemented by a JDEECo framework in terms of network requirements and resilience to data loss. The classic DEECo ensembles follow the *coordinator-member* structure. Where the ensemble membership and knowledge exchange are described for the two components featuring the *coordinator* and *member* roles. The membership condition is periodically applied on all pairs of components available using a technique similar to Duck-Typing. A component containing all the fields used in the membership condition for particular role is tested for possible membership. This technique make the membership implicitly shared and enables existence of star shaped ensembles centered around the *coordinator* component as displayed in Figure 7.1.

The simple structure based on the *coordinator* and *member* roles enables fast membership decision and fully distributed ensemble formation. These ensembles are most suitable for data aggregation tasks. Achieving coordination is a bit more

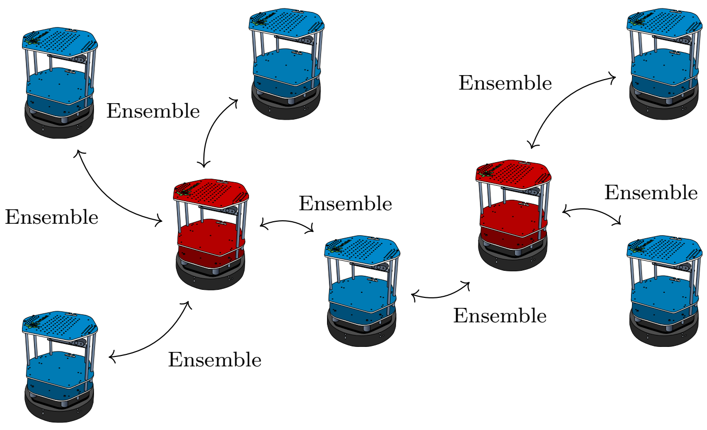


Figure 7.1: Start topology of ensembles allowing for data sharing. Coordinators (red) are in a bipartite ensemble with nearby members (blue). Data are exchanged from members to coordinators while aggregated results can be exchanged back to members.

complex task as a custom structure of the ensemble cannot be enforced easily. Therefore hence, the coordination usages are mostly limited to two component scenarios.

7.1.1 Membership Condition

The membership condition sets the requirement on the knowledge propagation in the network. In the JDEECo each component in the system publishes a replica of its knowledge and sends it to the network node responsible for decision of membership condition. In case of a centralized ensemble formation this is a single server responsible for maintaining ensembles. If the ensemble formation is distributed all the mobile nodes or cars in vehicular sCPS needs to receive fresh replica of each others knowledge. Considering large systems the technique seriously limits the availability of the system if used directly. Even when using more mature knowledge propagation, in case of the distributed ensemble formation, the consistency sometimes has to be sacrificed for the sake of the scalability.

In order to design a scalable and usable system it is necessary to find a sweet spot where the knowledge replica spreading is limited enough not to conquest the network while the utility of the system is still as high as possible. The techniques used to reduce network load are relying on two concepts. First is the boundary condition that limits where are the knowledge replicas spread based on an over approximation of the membership condition. The boundary condition is usable especially if the distributed ensemble formation is used. The second one is partitioning the components and looking for ensemble condition satisfaction only inside the partition. If centralized membership condition decision is used the partitioning removes necessity to concentrate knowledge replicas on a single node, thus allowing the system to scale. The details of mentioned techniques are described in the following sections and their possible effectiveness is evaluated in Chapter 13.

Prefer Close

If the membership condition restricts distance between *member* and *coordinator* to a reasonably low value. Such a restriction can be used as a boundary condition for the knowledge propagation in a distributed system. Even when the knowledge replica spreading needs to be further limited in order to cope with congestion the quality of the established ensembles will usually not suffer significantly.

Regarding the centralized ensemble formation the locality of ensembles can be used to partition the system. Multiple ensemble forming nodes, responsible for possibly overlapping areas, help the system to scale.

Prefer Random

In case of the membership condition being not related to distance between the *member* and *coordinator* it cannot be used directly as a boundary condition with the distributed ensemble formation. Further, it cannot be directly used to partition the centralized system. Depending on the other properties of the membership condition used for distributed ensemble formation, it may be possible to deploy an artificial distance limit that reasonably limits the network usage but still enables construction of valid ensemble instances. Assuming there is no relation between membership and distance a quality of ensembles should gradually degrade as the restrictions on maximum knowledge propagation distance are tightened.

In case of the centralized ensemble formation the distance can again be used to partition the system. Using physical location to partition the system should be as good as using any other membership unrelated value for partitioning.

Prefer Distant

Finally, in case of a high distance between components in ensemble being preferred, there is no easy way to enable the system to scale.

Considering a system with the distributed ensemble formation it is usually not easy to spread the knowledge only to nodes hosting distant components. Even if that would be technically possible the restriction on distance being more that a certain threshold is much less restricting than a condition on a maximum distance. Therefore the network congestion is not so easy to avert this way.

If the system uses a centralized ensemble formation and also performs partitioning by a component location the ensemble creation would be in most cases disabled as the components in the same partition would not be distant enough.

Partitioning

The component location is not the only option for partitioning the EBCS. In general the system can be partitioned by any value present in the knowledge. There are two reasons to partition the system that are discussed in following paragraphs.

Partitioning for Scalability As mentioned in the previous sections the DEECo system with centralized ensemble formation do not scale well as all the knowledge data needs to be passed to a single network node hosting the ensemble formation process. Apart from mentioned partitioning by a component location, it is possible to use any other knowledge field. i.e. the *id* of the component. Considering it is a numerical value the partitions can be $id \bmod N$. Splitting the system into partitions allow it to scale at the cost of limited ensemble creation opportunities as components in different partitions cannot take part in a single ensemble. The situation is different if selected partitioning is actually an over-approximation of the ensemble membership condition. In such case the ensemble creation is not significantly affected. Actually, if the partitioning is a strict over-approximation the ensembles are constructed as if the partitioning was not used.

Partitioning for Ensemble Formation In case where such a partitioning exists that it directly reflects a membership condition it can be utilized to implement the membership condition decision process. i.e. when the car components are about to be grouped in an ensemble in such a way that ensemble members have the same *destination* value in the knowledge, the partitioning by *destination* actually form such groups that can be transformed to ensembles directly. In such case a network node responsible for forming ensembles in a particular partition actually hosts a single instance of mentioned ensemble. Such a group can be further split into fine grained ensembles by additional constraints present in the membership condition.

7.1.2 Knowledge Exchange

A knowledge exchange is a periodical process that follows successful formation of an ensemble using a membership condition. The knowledge exchange is defined by a function that read and write knowledge of both the *coordinator* and *member* components. Put simply, the knowledge exchange can be seen as a process that runs on a merger of the two components. While the ensemble formation phase is sensitive to network throughput and knowledge reach, the knowledge exchange is usually limited by a network latency and stability. This is due to the fact that ensemble members are already identified and widespread knowledge replica propagation is not required. Instead, the data are exchanged only among the formed ensemble members resulting in much less overall traffic. Thus a possible coordination or state synchronization provided by the knowledge exchange is more sensitive to the latency than throughput.

The JDEECo implements bipartite ensembles in such a way that the knowledge exchange is performed just once right after the membership check. This behavior makes the knowledge distribution simpler as the same knowledge replica is used for the membership check and the knowledge exchange.

Centralized Knowledge Exchange

In a centralized system the ensemble membership decision and knowledge exchange are performed on a dedicated central server. This approach makes consistency between membership condition check and knowledge exchange easy to achieve. Also the membership condition check is easier to implement as the knowledge replicas can be archived and deeply analyzed on a central server that is supposed to have enough capacity to perform such task.

But still, the consistency of the knowledge replicas used is not guaranteed. Moreover, consistency issues arise when a result of knowledge exchange is about to be written back to the knowledge of the member component. Due to the delay caused by double network latency and knowledge exchange processing delay the knowledge of the component might have changed in a mean time. The change might have been performed by other ensembles or a local process of the component.

Distributed Knowledge Exchange

In case of the ensemble formation being performed in a distributed manner the consistency enforcement challenges are different. Relating to the vehicular sCPS, the deployment of the components on the cars enables also knowledge exchange to take place on cars. As well as the ensemble membership condition check also the knowledge exchange is performed between the local component hosted on a car and a remote component represented by a replica of its knowledge. The JDEECo implements the knowledge exchange by running the knowledge exchange function and writing the changed knowledge data to the local component only. A symmetrical knowledge exchange is supposed to take place on a remote component.

Distributed knowledge exchange simplifies integration of knowledge changes introduced by the ensemble operation. The exchange is performed locally on a network node that is also hosting the knowledge that is updated with the changes. Thus the knowledge updates can be synchronized among multiple ensembles and component processes.

On contrary, the consistency of knowledge exchange performed on different network nodes cannot be completely enforced. Consistency relies on ability of the system to perform the same actions on network nodes hosting the *member* and *coordinator* components. Latency and packet loss poses a threat to such ability and thus has a great impact on the overall consistency of the EBCS operation.

7.1.3 Communication Demands

Aforementioned operation of the ensemble formation and knowledge exchange can be translated to the requirements on the network in terms of reliability, reach, throughput, and latency. The requirements differ depending on the properties of the membership condition and a choice of either a centralized or distributed ensemble formation.

In an EBCS with a centralized ensemble formation where the underling network is IP based, the most of the requirements are usually satisfied. The IP network provides global reach. Reliability and high throughput are guaranteed on the side of the centralized server. On contrary, the latency is usually high due to complex routing and possibly sub-optimal location of the centralized server. Moreover, mobile nodes hosting the components are connected via a wireless network, thus subject to interference, limited bandwidth, and link quality issues. Also the mobile broadband solution used at the mobile node side may suffer from delays caused by link disconnects due to longer times needed to configure the connection.

In general the deficiency in mobile broadband coverage cannot be addressed at the level of system architecture. What can be influenced is the latency and throughput at the mobile node side. Emerging 5G as well as, in a limited manner, current 4G networks enable deployment of services, such as ensemble formation, directly to the mobile broadband network. This technology called edge cloud reduces latency in a server to client communication by placing the server closer to the client both physically and logically.

In case of the distributed ensemble formation the demands are almost exactly the opposite. Assuming the nodes hosting the components are communicating using a dedicated short range radio such as WAVE the latency and reliability demands are satisfied at the sort range. The problems arise at longer range where both packet loss and necessity to perform a multi-hop packet delivery make the situation a bit more complex. Especially in cases where a real-time cooperation of components is necessary, the possibility of losing a packet introduces a challenge to design the timing in a system in such a way that deadlines are met with sufficiently high probability. Naturally, as the chance of a packet loss gets higher the complexity of the timing design moves from hard to impossible.

The consistency of the distributed bipartite ensemble formation relies on the assumption that the communication works the same in both directions. In detail, this means that any pair of mobile nodes hosting two components A and B is guaranteed to either provide bi-directional communication between A and B or no communication at all. In case where the packets holding the knowledge replica are lost only in one direction the ensemble in question may be only partially established. This leads to knowledge exchange being performed only on A. Thus the possible coordination between A and B is seriously damaged as A behaves as cooperating with B, but B is not aware of that.

In order to support the distributed ensemble formation the network used by a nodes hosting both the components and ensemble former has to maximize available throughput a limit the packet loss. Experiments in the Chapter 13 prove these two properties are connected as with a high traffic and possible congestion the packet loss rate rises. Interesting guarantees are provided by the WAVE protocol designed for the Vehicle to Vehicle communication (V2V) communication. The dedicated radio channel and Time Division Multiple Access (TDMA) multiplex limit the packet loss due to interference and transmit collisions while the separation of communication into channels with QoS prevents losses of critical data due to congestion. Naturally, the packet loss cannot be avoided completely, but keeping it at manageable level is required for the system to generate desired utility.

Lack of the guarantees on latency and reliability can easily harm the consistency. In case of a delayed delivery of a changed knowledge or lost knowledge replica it may be impossible to apply effects of local processes and multiple ensembles the component is part of consistently. The effect of delays and losses in packet delivery can be mitigated by sending only changes that can be applied on a knowledge of the component in any order resulting in a consistent knowledge state. In order to use this feature the network itself or the network stack on the mobile nodes needs to provide reliable clock source and equip the packets with timestamps.

7.2 Intelligent ensembles

Intelligent ensembles [28][27] are based on bipartite ensembles while focusing on detailed description of complex groups of components featuring explicit roles. This chapter deals with requirements on the network communication of intelligent ensemble.

A knowledge in intelligent ensemble differs from the one in bipartite ensembles by explicit definition of a set of interfaces it implements. The interface is called role and defines a set of knowledge fields that has to be present in the knowledge of the component. The roles are used to restrict components that can take part in the particular ensemble. An ensemble specifies member roles and their cardinality. Only a component with a knowledge implementing the prescribed role can join the ensemble.

Another difference is that the membership of a component in a particular ensemble is exclusive. Put another way, a component that implements particular role can take part only in a single instance of the ensemble. This feature enables complex coordination and data aggregation to take place using the intelligent ensembles as it is possible to force ensemble instances not to overlap. Without this guarantee a close coordination, inside a group defined by an ensemble, is hard to achieve as a single component can be part of multiple instances possibly forcing it do perform contradicting actions.

As the system needs to chose which of possibly many ensemble instances should a particular component be part of a *fitness* function is introduced to extend the *membership* condition. The *fitness* function evaluate utility added by a particular ensemble instance to the system. The component is added to an ensemble in such a way that a system utility is maximized.

The exclusive membership together with fitness optimization come at the price of increased complexity of the ensemble formation. In fact, once the optimization of utility value is required an ensemble membership decision becomes an intractable problem.

Due to this the *membership condition* and *utility function* are expressed as logical expression using an EDL. This enables utilization of SMT solvers that can find out the optimal component to ensemble assignment. Providing that the number of components to evaluate is reasonably small the ensembles can be formed this way in a manageable time.

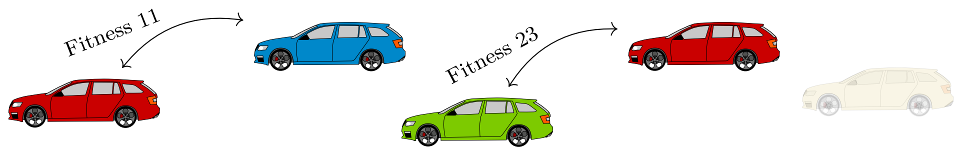
7.2.1 Solving Intelligent Ensembles

As the ensemble membership decision is not trivial, there are several approaches to the solution. In fact, usage of a particular ensemble membership decision technique has implications in form of different requirements on the network infrastructure. In detail, different decision techniques are differently sensitive to limited reach, reliability, and throughput of the underlying network.

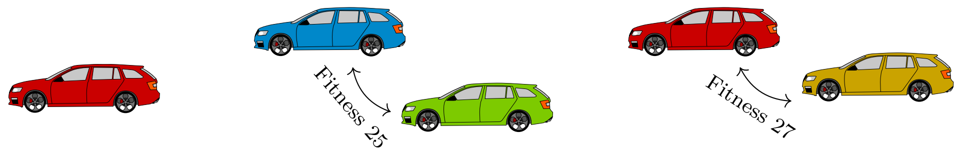
SMT Solvers

The standard approach to solve an intelligent ensemble, as proposed by intelligent ensemble authors in [27], is to use an SMT solvers on top of a membership condition and utility function expressed as logical expressions. This way the a particular area, or the whole system, is analyzed and the ensembles are instantiated in an optimal way as decided by the solver.

If a centralized formation of an ensembles is considered, this technique does not impose additional requirements on the network. In case of a distributed formation the situation is different. Due to direct optimization of ensemble utility on a



(a) Yellow car is not reachable due to network issues.



(b) Yellow car is reachable, ensembles reordered.

Figure 7.2: Instability of the optimal ensemble formation. Ensemble defines disjunctive car pairs, fitness is awarded for mutual closeness of cars in the pair. Inclusion of yellow car causes all ensemble instances to change.

large scale a minor change in the knowledge may result in a radically different instantiation of ensembles. Due to this fact usage of an outdated knowledge replica may significantly damage the system consistency. The same applies to the case where even a single knowledge replica is lost. As displayed in Figure 7.2 the optimal assignement of components to ensemble instances can differ dramatically when yellow car is hidden, Figure 7.2a, or visible, Figure 7.2b. Thus intelligent ensembles put strong additional requirements on symmetry in knowledge replica delivery a reduced or at last predictable latency.

Auctions

In response to the downsides of the SMT based approach to formation of the intelligent ensembles described in the previous section an auction based decentralized ensemble formation system is drafted here. The idea is to weaken requirement on the strict global optimization of instantiated ensembles by seeking only locally optimal assignment of components to the ensemble instances. This change enables limited knowledge propagation and simplifies ensemble formation process.

Naturally this change in the formation no longer support creation of ensembles that are formed from components that do not belong to a single local scope. As long as the local scope is defined by the rage of the knowledge propagation in a distributed system this limitation do not poses additional limits. Moreover, the the scope can be extended by long range communication links if necessary.

The operation of the auction based distributed ensemble formation system is maintained by a set of dedicated coordination components. The are deployed on mobile network nodes. These are hosting also the DEECo components. Alternatively the coordination components can be deployed on dedicated coordination servers or both the approaches can be combined. A coordination component receives knowledge replicas from DEECo components nearby and attempts to form an ensemble instance including the source components. In order to to make a component part of the ensemble instance the coordination component needs an acknowledgement from the component followed by a periodic keep-alive messages. This way the membership exclusivity and ensemble consistency is enforced.

This is where the auctions are used. A DEECo component that is wanted by multiple coordination components receives bids from those. A bid include increase in the utility caused by joining this particular ensemble instance and link quality to the coordination component. Based on the required stability and received bids the component acknowledges the highest biding coordination component.

The bidding system enables distributed ensemble formation while assigning components to the best ensemble it can reach. This ensures that vital ensembles with good connectivity are created that support further in-ensemble communication in order to perform the knowledge exchange.

7.2.2 Communication in Intelligent Ensembles

As well as in the case of the bipartite ensembles the communication requirements in intelligent ensembles differ based on the ensemble formation and knowledge exchange technique used. Different cases are detailed in the following sections.

Centralized Ensemble Management

The centralized ensemble formation has very similar requirements as in the case of the bipartite ensembles. As all the knowledge replicas are send to central servers for processing. This brings challenge to keep latencies low and maintain necessary global IP connection throughput in remote areas.

Distributed Ensemble Formation

An auction based ensemble formation, mentioned in Section 7.2.1, is considered for distributed ensemble formation. Assuming deployment of the components on cars and short range radio connection using a WAVE, Wi-Fi, or IEEE 802.15.4 the operation of auction based system brings several challenges with respect to the communication.

The most important one is the dissemination of the knowledge replicas. These needs to reach coordination components that will possibly include source DEECo component into an instance of the ensemble. If the dissemination using mentioned short range radio is used, it is critical to balance network utilization and dissemination reach. Based on the experiments in Chapter 13 it seems that a too wide condition on replica dissemination can easily lead to network congestion and significantly limited system utility.

Sometimes the local dissemination of the replicas using a short range radio is not enough. If possible, the technique used in bipartite ensembles to partition the DEECo components can be used to send a replica of the knowledge also to one of the remote servers that can provide additional coordination components.

Distributed Knowledge Exchange

As the intelligent ensemble has much richer structure than the bipartite ensemble the knowledge exchange needs to have network aware semantics. One of the possible approaches to the solution is introduction of ensemble knowledge. This move puts an ensemble on the level of component, thus makes it a building block of ensemble of ensembles. Apart from this architectural benefit the ensemble knowledge simplifies the knowledge exchange in a distributed system. Instead of a single knowledge exchange function that needs to be executed consistently in a distributed system a two functions are used. The first one executed on the coordination component, composes an ensemble knowledge based on the received knowledge replicas. Then the ensemble knowledge replicas are disseminated the same way as the ordinary knowledge replicas. The second function runs on, possibly many, components that are part of the ensemble and provide mapping from the ensemble knowledge back to the knowledge of the member component.

7.2.3 Communication Requirements of Component Coordination

Intelligent ensembles differ from the bipartite ensembles in the desired usage. Bipartite ensembles are mostly used for data sharing and aggregation. Explicit structure of intelligent ensemble makes them suitable for close coordination of the member components. Assuming usage in the coordination of components hosted on autonomous mobile nodes such as cars or robots a different network properties are required. The cooperation of mobile components require precise timing and state awareness.

Due to this a low latency network is usually required. A WAVE radio that uses a dedicated radio channel is one of the technologies available for this purpose in the automotive domain. Rebroadcast of the knowledge replicas, necessary to extend dissemination range, usually increases delivery latency. Fortunately, the longer is the distance at which the cooperation takes place the longer are the tolerable delays in replica delivery.

Another important property is awareness of broken or insufficient connection among the cooperating components. Due to this another requirement is to maintain active connection or send keep-alive messages in order to detect malfunction of the knowledge exchange. This is necessary to implement a safe tear-down of an ongoing cooperation among the components in order to avoid failures in coordination due to network deficiencies.

CHAPTER 8

Communication Groups

This chapter addresses limiting data propagation when dealing with coordination in a vehicular sCPS. The approach depicted on platoon scenario addresses knowledge propagation goal G1 of this thesis and partially also coordination goal G2 of this thesis.

Proposed solution discussed in this chapter takes into account both the local level coordination of a platoon performed using a short range MANET and the long range assignment of cars to different platoons. In particular, this chapter introduces concept of *groupers* that remove necessity to propagate knowledge data everywhere in the system. Proposed solutions that emerged during design and implementation of the demonstrator were tested in the simulation and evaluated in terms of network usage efficiency as described in Chapter 13. The results do not only characterize the chosen scenario but also describe features of coordination in vehicular system as a whole.

8.1 Platooning Scenario

As a running example, consider a scenario of emergency vehicles forming platoons. The purpose of a platoon is to optimize movement (in terms of speed, safety, and traffic disruption) of emergency vehicles towards the site of an accident. It is assumed that each vehicle is equipped with the necessary hardware enabling both short-range wireless communication (via MANETs) as well as infrastructure-based connectivity (long range, dedicated to emergency services). Vehicles within a single platoon communicate in order to maintain proper internal organization of the platoon and to ensure satisfiability of the safety requirements such as minimal distance between vehicles, maximal speed etc. Furthermore, all vehicles (also across different platoons) exchange information necessary to form a platoon, including desired destination, current location etc. The scenario together with two types of data flows is illustrated in Figure 8.1.

In this scenario, the focus is applied on the organization of vehicles seen as autonomous components that need to communicate globally (to form a platoon) and locally (while in a platoon). Whereas the local coordination requires low latency in data exchange, which is achieved by short-range communication, global coordination accounts for optimality in terms of network utilization.

Considering the chosen scenario, components correspond to the main actors of the system (i.e. vehicles). The template of these components is specified in Listing 8.1 by the Vehicle specification. Its state is captured by knowledge, Lines 8-17,

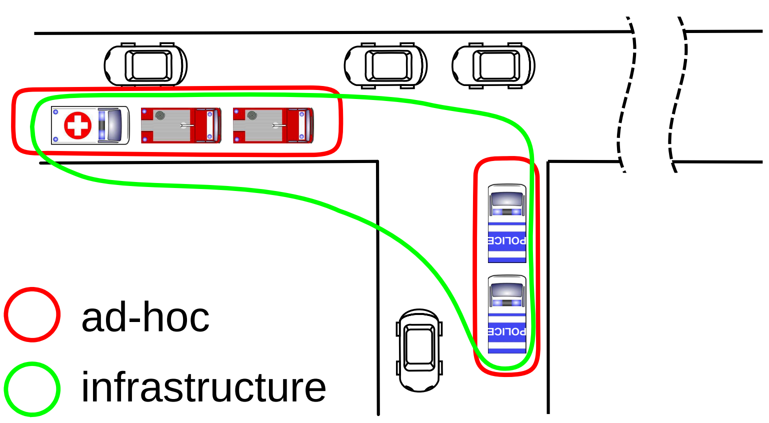


Figure 8.1: Platooning scenario – ad-hoc and infrastructure networks employment

and operational functionality by processes, Lines 19-27. Every component features a number of roles, Lines 1-5, which provide contract between the component and ensembles. Processes are executed by the DEECo runtime periodically or in a triggered manner. Each process execution starts with atomic reading (a part of) of component knowledge, executing the process body, and ends with atomic writing the results back to the knowledge.

```
1  role OtherVehiclesAgregator:
2    otherVehicles, position
3
4  role LocalTrainAgregator:
5    platoonId, localVehicles, speed, position
6
7  component Vehicle features OtherVehiclesAgregator, LocalTrainAgregator:
8    knowledge:
9      ID = V1
10     otherVehicles = [
11       (V2, {50.0722, 14.4568}),
12       (V4, {50.0745, 14.2356})  ]
13     localVehicles = [(V3, {50.25636, 14,4568}, 45.6)]
14     position = {50.075306, 14.426948}
15     speed = 54.2
16     destination = 109
17     platoonId = 5
18
19   process measureSpeed:
20       out speed
21
22     function:
23       speed ← SpeedSensor.read()
24
25     scheduling: periodic( 500ms )
26
27   ... /* other process definitions */
28
29   ensemble SameDestination:
30     coordinator: OtherVehiclesAgregator
31     member: OtherVehiclesAgregator
32
33     membership:
34       member.destination.Address = coordinator.destination.Address
35           AND !member.isPlatoonMember
36           AND !coordinator.isPlatoonMember
37
38     knowledge exchange:
39       coordinator.otherVehicles ← {
40           (m.ID, m.position) | m ∈ members }
41
42       for(m ∈ members)
43         m.otherVehicles ← {
```

```
          t ∈ coordinator.otherVehicles | t.ID ≠ m.ID }
      m.otherVehicles ← (
          coordinator.ID, coordinator.positon)

  scheduling: periodic( 700ms )

ensemble PlatoonManagement:
  coordinator: LocalPlatoonAgregator
  member: LocalPlatoonAgregator

  membership:
    member.platoonId = coordinator.platoonId

  knowledge exchange:
    coordinator.localVehicles ← {
        (m.ID, m.position, m.speed) | m ∈ members }

    for(m ∈ members)
      m.localVehicles ← {
           t ∈ coordinator.localVehicles | t.ID ≠ m.ID }
      m.localVehicles ← (
          coordinator.ID,
          coordinator.position,
          coordinator.speed)

  communication boundary:
    (sender: LocalPlatoonAgregator, node: NodeKnowledge)
         ⇒ ∃ component ∈ node.components:
      hasRole(component, LocalPlatoonAgregator)
          AND component.knowledge.platoonId = sender.platoonId

  scheduling: periodic( 200ms )
```
Listing 8.1: Examples of DEECo component and ensembles of the road trains scenario

In Listing 8.1, ensembles reflect the two types of communication groups of vehicles - within a platoon and across all the vehicles heading to the same destination. The ensembles in this scenario are bipartite ensembles with role defined component interfaces. The goal here is to propagate information about the vehicle's desired destination to other vehicles so that they can coordinate movement to form platoons. It should be emphasized that, knowledge exchange, realized by the ensembles to which a particular component belongs, is the only means of inter-component communication.

8.2 Communication Using Domain Knowledge

Tailored for development of sCPS, the DEECo component model allows designing a system at the architecture level without considering aspects related to its actual deployment - component distribution, communication infrastructure, and even its scaling in terms of the eventual number of component instances. Such an abstraction level simplifies modeling and development of the system, as it allows reasoning about components and ensembles in isolation, a crucial property when dealing with complex systems. Problems arise when it comes to deployment of the system, since there is a gap between the abstraction level of design and runtime infrastructure. This typically implies the need to apply standard generic methods for communication among distributed nodes. In particular in sCPS the efficiency of communication can be substantially improved by employing application domain data to optimize the deployment of the system.

This section describes how this can be achieved in DEECo by employing the concept of communication boundary [35], and, as a key contribution, the novel idea of communication groups.

8.2.1 Ad-Hoc Networks

DEECo and specifically its Java realization JDEECo [36], supports ad-hoc communication via MANET. It relies on periodic channel-level broadcasts (and rebroadcasts) of component knowledge. In a system, this allows a node not only be aware of the knowledge of the components it hosts but also to learn about knowledge of other (remote) components. This approach is appropriate for MANET that are not fully reliable and prone to frequent disconnections due to radio interference and mobility of nodes.

The communication protocol in JDEECo is based on bounded Gossip [35], where knowledge rebroadcasting is limited by communication boundaries articulated in ensemble specification. A communication boundary is employed by a node for deciding whether or not to rebroadcast the component knowledge heard from other nodes. This way, by constraining component knowledge dissemination to a specific geographical area, this mechanism allows better utilization of the communication channel, which in wireless settings comes at a great price. An example of communication boundary is given in Listing 8.1 (lines 69-73) when the component knowledge data dissemination is bounded to the nodes of the vehicles participating in the same platoon.

The specification of a communication boundary, is given as a predicate formulated on the component knowledge to be rebroadcasted and the knowledge of a rebroadcasting node. This way, the communication boundary reflects only the application domain-specific knowledge known at the architectural level. (Specifically, no information about future deployment is required.) In case of code displayed in Listing 8.1, the domain-specific knowledge captures the fact that a platoon is a spatially connected structure and thus it is sufficient to involve only the platoon nodes in the rebroadcasting. As an aside, this is the only enhancement to the original semantics of the ensemble as specified at the architectural level.

8.2.2 Infrastructure Networks

The benefits of the communication boundary is apparent for MANET and, in particular, VANET networks; nevertheless the idea is also applicable when dealing with more reliable infrastructure networks. In such settings, however, one can do more than just restrict data retransmission. Having a topology that does not change often (in particular if established links hold for a relatively long time), a routing mechanism can be introduced to provide for optimality with respect to, e.g., bandwidth utilization, latency, and computation balancing.

Therefore, JDEECo utilizes Gossip in case of infrastructure networks [37]. As a data dissemination protocol, Gossip is resilient to communication failures. Nevertheless, depending on the application, standard Gossip may still be costly, especially in terms of the amount of data being transmitted. Specifically, in JDEECo standard Gossip causes that component knowledge is published periodically to all nodes in a system, which does not scale well for large-scale systems.

Communication groups

To mitigate the problem of unnecessary data transmission over the network, an extension was introduced to ensemble definition by introducing communication group, delineated according to the component knowledge of the *coordinator* and *members* of an ensemble. The basic idea is to introduce the groups of components (*members*) that are related to each other in terms of a specific knowledge value (e.g. having the same value of the destination attribute). Such a group serves as a hint for optimizing deployment in terms of communication efficiency by restricting and localizing the area in which discovery of components to form an ensemble is performed. Defined again at the architecture level via component knowledge specified in roles, orthogonally to the membership, knowledge exchange, and communication boundary, the concept of communication group just enhances the original semantics of ensemble, not modifying the meaning of other DEECo abstractions. For illustration, consider line 5 in code Listing 8.2, indicating that communication groups will be based on the destination value in the *coordinator's* knowledge. In this case vehicles going to the same destination (expressed by the membership condition) compose communication groups, each of them corresponding to a specific value of the destination attribute in the *coordinator's* knowledge. The situation is visualized in Figure 8.2, where ensembles of different emergency vehicles platoons are heading to distinct locations in Prague 6 and in Prague 4 districts.

```
1  ensemble SameDestination:
2    coordinator: OtherVehiclesAgregator
3    member: OtherVehiclesAgregator
4
5    communication group: coordinator.destination.CityDistrict
6
7    membership:
8      member.destination.Address = coordinator.destination.Address
9        AND !member.isRoadTrainMember
10       AND !coordinator.isRoadTrainMember
11
12   knowledge exchange:
13     coordinator.otherVehicles ← {
14         (m.ID, m.position) | m ∈ members }
15
16     for(m ∈ members)
17       m.otherVehicles ← {
18           t ∈ coordinator.otherVehicles | t.ID =neq m.ID }
19       m.otherVehicles ← (coordinator.ID, coordinator.positon)
20
21   scheduling: periodic( 700ms )
```

Listing 8.2: An example of communication group specification

Groupers

Communication groups are utilized to optimize deployment, where they support the planning of inter-node communication links. For that reason an extension to the JDEECo runtime environment is proposed by introducing the concept of grouper. The basic idea is that a grouper limits the Gossip only to the nodes that host the components belonging to a particular communication group. Thus a grouper employs the communication group specification. Technically a grouper enhances the JDEECo runtime environment in the following way. The environment contains a set of JDEECo runtime instances (Figure 8.3). Each of them hosts a set

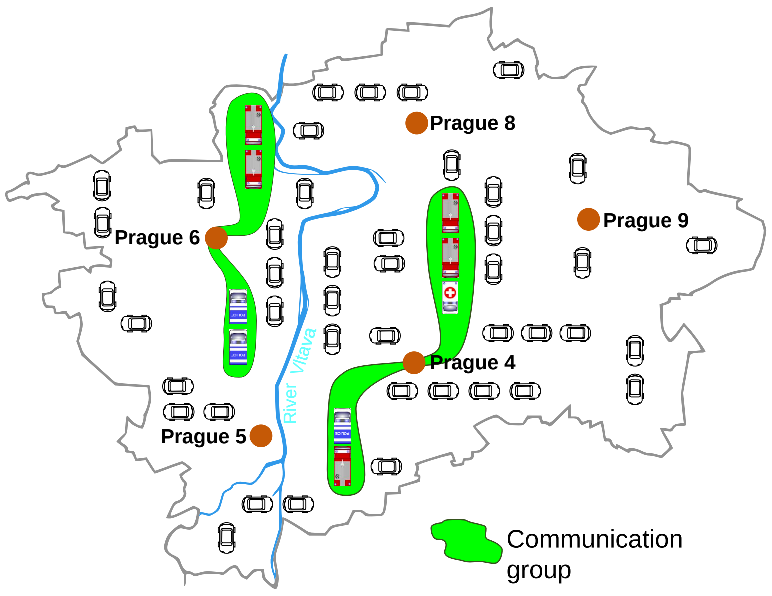


Figure 8.2: Illustration of communication groups. Each one associated with an instance of the *SameDestination* ensemble.

of components, the knowledge of which is gossiping around, using the addresses of other nodes stored in its peers table. In the enhancement, a grouper can also be referenced in the peers table as illustrated in Figure 8.3. It is assumed that a grouper (i) is a representative of a communication group(s), (ii) is equipped with all the ensemble definitions in the system, (iii) has access to knowledge of all components needed to decide the membership conditions, and (iv) can modify the peers tables that contain a reference to it. The basic functionality of a grouper is to continuously monitor the current ensemble memberships of all the components in the groups it represents and update the peers tables accordingly. By modifying the peers tables, a grouper implicitly routes the component knowledge to the most relevant nodes (i.e. hosting the same ensemble components) in the network. Figure 8.3 exemplifies the whole idea on the *SameDestination* ensemble, the communication group of which depends on the destination field in the *coordinator's* knowledge. In this case the grouper dedicated to Prague 6 continuously monitors the ensemble membership status of all the components it is aware of. If necessary, it updates the peers tables of the respective JDEECo runtime instances on the nodes hosting the components in the *SameDestination* ensemble. The specification of the communication group allows a node to determine the groupers for a given pair of an ensemble type and a component hosted on the node. This way the knowledge of the component is routed only to a limited set of groupers (as opposed to being propagated throughout the whole system). Also, as communication groups are typically geographically or network-wise localized, they lead to a decentralized solution, potentially characterized by low-latency. The decentralization also means that the operation is possible even in case that the infrastructure network gets partitioned into a number of disconnected subnets (i.e. without global internet connectivity).

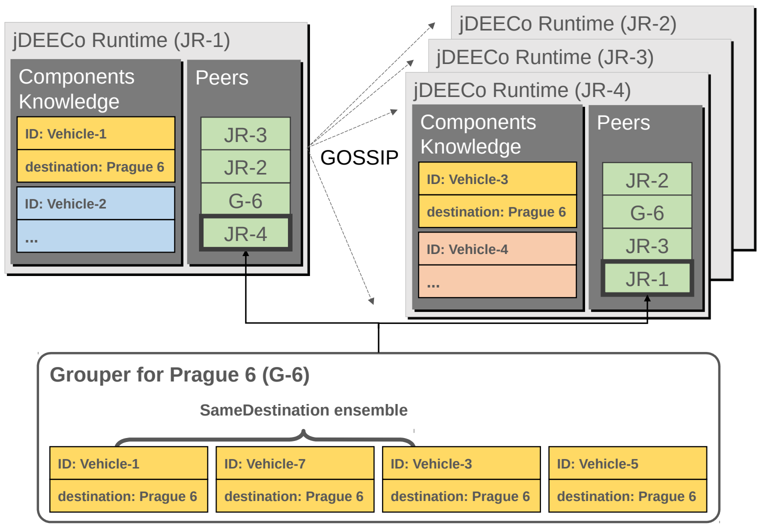


Figure 8.3: JDEECo runtime instances – groupers associations

CHAPTER 9

Adaptive communication

Communication is an important factor for coordination and formation of dynamic groups. Based on the communication technology used there are some parameters such as throughput, packet loss rate, maximum range that are determined by the technology and the environment. The only thing that remains in the hands of the designer is configuration of the system in such a way that it leverage available communication assets efficiently. Naturally, in case of wireless media usage the quality of the communication changes dynamically based on the current noise in the environment and congestion cause by communicating too much.

This chapter addresses goal G2 of this thesis with respect to the network limitations. The problem is addressed by introduction of a resilient component design that enable operation with limited network usage. The resulting system can operate with various settings that limit network usage, but the cost of limited networking is reflected in lower overall system utility. Moreover, the parameters can be used to adapt system towards the realistic network properties while optimizing its utility.

Considering a DEECo based system parameters that influence the communication demands and at the same time the utility of the system include ensemble period, knowledge propagation distance, maximum replica age, and a few others. Based on the fact that these variables captured by the system architecture are in control of the framework it is possible to let the framework decide dynamically on the optimal values assigned to these variables. The text in this chapter describes an extension to DEECo architecture that enables the developer to set these variables separately and even let the framework dynamically optimize its behaviour with respect to the changing network conditions.

9.1 Scenario

To illustrate this enhanced semantics of ensembles, consider a game in which a group of robots has to cooperate in order to touch as many beacons as possible, with the beacons dynamically appearing in the game area (Figure 9.1). More specifically, a beacon has to be touched by a pair of robots at the same time. Additionally, the area is divided into islands and robots cannot easily move from one island to another.

The robots in this scenario are modeled as components. A robot's knowledge contain its position and the position of a targeted beacon. There are two types of ensembles in the scenario: (i) an ensemble grouping all robots on a single island

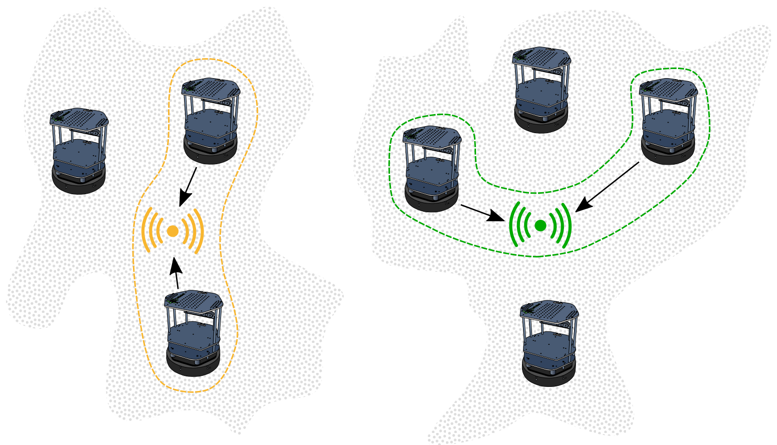


Figure 9.1: A robotic example visualization. Dotted areas represent islands. Orange and green radio symbols are the beacons. Ensembles pairing robots heading to a particular beacon are displayed using dashed curves.

(this one serves to exchange information about beacons discovered by the robots), and (ii) an ensemble grouping two robots that are selected to touch a particular beacon. Obviously, the former ensemble exists in multiple instances (one per island), the latter exists in multiple instances too (up to one per beacon – there may be less if there are not enough robots to pair with all the beacons present in the area).

An ensemble (i) expresses that all robots in the ensemble are on the same island. The ensemble coordinates robots within it and allows them to share their knowledge. For instance ensemble (i) provides each robot a union of beacons seen by each individual robot in the ensemble; ensemble (ii) tells a robot which beacon to touch.

9.2 Communication Concerns

Due to the inherently distributed nature of sCPS, communication becomes an important issue. The distributed character of sCPS, combined with the potentially limited ability to communicate, prevents a centralized management of ensemble formation and component communication – mainly due to the fact that components need to efficiently and safely operate even in cases when communication becomes unavailable.

In this robotic example, a centralized solution would simply not scale with an increasing number of robots, bringing an inherent bottle-neck and a single point of failure of the system. Thus, a decentralized solution is necessary. Not surprisingly, this is clearly in line with the fact that systems like car-to-car communications or swarms-of-robots typically rely on MANETs. In order to effectively distribute data in a decentralized and highly dynamic system, MANETs commonly use protocols that are based on proactive data distribution, i.e., Gossip [38], a rather effective style of communication for data distribution, and geographical routing [39].

The central idea in Gossip protocols is iterative information exchange between network nodes. In every step, a node communicates with a selected peers and exchanges a small amount of information with them. Typically, the peers to communicate with are chosen via a randomized mechanism. Commonly, the messages are replicated by peers and thus, there is an implicit redundancy (and therefore robustness of the protocol).

In [35] an application of gossip-based decentralized communication for Autonomic Component Ensembles (ACE) was demonstrated. Nevertheless, if applied without any further restrictions/optimizations, such communication can quickly lead to very long communication latencies, or, even, can totally congest the whole network. To deal with this issue, other MANETs properties like potential data loss, varying latency, bandwidth, etc., and resulting effects like stale data, necessity to rebroadcast data, etc. have to be taken into account (MANETs are primarily intended for unreliable communication). To tackle these properties, the communication boundary predicate was proposed. In short, it defines the furthermost limit where data have to be disseminated (i.e., data are not disseminated throughout the whole network), but the resulting system is functionally equivalent to a system where data would be disseminated everywhere. As shown in [35], this can lower the congestion on the network and improve other non-functional properties of the system.

The communication boundary as such provides static safe over approximation – unfortunately, this is often not enough for complex sCPS. If the robotic example above is considered and a further rule is added to the game. That for pairing the robots, two robots with largest distance between them should be preferred. The system is essentially forced to propagate data all over in order to learn about beacons that lie really far apart. The excessive communication and ensuing congestion and latencies then prevent the system from functioning as designed (the robots target beacon pairs with the largest distance apart but due to latencies, the robots are unable to coordinate in timely manner).

To remain functional in that case, the system has to adaptively modify/restrict the communication, which of course decreases its overall effectiveness, but on the other hand, it allows it to keep at least some functionality. This requires that the system has the ability to dynamically scale w.r.t. to temporarily and spatially varying properties of the communication. There are existing techniques to do this on the communication middleware layer. However, to perform such a scaling efficiently, the system has to possess a certain degree of domain knowledge. In particular, the system has to know (i) how its effectiveness and potential to reach the goals are influenced by communication (e.g., what the permissible latencies are, what of the communication parameters have positive effects on the system efficiency), and (ii) how the system can be restricted (e.g., by disregarding some mutually distant components from forming an ensemble) in order to achieve at least a sub-optimal functionality of the system. Unfortunately, there are no universal answers to the questions above since they are highly problem specific.

9.3 Towards Self-Optimizing Ensembles

As pointed out in the previous section, there is a clear need that designers/developers of sCPS can explicitly express the information influencing the communication in order to allow sCPS to self-optimize themselves at runtime based on the current conditions of the network. Following the architectural hoisting concept, such information has to be a part of the architectural specification. Since this is to be provided at design time, the specification of communication parameters has to be platform-independent, to reflect that decisions on a particular communication platform are typically made later than those on the architectural level.

Another strong requirement is that from the architectural description, there has to be a clear and exactly defined relationship between the views on the designed system, considering (i) an ideal system (i.e., a system with instant communication, without any latencies, and no congestions) and (ii) an actual system (i.e., a system with real latencies, limited bandwidth, etc.). The ideal system view is important since it is easy to understand it and verify (using the Model driven Architecture (MDA) [40] terminology, it represents a platform-independent model of the system). The actual view then reflects a reality, which should behave as the ideal system (a platform-specific model in the MDA terminology) as much as possible. From the functional point of the view, both the ideal and actual systems work correctly, however the actual system can produce sub-optimal results or behave with smaller efficiency, compared to the ideal system.

An approach presented in this section thus enhances ensembles with communication related information but in an underlying platform-independent way. These pieces of information are then interpreted by the deployment infrastructure keeping the goal to run the actual system to correspond to the ideal system as much as possible with regards to current conditions of network, etc.

9.4 Network Aware Ensembles

In this section, an approach is demonstrated how to equip ensembles with high-level problem-specific information which allows the system to scale its functionality w.r.t. the limitations of the communication. As a particular example for illustration of the described principles the robotic game example defined in Section 9.1 is used.

Figure 9.1 shows the example in (a simplified version of) the DSL of the DEECo component model (however, almost any component model could be used to model the example). As introduced in Section 9.1, there is one type of component – *Robot* – and two types of ensembles – the *BeaconInformationExchange* ensemble and *ForSingleBeacon* ensemble.

```
component Robot
  knowledge
    position // robots position
    beaconPosition // targeted beacon position
    islandID // island, on which the robot is located
    beaconPositions // positions of known beacons

  ensemble BeaconInformationExchange
    id islandID

    membership
      roles
        source: Robot
        target: Robot
      condition
        source != target

    knowledge exchange
      target.beaconPositions = target.beaconPositions.unionWith(source.
    beaconPositions)

    communication constraints
```

```
      boundary
        relay: RobotRelay, replica: Robot
        relay.islandID == replica.islandID

ensemble ForSingleBeacon
  id beaconPosition // targeted beacon position

  membership
    roles
      robotsAssignedForBeacon[2]: Robot
    condition
      robotsAssignedForBeacon[0].islandID == robotsAssignedForBeacon[1].islandID
    == islandIDOf(beaconPosition)

    fitness
      max(distance(robotsAssignedForBeacon[0].position, beaconPosition),
  distance(robotsAssignedForBeacon[1].position, beaconPosition))

  knowledge exchange
     robotsAssignedForBeacon[0].beaconPosition = beaconPosition
     robotsAssignedForBeacon[1].beaconPosition = beaconPosition

   communication constraints
    boundary
      relay: RobotRelay, replica: Robot
      relay.islandID == replica.islandID
    optimization
      smallestRadius > 10m
      max staleness beaconPostion 30s
```

Listing 9.1: Robotic example in the simplified DEECo DSL

The first ensemble serves for spreading information about beacon positions since the robots have just limited field of view. Simply, all robots on the same island merge their knowledge about beacons they have seen on the island. The ensemble is qualified by an island ID, which means that the system may create as many instances of the ensemble as there are islands. The second ensemble serves for both communication and coordination – it tells a pair of robots which beacon to target. The ensemble is qualified by a beacon (represented by its position), which means that the system may create as many instances of the ensemble as there are beacons.

9.4.1 Ideal System

As explained above, with real-life communication in place, the ideal view of the system and its actual view, which comes about as the result of compromises that have to be done due to restrictions imposed by the communication, is distinguished. The logic of the ideal system (when communication and ensemble formation would be instantaneous) is given by the definition of membership and knowledge exchange.

The membership defines (i) types of components taking part in the ensemble along with their roles and (ii) a filtering condition over their knowledge. For instance in case of the *ForSingleBeacon* ensemble, it groups two robots and requires that both these robots plus the targeted beacon have to be on the same island.

The membership may further contain the fitness definition, which is a function used to rank potential ensemble instances if the membership condition allows more than one solution. This indeed happens in our case as every pair of robots on the same island satisfies the condition. Further, considering that more beacons (i.e., ensemble instances) are present on the same island and that a robot can be assigned only to one ensemble *ForSingleBeacon*, there ensues a large number pair combinations. Out of these, the system selects such robots-beacon pairs that yield the maximum sum of *fitness* (which in our example is the maximum sum of distances).

The knowledge exchange specifies what knowledge fields to assign to which roles in the ensemble. In case of the *ForSingleBeacon* ensemble it assigns the beacon position to both the robots in the pair.

To reflect the fact that the system is dynamically changing (e.g., new beacons can appear in the system), the membership (with fitness) and the knowledge exchange are continuously reevaluated and ensembles are reformed to reflect changes in the membership condition and to optimize with respect to the maximum sum of *fitness*.

9.4.2 Real System

The actual system is essentially a restriction of the ideal system in which no global system state is available. Technically, this boils down to limiting propagation of data between physical nodes on which components are deployed. This takes two forms: (i) communication happens only among nodes which may be together involved in forming an ensemble – i.e., there is no communication across islands because there is no ensemble needing it, (ii) ensemble instances that are hard to create and synchronize may not be considered by the system if it would lead to network congestion – i.e., very distant robots pairs on the same island which would require expensive multi-hop communication are disregarded.

The data propagation limit (i) is realized by the communication boundary [35] (captured in the specification as boundary condition under communication constraints). It is a predicate which tells whether to re-transmit a data packet from a given node. An important feature of a communication boundary is that by design it is required to be implied by the membership. As such, the communication boundary is an over-approximation of the membership and cannot omit an ensemble instance which would normally be present in the ideal system.

The data propagation limit (ii) is realized by dynamic communication boundary, which is controlled by setting the communication parameters (performed by the system at runtime based on the actual network utilization). This is explained in Section 13.3 in more detail. An important feature of this is that it can omit ensemble instances that would normally be present in the ideal system. As such, it makes the actual system deviate from the ideal. This tradeoff however has to be made because otherwise the system would stop functioning completely due to network congestion. The dynamic communication boundary is continuously optimized by the system with two objectives – (a) keeping the network load below the congestion point, (b) maximizing the sum of ensemble fitness.

To prevent cases when the dynamic communication boundary would become too tight for the system to work, the ensemble specification constrains the optimization of the boundary (specified as optimization condition under communication constraints). These constraints are expressed in terms of time (maximum staleness) and space (smallest radius) rather than particular technology dependent communication parameters. This allows keeping the level of abstraction closer to

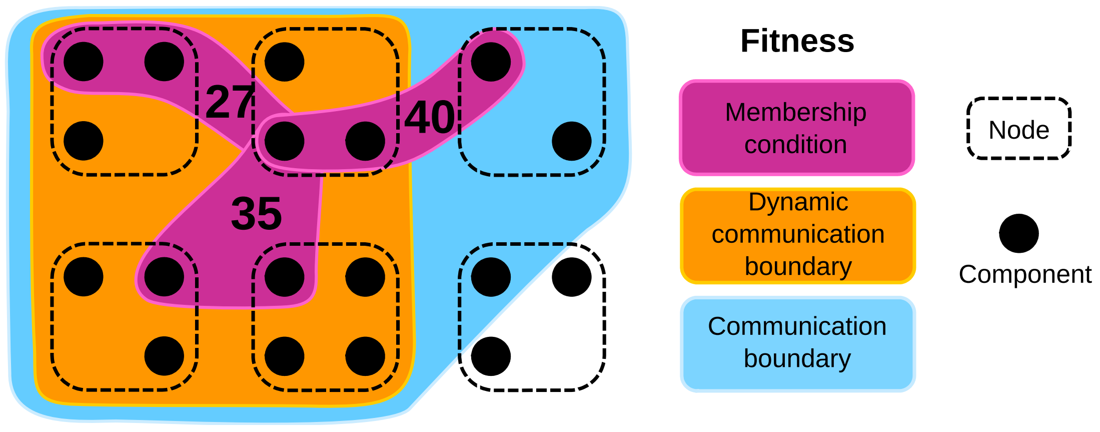


Figure 9.2: Membership vs. boundary conditions in ensemble formation

the problem domain and easier to reason about on the level of the system design.

In particular the maximum staleness parameter defines the maximum age of data that can be still relied on. In the example, as the robots have limited range to see the beacons, the targeted bacon can become “stale”, i.e., the particular robot does not see it yet and it was not “refreshed” via the *BeaconInformationExchange* ensemble. Thus, information about targeted beacon must not be too old (30s in the DSL) as with increasing age of the beacon information, the probability that the beacon has already disappeared also increases and, regarding the overall goal of the system (maximize the number of the touched beacons), a better option might be to discard the particular ensemble instance and form a new one using more fresh data. The maximum staleness has to be defined per particular data elements as each of them can “stale” in a different pace. The smallest radius defines a minimum range around a component to which the data has to be disseminated. In case the radius is smaller than the neighborhood defined by the communication boundary specified in the ensemble, then the smallest radius is disregarded (this happens for instance if an island is very small).

Figure 9.2 graphically demonstrates the relation between the ideal system and the boundaries of the actual system. There are multiple components (robots and beacons) and they are deployed to physical nodes. (Note that nodes are not represented in the DSL as the particular deployment is not specified at this level of abstraction.) In order to allow formation of ensembles, the deployment infrastructure has to disseminate information between the nodes. Usage of MANETs and Gossip (as outlined in Section 9.1 and 9.2) is considered. The communication boundary, as described in the ensemble definition, represents a farthest limit for the data dissemination. As it is constructed as an over-approximation of the membership condition, all potential sets of components which satisfy the membership condition are deployed on nodes within communication boundary. Note that membership condition can select multiple sets, which are then ranked by fitness. This is the case in Figure 9.2, where the fitness is shown as the number set in green.

The communication parameters (staleness, smallest radius) can further limit the actual dissemination, thereby establishing the dynamic communication boundary. In the example, there are three possible ensembles to be formed for a single particular beacon. Even though the ensemble with the fitness 40 is the best one (the best fitness) it is not considered as one of its component is beyond dynamic communication boundary.

9.4.3 Communication Parameters

Assuming the communication in ensembles is realized via an optimized Gossip algorithm on top of a MANET, the dynamic communication boundary can be controlled by several communication parameters, which have positive effect on network utilization or system utility. The system utility denotes the effectiveness of the system in reaching its goal – in the example, this can be for instance the sum of distances of the beacons which were successfully touched by two robots over the system lifetime. As such, the system utility is something which can possibly be evaluated only once the system finished its operation. However, it is assumed that the system is designed in such a way that maximizing the sum of ensemble fitness values at any given point in time brings the system close to its highest utility, enabling calculation of system utility at runtime.

Based on an initial round of experiments, the following key communication parameters were identified: (i) rebroadcast period, (ii) rebroadcast radius, (iii) max. packet age. While the semantics of the rebroadcast period is obvious, the latter two need more explanation: Together they determine when to stop propagating packets in the network based on their source location and timestamp. Rebroadcast radius is expressed as the spatial distance from the source of the packet in question, while the max. packet age limits the maximal packet age which is still considered for rebroadcasting based on its timestamp.

It should be emphasized that setting communication parameters per ensemble instance cannot be done in isolation, i.e., not considering how communication in other ensemble instances would be affected. In other words, optimizing communication in a particular ensemble instance may cause increase in network load, thus limiting communication in other ensemble instances. In order to optimize the overall system utility it is necessary to set network parameters with respect to all ensemble instances currently present in the system.

CHAPTER 10

Real-time Analysis

Considering sCPS often there are limits on maximum processing time of a particular operation in the system. Assuming the system is in control of heavy mobile devices such as in the case of vehicular or robotic systems these requirements are frequent and require to think of real-time design at the level of system architecture. Any heavy moving device can crash into another one or harm a human being. A vehicle and vehicular systems are a good model for practising design of such systems.

In this chapter the goal G2 of this thesis in addressed in terms of coordination in an real-time EBCS. The challenge of implementing real-time system can be split in two. First part is to implement a hard real-time runtime and predict worst case execution time of all tasks in the system. Once this is done a deep analysis of different timings in the system needs to be done in order to guarantee that the deadlines are always met. Considering the DEECo based systems the CDEECo++ provides the hard real-time capable runtime, but an approach to timing analysis of EBCS needs to be drafted.

In the following chapter a case study that exemplifies necessary workflow leading to proper real-time design of task execution and network communication is described. An ICS system was chosen as it demonstrates a typical system dealing with many moving entities connected using unreliable wireless network. Correctness of the described calculations are validated in Chapter 13 where a simulation of the system is described including measured delays.

10.1 Intelligent Crossroad System Case Study

An application scenario in the context of VANET [41] and autonomous vehicles is considered, where an ICS optimizes the *car throughput* at a road crossing. This is illustrated in Figure 10.1, where cars approach a two-lane crossing managed by the ICS.

The idea is to replace traffic lights to a great extent by using Car to Infrastructure (C2I) communication and synchronizing in what order cars cross the intersection for an uninterrupted flow in all directions. Note that there are different ways of implementing this case study. In particular, one can design the ICS to take full control over cars adjusting their speed and steering as considered in [42]. However, this concentrates almost all computation workload at the ICS – making more expensive hardware necessary – and requires cars to be enabled for remote operation.

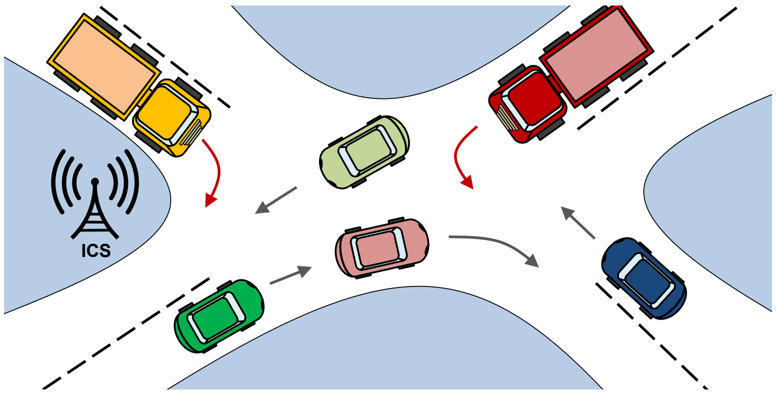


Figure 10.1: Intelligent crossroad system (ICS)

In this study, an alternative approach is followed, where the ICS computes – taking traffic conditions and regulations into account – and *assigns* a speed to an approaching car at the intersection. The car will then have to keep this speed constant to cross the intersection without stopping. This solution is more viable to implement the proposed case study, since it does not require cars to be modified for a remote operation. On the contrary, each (autonomous) car is responsible for driving along its own trajectory and maintaining its speed as assigned to it. If necessary, however, a car may stop to avoid a crash or accident.

The ICS assigns speeds to cars and these report their current speeds back to the ICS via C2I communication. As discussed later, the ICS needs to keep track of cars' speeds in order to detect potential hazards and trigger safety mechanisms. To this end, two operation modes are foreseen:

(i) Automatic mode: This is the default mode where all cars at the crossing behave as expected, i.e., they maintain their corresponding speeds as computed by the ICS. Here, the highest possible *throughput* is reached provided that speed limit regulations are observed and safety of all traffic participants can be guaranteed.

(ii) Manual mode: This is the exception mode where one or more cars do not maintain their computed speeds and/or there is no communication between a car and the ICS, e.g., conventional cars with a human driver. In this mode, the ICS works as standard traffic lights.

The *region of influence* is defined by the area in which the ICS *controls/monitors* all approaching cars. It is assumed that this area consists of a 50 m radius around the intersection. In this region, vehicles are prioritized such that their priorities increase as they get closer to the center of the intersection and drop when they move away of it. Some of the vehicles at the ICS might also be *privileged* such as, for example, ambulances or police cars in an emergency situation, etc.

The ICS can detect when a car enters the region of influence, e.g., by radar, pressure sensors, etc. If no communication is received from one car after it has entered the region of influence (in particular, a car's intended direction, its current speed, etc. are needed), the ICS assumes that either there has been an error or it is a conventional car with no C2I communication and switches to manual mode.

The same happens if communication is lost to one or more cars; a more detailed analysis of this is given in Section 10.5.

Pedestrians can be easily handled in the manual mode. In the automatic mode, the ICS can detect when pedestrians stand at the crossing for some time, e.g., by pressure sensors, request buttons, etc., and stop the traffic to let them cross in a safe manner. Again, each car is responsible for itself and should be able to react to unpredicted situations, e.g., performing an emergency break, according to valid traffic regulations.

This scenario exhibits different challenges that need to be faced when designing dynamic distributed systems. One of those challenges is the description of architectural changes that occur during runtime. In our case study, cars/vehicles arrive to and leave the system at different points in time, their priorities vary according to their distances to the crossing, etc. Such details need to be properly reflected in the system design.

Furthermore, this scenario exhibits real-time requirements imposed to the system. In particular, it is required that the *reaction time* between a car and the ICS is kept below a certain upper bound in order to ensure an appropriate responsiveness of the overall system, where unreliable communication needs to be considered. In turn, meeting those real-time requirements allows us to guarantee safety, which translates into a collision-free crossing in our case study.

Lastly, since it is not possible to guarantee a fully reliable communication, the system has to be designed to be self-adaptive. This way, the system switches to manual mode when it realizes that real-time requirements cannot be met. To this end, as discussed later, a *watchdog timer* at all components (cars and ICS) is configured that triggers a switch to manual mode.

10.2 Modeling with DEECo

10.2.1 Components

Listing 10.1 depicts Vehicle and ICS components using a DSL description[1]. Processes of each component can be executed in response to a timer event (i.e., periodic execution) or as a reaction to a change in one of its attributes. In our example, the *Vehicle* component has a process that sets/updates the speed of the car or vehicle. This is repeated periodically every 5 ms (see line 22). As another example the *ICS* process, shown in Listing 10.1, determines whether there are privileged vehicles in its region of influence (lines 34-40) and is executed whenever the number of vehicles changes (line 40). Once processes are released (by DEECo runtime environment), these are handed over to the platform's Operating System (OS), which is responsible for scheduling them according to a desired policy – see Section 10.2.4.

```
1 interface MovingUnit:
2   id, time, crossingId, crossingDistance, crossingDirection, speed, privileged,
     mode
3
4 interface MovingUnitAggregator:
```

[1]Note that this DSL specification serves for demonstration only. Later it is discussed how to derive a C++ implementation from this specification, which can then be used on embedded devices.

```
  id, time, vehicles, speeds, mode

component Vehicle features MovingUnit
  knowledge:
    id: 42,
    time: 1440691842456 ms,
    crossingId: 12
    crossingDistance: 35 m,
    crossingDirection: South-West,
    speed: 50 Km/h,
    privileged: FALSE,
    mode: AUTOMATIC,
    ...
  process UpdateSpeed:
      in speed
    function:
      Actuators.setSpeed(speed);
    scheduling: periodic( 5 ms )
  ...

component ICS features MovingUnitAggregator
  knowledge:
    id: 12,
    time: 1440691842458 ms,
    vehicles: [...],
    speeds [...]
    privileged: [...],
    mode: AUTOMATIC,
    ...
  process findPrivilegedVehicles:
      in vehicles, inout privileged
    function:
      for (v : vehicles)
        if (v.privileged)
          privileged.add(v)
    scheduling: triggered( changed(movingUnits) )...
```

Listing 10.1: DEECo component definitions based on a DSL

10.2.2 Ensembles

The ensembles are considered to be bipartite ensembles with roles defined by the interfaces, in our example, *MovingUnit* and *MovingUnitAggregator* In Listing 10.2, the coordinator role is determined by the interface definition *MovingUnitAggregator* and the member role by *MovingUnit.* The membership condition further constraints the ensemble members to those, which are located no more than 50 m from the coordinator's location. Then, according to the knowledge exchange description, the coordinator's *movingUnits* attribute is updated with information about all components that fulfill the membership condition (which is checked every $p_{ens,i}$ time units – see line 8 – with i being an index representing the component). This way, the *ICS* is aware only of those vehicles, which are currently in its close proximity.

```
1 ensemble UpdateMovingUnitInformation:
2   coordinator: MovingUnitAggregator
3   member: MovingUnit
4   membership:
5     coordinator.id = member.crossingId, member.crossingDistance < 50 m
6   knowledge exchange:
7     coordinator.movingUnits.add({member})
8   scheduling: periodic( p_ens,i )
```

Listing 10.2: A DSL example of an ensemble definition.

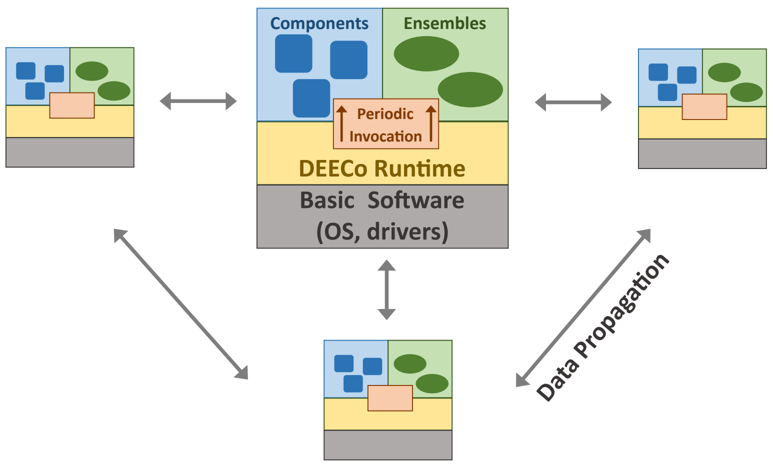


Figure 10.2: DEECo distributed deployment.

10.2.3 DEECo's Deterministic Semantics

Technically, the runtime environment periodically *propagates* ensemble-relevant knowledge to all other components or nodes in the system – note that Gossip-based algorithms [35] might be used for this purpose. In our case study, ensemble-relevant data are the car's distance to the crossing, its speed, and its intended direction, etc. This is used to evaluate whether cars are heading in the direction of the crossing or not.

Each node then keeps relevant reference knowledge from all other nodes from which it has received data. In other words, components or nodes join (and leave) the system in an implicit manner without performing any *handshaking*. Since ensemble-relevant information is present at all nodes, DEECo runtime environment performs a local decision of an ensemble membership condition. If this holds true, the local reference knowledge of the remote components involved is used for the *knowledge exchange* process.

In this way, DEECo semantics separates *decision taking* (i.e., ensemble formation and its eventual knowledge exchange) from *information sharing* (i.e., knowledge propagation) at the components. Since a DEECo component takes decisions based on locally available data, it does not need to synchronize with other components in the system. On the one hand, this has the advantage of high flexibility and adaptability. On the other hand, clearly, local data might get outdated at the different nodes, which needs to be analyzed carefully as illustrated in Section 10.3.

10.2.4 Implementation and Deployment

The implementation and distributed deployment of hard real-time DEECo systems are supported by the CDEECo++ framework. This framework maps DEECo concepts to C++ and constitutes our runtime environment (taking care of periodically propagating knowledge, performing ensemble formation, performing knowledge exchange if applicable, etc.). As depicted in Figure 10.2, CDEECo++ relies on an OS providing hardware abstraction and other services. Clearly, this OS needs to support real-time behavior, i.e., real-time scheduling, interrupt handling, etc., to be used in safety-critical applications. The CDEECo++ and its runtime platform are further described in Chapter 11.

10.3 Closed Loop Reaction Time

In this section DEECo closed-loop reaction time in the worst case is analyzed. This is defined as the maximum delay that it takes a DEECo-based system to react to changes in the environment. The term *closed-loop* reflects the fact that DEECo components interact with one another.

For example, if a component A experiences a change in its internal states, e.g., due to one or more physical variables measured by its sensors, this will take some time to reach another component B – connected by ensembles – in the system. Similarly, component B's reaction to the change in component A will take some additional time to reach back component A. The sum of these two times is the closed-loop delay between A to B. In other words, component A and B form a loop.

For ease of exposition, case study is used first and then the results are generalized to make them independent of the application. In the ICS case study, knowledge needs to be exchanged from a car to the ICS and from the ICS back to the car for the system to work as expected. However, knowledge exchange happens based on *local data* when the corresponding ensemble condition is evaluated to true at both the ICS and the car nodes separately.

As discussed above, knowledge is propagated (from the car to the ICS and vice versa) and the ensemble membership check is performed (at the car and at the ICS) on a periodic basis. Let $\hat{p}_{pro}$ denote the maximum period with which knowledge is propagated by any node in the system. Similarly, let $\hat{p}_{ens}$ be the maximum period with which ensembles are formed at any node in the system. That is:

$$\begin{aligned}\hat{p}_{pro} &= \max_{\forall i}\left(p_{pro,i}\right),\\ \hat{p}_{ens} &= \max_{\forall i}\left(p_{ens,i}\right),\end{aligned}$$

where $p_{pro,i}$ and $p_{ens,i}$ are the knowledge propagation and the ensemble formation period of a node i, respectively, with i being an index that identifies the corresponding component/node.

In order to eliminate the need for synchronization and handshaking between components, these two processes in DEECo are not synchronized with one another and, hence, the following conditions are present in the worst case:

(i) A car propagates its knowledge to the ICS immediately after a membership decision has been performed at the ICS. As a result, data is received $\hat{p}_{ens}$ time later at the ICS, when the next membership decision is performed.

(ii) In a similar manner, the ICS propagates its knowledge to the corresponding car just after a membership decision has been performed at the car. As a result, data is received $\hat{p}_{ens}$ time later at the car too.

(iii) The knowledge of the car changes immediately after knowledge has been propagated to the ICS. As a result, the *current* knowledge is propagated with a delay $\hat{p}_{pro}$ from the car to the ICS, when a new propagation is performed.

(iv) The ICS's knowledge changes immediately after knowledge has been propagated to the car. Hence, the *current* knowledge is not propagated until a new propagation is started $\hat{p}_{pro}$ time later.

As a result, in the worst case, a delay due to the asynchronous nature of the DEECo framework is given by the following expression:

$$2 \times \hat{p}_{pro} + 2 \times \hat{p}_{ens}. \tag{10.1}$$

In addition, there is also a process running at the ICS which computes the speed for the car that guarantees no collisions at the current traffic situation. This process is triggered when a knowledge exchange is executed at the ICS (i.e., when the ensemble membership condition is evaluated to true between the car and the ICS). Let r_{ICS} denote the Worst-Case Response Time (WCRT) of this process. Analogously, there is a process running at the car, which applies the new speed values to the *physical* car. This process is also triggered when a knowledge exchange happens at the car and its WCRT is r_{car}.

As a result, the worst-case delay D_{max} for a closed-loop reaction in DEECo, i.e., a reaction of a car to an input from the ICS computed based on the car's current knowledge, is given by the following equation also illustrated in Figure 10.3:

$$D_{max} = 2 \times \hat{p}_{pro} + 2 \times \hat{p}_{ens} + c_{ICS} + c_{car} + r_{ICS} + r_{car}, \tag{10.2}$$

where c_{car} is the communication delay from the car to the ICS and c_{ICS} is the communication delay from the ICS to the car.

Since there is interference by other messages (from other cars), c_{car} is the maximum possible communication delay in the network. However, from the ICS to the car, there is no interference – assuming a full-duplex communication channel – and the communication delay c_{ICS} is equal to the transmission time, since the ICS does not compete for accessing the network.

The term $2 \times \hat{p}_{pro} + 2 \times \hat{p}_{ens}$ in Equation (10.2) is clearly intrinsic to DEECo and does not depend on the application, but rather on how components are configured. In addition, $c_{ICS} + c_{car}$ is the total communication delay between the ICS and a car, whereas $r_{ICS} + r_{car}$ is the delay due to computation at the ICS and at the car. As a result, DEECo's closed-loop delay in the worst-case can be generalized, i.e., made independent of the application under consideration, as follows:

$$\bar{D}_{max} = 2 \times \hat{p}_{pro} + 2 \times \hat{p}_{ens} + C_{max} + R_{max}, \tag{10.3}$$

where C_{max} is the sum of the worst-case communication delay between any two DEECo components and R_{max} is the sum of the WCRTs of computation processes involved at the corresponding components or nodes.

10.4 Real-Time Analysis

The purpose of performing a real-time analysis is to guarantee that timing constraints can be met by the system, which is required in safety-critical applications. In case of DEECo, this boils down to checking that $\bar{D}_{max}$ as per Equation (10.3) is below a given time upper bound, which stems from physical processes involved

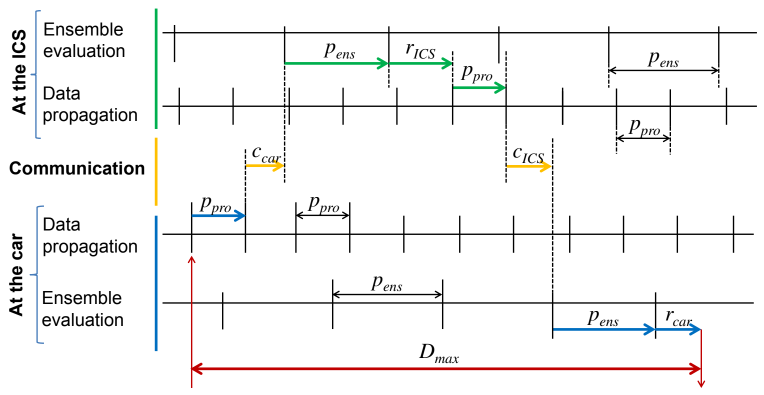


Figure 10.3: Composition of DEECo's closed-loop delay D_{max}: In the worst case, data may change immediately after knowledge has been propagated at the car. This data may also arrive after an ensemble formation has been performed at the ICS. In addition, computation at the ICS may finish immediately after knowledge propagation and the resulting data then reaches the car just after the end of an ensemble formation.

and needs to be known to or obtained by the designer. In turn, $\bar{D}_{max}$ depends on C_{max} and R_{max}, i.e., on the delays incurred by communication and computation processes involved – which are closely related to the technologies and techniques used for implementing the system.

All in all, the real-time analysis consists of the following steps that we illustrate next in the context of our case study: i) obtaining the worst-case computation delay, ii) obtaining the worst-case communication delay, iii) determining system constraints, and iv) obtaining a feasible DEECo configuration.

10.4.1 Obtaining the Worst-Case Computation Delay

As discussed previously, CDEECo++ – DEECo's runtime environment – is executed on top of an OS at each node in the system. Among others, CDEECo++ is in charge of releasing component's processes as specified in the component description. Clearly, to be used in safety-critical applications, CDEECo++ relies on specific technologies that make real-time scheduling and real-time communication possible. In particular, the OS needs to support real-time scheduling; otherwise, it will not be possible to guarantee real-time behavior.

The techniques for schedulability analysis, i.e., testing whether processes meet their deadlines at the different nodes, strongly depend on the scheduling algorithm used. As mentioned above, CDEECo++ makes use of FreeRTOS, which supports fixed-priority scheduling and allows for the *rate monotonic* policy [43]. That is, processes are given fixed priorities according to the following rule: The shorter a process's period is, the higher the priority assigned to it.

Let $\mathbf{T}$ denote set of processes on a given node. Further, τ_i is a process that belongs to $\mathbf{T}$ where e_i denotes its Worst-Case Execution Time (WCET) and p_i denotes its period of repetition. For $\mathbf{T}$ to be schedulable, the following has to hold for each τ_i and $1 \leq i \leq |\mathbf{T}|$:

$$\sum_{\forall \tau_j \in \widehat{\mathbf{T}}} \left\lceil \frac{p_i}{p_j} \right\rceil e_j \leq p_i, \tag{10.4}$$

where $|\mathbf{T}|$ is the number of elements in $\mathbf{T}$ and $\widehat{\mathbf{T}}$ denotes the subset of processes from $\mathbf{T}$ which have a higher priority than the corresponding τ_i.

This expression means that for each process τ_i to be schedulable (and, hence, for $\mathbf{T}$ to be schedulable), the sum of all executions of higher-priority processes in a time interval equal to p_i plus its own execution e_i should be less than its deadline p_i. Note that Equation (10.4) is sufficient but not necessary. A sufficient and necessary test can be achieved by response time analysis [44]; however, the sufficient test of Equation (10.4) is enough for the purpose of this paper. Now, since the following holds:

$$\sum_{\forall \tau_j \in \widehat{\mathbf{T}}} \left\lceil \frac{p_i}{p_j} \right\rceil e_j \;\leq\; \sum_{\forall \tau_j \in \widehat{\mathbf{T}}} \left(\frac{p_i}{p_j} + 1 \right) e_j,$$

Equation (10.4) can be reshaped to:

$$\sum_{\forall \tau_j \in \widehat{\mathbf{T}}} \frac{e_j}{p_j} + \frac{\sum_{\forall \tau_j \in \widehat{\mathbf{T}}}(e_j)}{p_i} \leq 1. \tag{10.5}$$

Clearly, if Equation (10.5) holds, Equation (10.4) will also hold. However, Equation (10.5) is easier to compute and operate with.

In our case study, to obtain $R_{max} = r_{ICS} + r_{car}$, let $\mathbf{T}_{ICS}$ denote set of computation processes at the ICS. Further, we assume that a process τ_i is the one involved in the closed-loop delay, i.e., the one computing the new speed for a given car. Considering that $\widehat{\mathbf{T}}_{ICS}$ is the subset with higher or equal priority processes than τ_i at the ICS, the r_{ICS} can be computed as follows:

$$r_{ICS} = \left(\sum_{\forall \tau_j \in \widehat{\mathbf{T}}_{ICS}} \frac{e_j}{p_j} \right) p_i + \sum_{\forall \tau_j \in \widehat{\mathbf{T}}_{ICS}} e_j. \tag{10.6}$$

Analogously, at the car, r_{car} can be obtained as follows:

$$r_{car} = \left(\sum_{\forall \tau_j \in \widehat{\mathbf{T}}_{car}} \frac{e_j}{p_j} \right) p_i + \sum_{\forall \tau_j \in \widehat{\mathbf{T}}_{car}} e_j. \tag{10.7}$$

10.4.2 Obtaining the Worst-Case Communication Delay

Similar to computation delay, communication delay strongly depends on the underlying technologies and techniques used. Since VANETs are usually based on wireless communication [45], we assume that IEEE 802.1Q is the underlying protocol, which provides mechanisms to prioritize *messages* [46]. In general, we will normally have a number of Access Point (AP) which are connected to a full-duplex switch via IEEE 802.3 Ethernet.

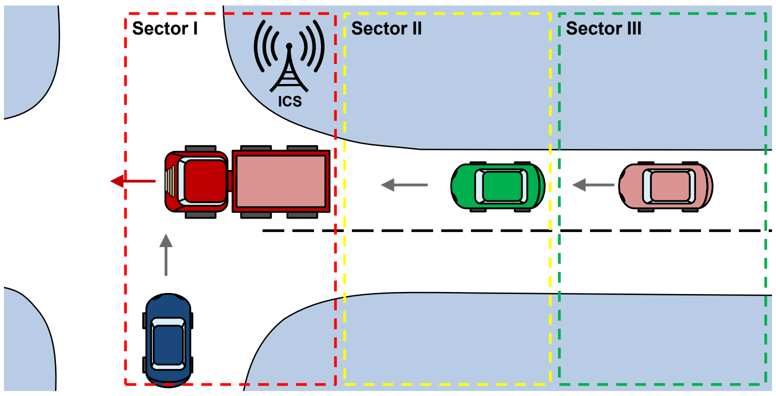


Figure 10.4: Priorities are given according to the proximity to the intersection

In this section, it is assumed that communication to the AP is collision-free as per some VANET scheme – later this assumption is removed to analyze the effect of packet loss. For example, Space Division Multiple Access (SDMA) has been proposed [47], [48], which guarantees a collision-free communication in busy intersections by dividing a road into sectors and assigning different time slots to each such sector. Other solutions combine special antennas with also a TDMA scheme to reduce packet loss [49], [50].

Now, assuming that wireless network provides 100 Mbps and that messages are at most 1 Kbit (1024 bits), then the transmission time c_W on the wireless network is at most – considering a 144-bit protocol overhead:

$$c_W = \frac{1024 + 144}{100\,\text{Mbps}} = 11.68\,\mu\text{s}.$$

However, the switch then sends messages to the ICS according to their priorities. Considering that the ICS's region of influence is divided into sectors with different priorities – see Figure 10.4. Cars that are in the first sector (e.g., within 10 m from the intersection) have higher priority than cars in the second sector (e.g., from 10 m to 20 m) and so on. At a given point in time, if a car is in more than one sector simultaneously, it will be assigned the highest priority among those sectors. The switch then sends messages to the ICS according to these priorities.

Further the communication segment between the switch and the ICS will be analyzed. To this end, let $\mathbf{M}$ denote the set of all messages being sent to the ICS over the switch. Further, m_i denotes one such message in $\mathbf{M}$ where c_i is its transmission time – note that c_i is constant for a given i which results from the amount of bits to be sent and the bandwidth of the communication channel – and z_i denotes the minimum inter-arrival time between two consecutive such messages. The deadline of a message is also given by z_i.

Generally, for all messages in $\mathbf{M}$ to meet their deadlines, the following has to hold for $1 \leq i \leq |\mathbf{M}|$, where $|\mathbf{M}|$ is the number of elements in $\mathbf{M}$:

$$b_i + \sum_{\forall m_j \in \widehat{\mathbf{M}}} \left\lceil \frac{z_i}{z_j} \right\rceil c_j \leq z_i, \tag{10.8}$$

where $\widehat{\mathbf{M}}$ is the subset of $\mathbf{M}$ with higher- or equal priority messages than m_i and b_i denotes *blocking time* on the communication channel. That is, whenever a message needs to be sent, a lower-priority message might eventually be using the communication channel. Since this lower-priority message cannot be interrupted, there is a blocking time on the bus. Clearly, in worst case, b_i is given by the maximum transmission time among all lower-priority messages:

$$b_i = \max_{l=i}^{|\mathbf{M}|}(c_l). \tag{10.9}$$

Considering that $\sum_{\forall m_j \in \widehat{\mathbf{M}}} \left\lceil \frac{z_i}{z_j} \right\rceil \leq \sum_{\forall m_j \in \widehat{\mathbf{M}}} \left(\frac{z_i}{z_j} + 1 \right)$ holds, the ceiling function can be removed and approximate Equation (10.8) as shown below:

$$\sum_{\forall m_j \in \widehat{\mathbf{M}}} \frac{c_j}{z_j} + \frac{b_i + \sum_{\forall m_j \in \widehat{\mathbf{M}}}(c_j)}{z_i} \leq 1. \tag{10.10}$$

To demonstrate this, assuming that the IEEE 802.3 Ethernet link between the switch and the ICS has a bandwidth of 1 Gbps. If messages have a length of at most 1 Kbit (1024 bits), and the protocol overhead is of 144 bits, the transmission time c_i of a message m_i is given by $\check{c}_E$:

$$\check{c}_E = \frac{1024 + 144}{1\,\text{Gbits}} = 1.168\,\mu\text{s}.$$

Further, with help of Equation (10.10), the transmission time on the IEEE 802.3 Ethernet link can be computed taking contention by higher- and equal priority messages into account, which is denoted by $\hat{c}_E$:

$$\hat{c}_E = \left(\sum_{\forall m_j \in \widehat{\mathbf{M}}} \frac{c_j}{z_j} \right) z_i + b_i + \sum_{\forall m_j \in \widehat{\mathbf{M}}} c_j. \tag{10.11}$$

In addition to the transmission time, there is always a delay at the AP and at switches in IEEE 802.3 Ethernet – denoted by e_{AP} and e_{SW} respectively, which accounts for buffering and routing tasks. This is typically in the order of $2\,\mu$s.

$C_{max} = c_{ICS} + c_{car}$ can now be obtained. To this end, recall that there are two IEEE 802.3 Ethernet links: one from the AP to the switch and another from the switch to the ICS. The ICS does not suffer from contention at the communication channel, since the connection from and to the APs is assumed to be full duplex. As a result, c_{ICS} is given by:

$$c_{ICS} = 2 \times \hat{c}_E + c_W + e_{SW} + e_{AP}. \tag{10.12}$$

On the other hand, cars share the communication channel and, hence, they may have contention at the communication channel leading to a c_{car} as follows:

$$c_{car} = 2 \times \hat{c}_E + c_W + e_{SW} + e_{AP}. \tag{10.13}$$

10.4.3 Determining System Constraints

Timing constraints are clearly derived from the application. It is considered that a car needs to be provided with new speed values at every single meter of its trajectory (taking vehicle dynamics into account such inertia, braking distance, etc.). If a car's speed is at maximum 50 Km/h (assuming an urban scenario), the time $t_{1\mathrm{m}}$ that it needs to cover 1 m of its trajectory is:

$$t_{1\mathrm{m}} = \frac{1\,\mathrm{m} \times 3600\,\mathrm{s/h}}{50 \cdot 10^3\,\mathrm{m/h}} = 72\,\mathrm{ms}. \tag{10.14}$$

On the other hand, the computation and communication overhead depends on the number of components, in particular, cars/vehicles in the system, which is the second constraint from the application. Clearly, the more cars enter the ICS's region of influence, the more computation and communication overhead there will be. To compute the maximum possible number of cars at the crossing, it is assumed that a car is at least 2 m long and that there is a least a 1 m distance between any two cars. As a result of this, in the worst possible case, the number of cars n approaching the intersection from all directions is given by the following equation:

$$n = 4 \times \left\lceil \frac{50\,\mathrm{m}}{3\,\mathrm{m}} \right\rceil = 68. \tag{10.15}$$

An Equation (10.15) can be used to configure $\hat{p}_{ens}$ and $\hat{p}_{pro}$ in the next section and the timing constraint as per Equation (10.14) to perform a feasibility analysis as discussed later.

10.4.4 Obtaining a Feasible DEECo Configuration

There will be at most 68 different ensemble instances (between the ICS and each of the cars) at the ICS – see Equation (10.15). In addition, there will be 68 processes to compute new speed values for each car. Since the ensemble membership check triggers a knowledge exchange – recall that knowledge exchange is based on locally available data and that knowledge/data propagation (from the ICS to the cars and vice versa) is a separate and asynchronous process – when evaluated true, it can be assumed that in the worst case all 68 ensemble processes trigger their corresponding computation processes simultaneously. In addition, there will be one knowledge propagation process for the ICS[2].

Assuming that all processes have a WCET $e_i = 25\mu\,\mathrm{s}$ (note that most these processes consist in checking logic conditions, assigning pointers to given memory spaces, etc., or are simple computations), an Equation (10.5) can be applied to the ICS as follows:

$$\frac{0.025\,\mathrm{ms}}{\check{p}_{pro}} + 68 \cdot \frac{2 \times 0.025\,\mathrm{ms}}{\check{p}_{ens}} + \frac{(0.025\,\mathrm{ms} + 68 \cdot (2 \times 0.025\,\mathrm{ms})}{\check{p}_{ens}} \leq 1, \tag{10.16}$$

where $\check{p}_{pro}$ and $\check{p}_{ens}$ are the minimum periods with which knowledge is propagated and with which ensembles are formed in the system. That is:

$$\check{p}_{pro} = \min_{\forall i} \left(p_{pro,i}\right),$$
$$\check{p}_{ens} = \min_{\forall i} \left(p_{ens,i}\right).$$

[2]Note that the knowledge propagation from cars does not produce any overhead at the ICS, but at the respective cars.

Note that if Equation (10.16) holds for $\check{p}_{pro}$ and $\check{p}_{ens}$, it will also hold for any $p_{pro,i}$ and $p_{ens,i}$. The following value for $\check{p}_{ens}$ is obtained assuming $2 \times \check{p}_{pro} = \check{p}_{ens}$, i.e., that knowledge propagation is done twice as frequently as any ensemble membership check[3]:

$$\check{p}_{ens} \geq 6.88\,\text{ms},$$

and, hence, $\check{p}_{pro}$ has to be greater or equal to 3.44ms. Note that there cannot be a $p_{ens,i}$ that is less than $\check{p}_{ens}$. Similarly, $p_{pro,i}$ is bounded from below by $\check{p}_{pro}$. Otherwise, Equation (10.16) will not hold.

On the other hand, the upper bounds $\hat{p}_{ens}$ and $\hat{p}_{pro,i}$ need to fulfill the system's feasibility condition. That is, DEECo's closed-loop delay must be at most equal to the timing constraint $t_{1\text{m}}$:

$$D_{max} \leq t_{1\text{m}}. \tag{10.17}$$

D_{max} is DEECo's closed-loop delay as per Equation (10.2), i.e., where $R_{max} = r_{ICS} + r_{car}$ and $C_{max} = c_{ICS} + c_{car}$ in general expression $\bar{D}_{max}$ as given in Equation (10.3).

Choosing $\hat{p}_{ens} = 14\,\text{ms}$ – twice as much as $\check{p}_{ens}$ – and hence $\hat{p}_{pro} = 7\,\text{ms}$, i.e., there is $2 \times \hat{p}_{pro} = \hat{p}_{ens}$. With these values of $\hat{p}_{ens}$ and $\hat{p}_{pro}$, it is verified that Equation (10.17) can be met. If not, new values of $\hat{p}_{ens}$ and $\hat{p}_{pro}$ need to be chosen – clearly, these should be greater than or equal to their lower bounds $\check{p}_{ens}$ and $\check{p}_{pro}$ respectively.

To test whether Equation (10.17) holds for the chosen $\hat{p}_{ens}$ and $\hat{p}_{pro}$, it is necessary to compute the corresponding r_{ICS}, r_{car}, c_{ICS}, and c_{car}. It can be computed r_{ICS} using Equation (10.6) and assuming that all processes are respectively released either at a $\hat{p}_{ens}$ or a $\hat{p}_{pro}$ rate.

$$r_{ICS} = 14\,\text{ms} \cdot \left(\frac{25\,\mu\text{s}}{7\,\text{ms}} + 68 \cdot \frac{2 \cdot 25\,\mu\text{s}}{14\,\text{ms}}\right) + 25\,\mu\text{s} + 68 \cdot (2 \cdot 25\,\mu\text{s}) \approx 7\,\text{ms} \tag{10.18}$$

Similarly, it can be computed r_{car} using Equation (10.7). In the car, there are only one ensemble process, one process to update the speed with the new one assigned by the ICS, and a knowledge propagation process. Again, it is assumed that the ensemble process triggers the knowledge exchange process at cars. Assuming again $e_i = 25\,\mu\text{s}$, results in:

$$r_{car} = 14\,\text{ms} \cdot \left(\frac{25\,\mu\text{s}}{7\,\text{ms}} + \frac{2 \cdot 25\,\mu\text{s}}{14\,\text{ms}}\right) + 25\,\mu\text{s} + 2 \cdot 25\,\mu\text{s} = 0.18\,\text{ms}$$

Now $\hat{c}_E$ needs to be computed, i.e., the transmission time on the IEEE 802.3 Ethernet link taking contention into account, using Equation (10.11):

$$\hat{c}_E = \hat{p}_{pro} \times 68 \times \frac{1.168\,\mu\text{s}}{\hat{p}_{pro}} + 68 \times 1.168\,\mu\text{s} = 158.85\,\mu\text{s},$$

where c_i and z_i have been replaced by c_E and $\hat{p}_{pro}$ respectively. Further, b_i is zero according to Equation (10.9), since it is considered the lowest priority

[3]This is a design decision that needs to be taken. In general, since ensemble membership checks rely on local knowledge, it is meaningful that knowledge be updated as often as necessary to guarantee desired functionality.

message for $\hat{c}_E$, i.e., the one suffering the most contention by other messages. The communication delay from and to the ICS, can be computed using Equation (10.12) and Equation (10.13):

$$c_{ICS} = 18.02\,\mu\text{s}, c_{car} = 333.38\,\mu\text{s}.$$

Finally, from Equation (10.2), follows that:

$$D_{max} = 2 \cdot 7\,\text{ms} + 2 \cdot 14\,\text{ms} + 334\,\mu\text{s} + 18.1\,\mu\text{s} + 7\,\text{ms} + 0.18\,\text{ms} \approx 50\,\text{ms},$$

which is less than $t_{1\text{m}} = 72ms$ – see Equation (10.14). This means ICS is able to meet all deadlines in the worst case.

10.5 Robustness to Unreliable Communication

In general, if many consecutive packets are lost on the communication channel, the system will experience malfunction putting safety into risk. In this section, the number of consecutive packets that can be lost at maximum without compromising safety is determined. In other words, the system's *robustness* under unreliable communication is quantified.

As discussed above, Equation (10.2) states the worst-case delay incurred by DEECo-based ICS in case of a fully reliable communication. This is obtained considering that data at a car can be updated immediately after knowledge has been propagated – for the reason that processes updating and propagating data are not synchronized with each other. As a result, this data will be sent the next time a knowledge propagation is performed, i.e., $\hat{p}_{pro}$ time later. If now this packet is lost on the communication channel, data will incur an additional delay equal to $\hat{p}_{pro}$. Further, if k_{car} denotes the number of consecutive packets that are lost, then data from the car to the ICS incurs the following delay:

$$\hat{p}_{pro} + k_{car} \times \hat{p}_{pro} + c_{car}, \tag{10.19}$$

where again c_{car} denotes the delay on the communication channel from the car to the ICS.

In a similar manner, if k_{ICS} denotes the number of consecutive packets that are lost from the ICS to the car, then data from the ICS incurs following delay to reach the car:

$$\hat{p}_{pro} + k_{ICS} \times \hat{p}_{pro} + c_{ICS}, \tag{10.20}$$

where, as discussed above, c_{ICS} is delay on the communication channel from the ICS to the car.

Let $k = k_{car} + k_{ICS}$ denote the total number of packets lost between the ICS and the car. Equation (10.19) can be combined with Equation (10.20) to determine the closed-loop delay of the system in case of unreliable communication:

$$\hat{D}_{max} = (2 + k) \times \hat{p}_{pro} + 2 \times \hat{p}_{ens} + c_{ICS} + c_{car} + r_{ICS} + r_{car}, \tag{10.21}$$

where again r_{ICS} and r_{car} denote the maximum delay to finish computation at the ICS and the car respectively. Note that Equation (10.21) reduces to Equation (10.2) for $k = 0$, i.e., when no packets are lost on the communication channel.

Using the values of $\hat{p}_{pro}$, $\hat{p}_{ens}$, c_{car}, c_{ICS}, r_{car}, and r_{ICS} computed in the previous section, the maximum k can be determined, that can be tolerated without affecting the system's functionality and safety. That is the maximum k that makes $\hat{D}_{max}$ be at most equal to $t_{1\mathrm{m}}$ as per Equation (10.14). This maximum is denoted k by k_{max}:

$$k_{max} = \left\lfloor \frac{t_{1\mathrm{m}} - D_{max}}{\hat{p}_{pro}} \right\rfloor = 3. \tag{10.22}$$

Equation (10.22) indicates that the sum of k_{car} and k_{ICS}, each of which represents the number of consecutive packets being lost in one or the other direction, cannot be more than 3 for the system to operate correctly.

10.5.1 Safety Mechanisms

Note that the previous results can be used to implement safety mechanisms at the ICS and at cars. In particular, whenever communication is lost for longer than $t_{1\mathrm{m}}$ time between the ICS and any car in the system or vice versa, both the ICS and the car switch to manual mode, i.e., the ICS starts working as standard traffic lights and the driving control is returned to or taken over by cars/vehicles.

This can be implemented by DEECo processes that run at the different cars and at the ICS and trigger the manual mode in a decentralized manner. In other words, these processes behave as *watchdog timers* at the different nodes. They force a switch to manual mode at the corresponding node, if no packets have been received for longer than $t_{1\mathrm{m}}$ time. Note that here, for ease of exposition, we neglect the time which is necessary to process data at cars, i.e., $\hat{p}_{ens} + r_{car}$, and at the ICS, i.e., $\hat{p}_{ens} + r_{ICS}$. Whereas there is only one such watchdog processes at a car, there are multiple ones at the ICS; one for each car in the system.

It should be noticed that the ICS does not need to notify cars whenever it switches to manual mode; it suffices if it stops assigning speeds to them and cars themselves will automatically switch to manual mode. In the same way, if a car first switches to manual mode, the ICS will detect this on its own without need for notification from the car.

In the worst case, since it is not known which packets may be lost, there may be up to $3 \times t_{1\mathrm{m}}$ delay for the whole system, i.e., all cars and the ICS, to switch to manual mode in a decentralized manner. This results from considering the following conditions:

(i) A packet from ICS is sent to arrive exactly $t_{1\mathrm{m}}$ time after its last packet at a given car.

(ii) This packet from the ICS is lost at the communication channel such that the car (locally) switches to manual mode.

(iii) All packets from the car to the ICS also get lost such that the ICS realizes that the car is in manual mode – and triggers itself a switch to manual mode – not until $t_{1\mathrm{m}}$ time later.

(iv) All packets from the ICS to the remaining cars get lost such that, in the worst case, all other cars switch to manual mode $t_{1\mathrm{m}}$ time after the ICS.

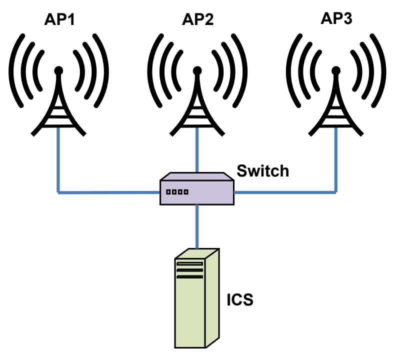


Figure 10.5: Simulated network consisting of three APs and a switch

As discussed above, this delay corresponds to 3 m in the trajectory of a car in our case study. Hence the ICS has to assign speeds to cars – in the automatic mode – such that there is sufficient distance between them taking vehicle dynamics into account (e.g., if a car suddenly breaks, it will not stop immediately due to its inertia, etc.).

Finally, only the ICS can decide to go back to the automatic mode whenever communication to all cars has normalized. To this end, the ICS needs to notify or start assigning speeds to all cars in the system. Note that the delay for switching to the automatic mode is given by $t_{1\mathrm{m}}$ since a normal communication is assumed.

CHAPTER 11

Implemented frameworks

Work on this thesis encompass implementation of several experimental frameworks and proof of the concept applications. The work on the experimental applications is covered in Chapter 13 where the code and detailed analysis of the applications is described. This chapter gives an overview of the work on the frameworks that support the experimental applications and the test-bed presented in Chapter 12. The frameworks implemented or extended in order to support work on this thesis and, in particular address Challenge C5 of this thesis, include JDEECo, CDEECo++ and PyDEECo.

The JDEECo framework was not implemented from scratch. An earlier version of the framework was extended in terms of modularity and network awareness. The work done by the author of this thesis consisted mainly from implementing bindings to ROS, OMNeT++, Stage, and interface to sensors and actuators provided by the Turtlebot and some custom hardware. These extensions were used in the artifact that is described in Chapter 12.

The CDEECo++ and PyDEECo frameworks were implemented from scratch in order to provide alternative to JDEECo framework in cases where heavy-weight Java implementation was not suitable. In particular, CDEECo++ provides ability to create hard real-time applications running on bare hardware. On contrary, the PyDEECo enables quick prototyping of new features. These can be later verified in the simulation, implemented in the JDEECo or run on the bare hardware using the CDEECo++.

11.1 JDEECo

JDEECo is mainline implementation of the DEECo. It is written in Java and serves mainly for simulation of the DEECo based sCPSs but the deployment on real devices is also supported. The latest version uses plugins to separate different features and simulation modes that were implemented in order to provide evaluation for various ideas. The source code of the JDEECo framework are located on the GitHub[1].

The JDEECo enables both classical bipartite ensemble and intelligent ensemble definitions. The first one is provided by an internal DSL implemented on top of Java classes and annotations. The intelligent ensembles are captured in an external DSL called EDL.

[1]`https://github.com/d3scomp/JDEECo`

An example of a component definition is displayed in Listing 11.1. The component definition is based on the Java class. The knowledge is defined as public fields, Lines 3-7. So called local knowledge that is not part of the component' public interface is exemplified on Line 10. A process example is displayed on Lines 17-33. Knowledge fields that are read and written by the process are defined as parameters of the process method. The knowledge fields read are accessed directly, Line 21. The output knowledge fields are accessed using a wrapper, Line 20.

```
1  @Component
2  public class Follower {
3   public String id = "Follower";
4   public final String name;
5   public Waypoint position = new Waypoint(1, 1);
6   public Waypoint destination = new Waypoint(1, 3);
7   public Waypoint leaderPosition;
8
9   @Local
10  public CurrentTimeProvider clock;
11
12  public Follower(CurrentTimeProvider clock) {
13   this.name = "Follower";
14   this.clock = clock;
15  }
16
17  @Process
18  @PeriodicScheduling(period = 2500)
19  public static void followProcess(
20     @InOut("position") ParamHolder<Waypoint> me,
21     @In("destination") Waypoint destination,
22     @In("name") String name,
23     @In("leaderPosition") Waypoint leader,
24     @In("clock") CurrentTimeProvider clock) {
25
26   if (!destination.equals(me.value) && leader != null) {
27    me.value.x += Integer.signum(leader.x - me.value.x);
28    me.value.y += Integer.signum(leader.y - me.value.y);
29   }
30
31   System.out.format("%06d:␣Follower␣%s:␣me␣=␣%s␣leader␣=␣%s%n",
32    clock.getCurrentMilliseconds(), name, me.value, leader);
33  }
34 }
```

Listing 11.1: Example of a DEECo component definition in JDEECo framework. Follower component attempts to follow the Leader at position provided by ensemble knowledge exchange.

Listing 11.2 exemplifies ensemble definition. Again a Java class is used as a wrapping package. A class defining an ensemble contains static methods used for membership check, Lines 4-15, and a knowledge exchange, Lines 17-25. Parameters of these methods are again decorated in order to provide mapping to the knowledge fields of the *coordinator* and the *member* components.

```
1  @Ensemble
2  @PeriodicScheduling(period=1000)
3  public class Convoy {
4   @Membership
5   public static boolean membership(
6      @In("member.id") String memberId,
7      @In("coord.id") String coordId,
8      @In("member.position") Waypoint fPosition,
9      @In("member.destination") Waypoint fDestination,
10     @In("coord.position") Waypoint lPosition,
11     @In("coord.path") List<Waypoint> lPath) {
12
13   return
14     !fPosition.equals(fDestination) && (Math.abs(lPosition.x - fPosition.x) +
           Math.abs(lPosition.y - fPosition.y)) <= 2 && lPath.contains(fDestination);
```

```
  }

  @KnowledgeExchange
  public static void map(
     @In("member.id") String memberId,
     @In("coord.id") String coordId,
     @Out("member.leaderPosition") ParamHolder<Waypoint> fLeaderPosition,
     @In("coord.position") Waypoint lPosition) {

   fLeaderPosition.value = lPosition;
  }
}
```

Listing 11.2: Example of DEECo ensemble definition in JDEECo framework. Convoy ensemble pass position from Leader to Follower component.

11.1.1 Modularity and Network Infrastructure

The JDEECo framework, as described earlier, used to lack support for realistic network. Work on this thesis include creation of a modular network infrastructure that enables smooth transition from a fully centralized knowledge repository with zero latency and guaranteed consistency to a distributed execution of different components and knowledge propagation using a precisely simulated packet radio. Previously structure-less initialization code was converted to a modular system with plugins implementing knowledge handling and propagation. Apart from the plugins listed in the following code examples plugins for OMNeT++ based network devices, ROS integration plugin and MATSim vehicular simulation plugin were created.

```
 1 // Create main application container
 2 SimulationTimer simulationTimer = new DiscreteEventTimer();
 3 DEECoSimulation realm = new DEECoSimulation(simulationTimer);
 4 realm.addPlugin(new SimpleBroadcastDevice());
 5 realm.addPlugin(Network.class);
 6 realm.addPlugin(DefaultKnowledgePublisher.class);
 7 realm.addPlugin(KnowledgeInsertingStrategy.class);
 8
 9 // Create first DEECo node
10 DEECoNode deeco1 = realm.createNode();
11 // Deploy components and ensembles
12 deeco1.deployComponent(new Leader(simulationTimer));
13 deeco1.deployEnsemble(ConvoyEnsemble.class);
14
15 // Create second DEECo node
16 DEECoNode deeco2 = realm.createNode();
17 // Deploy components and ensembles
18 deeco2.deployComponent(new Follower(simulationTimer));
19 deeco2.deployEnsemble(ConvoyEnsemble.class);
20
21 // Run the simulation for 20 seconds
22 realm.start(20000)
```

Listing 11.3: Example of DEECo application initialization for two nodes using the JDEECo framework in discrete event simulation mode.

Usage of a modular simulation setup code is displayed in Listing 11.3. The simulation is wrapped in a *DEECoSimulation* class instance, Line 3, and driven by a *SimulationTimer*, Line 2. In the example the timer is a discrete event timer. Definition of the simulation is followed by addition of plugins used in the simulation. The plugins define dependencies on other plugins and can use several interfaces provided by the other plugins and the JDEECo framework itself.

Plugins passed as classes are instantiated on each network node in the simulation while the plugins passed as instances are shared among the nodes in the simulation. The shared plugins cannot exist out of the simulation environment. These are used to provide simulated services.

SimpleBroadcastDevice plugin, Line 4, provides a simple packet radio device to each node. It is capable of basic simulation of range and delivery delay. The device is used by a *Network* plugin, Line 5, that provides serialization of knowledge and formation of knowledge packets. *DefaultKnowledgePublisher* plugin, Line 6, leverages ability of *network* plugin to broadcast knowledge and periodically publishes the knowledge snapshot of the local components. The knowledge replicas received are processed by a *KnowledgeInsertingStrategy* plugin and stored in a remote knowledge repository used in the ensemble formation process.

Plugin definition is followed by the definition of nodes hosting components and deployment of the ensemble definitions. In the example two nodes are defined on Lines 10-13 and 16-19. A node is created using a factory method on the simulation object. Once a node is created, an initialized instance of a component class is deployed on it followed by a class object describing a *Convoy* ensemble.

Finally the configured simulation can be started, Line 22. In a simulation mode the execution of the framework is limited by the simulation time interval.

11.1.2 Simulation and Reality

Another set of plugins was developed in order to support deployment on real devices with real packet radio. These are based on the Turtlebot platform and STM32F4 embedded board equipped with sensors and IEEE 802.15.4 radio.

The initialization displayed in Listing 11.4 is very similar to the simulation setup described in Section 11.1.1. It general it lacks the simulation container and setup only a single local node.

The setup starts with creation of a timer, Line 1. A *WallTimer* is used to drive the execution of processes and ensembles in synchronization with real wall time. Then a position plugin with a static initial position is created, Line 2. This one is used as an initial position hint to the SLAM algorithm. The robot and sensor board are exposed using ROS interfaces. *RosServices* plugin, Lines 4-6, provides interfaces to the ROS topics that can be used by the other JDEECo plugins. After the initialization of the plugins a node can be created, Lines 9-14, using all the plugins necessary to access the real hardware of the robot and the sensor board. The *BeeClick* plugin uses *RosServices* plugin in order to talk to the sensor board and also provides packet radio as communication device to the JDEECo framework. Finally the *CleanerRobot* component is deployed, Line 22, on the just created node, together with the ensembles, Lines 25-26. The *Positioning* plugin instance, Line 19, is passed to the robot component in order to let its processes sense current position of the robot and set navigation goals to its route planner.

```
1 WallTimeTimer wallTimer = new WallTimeTimer();
2 PositionPlugin positionPlugin = new PositionPlugin(12, 5);
3
4 RosServices rosServices = new RosServices(
5    System.getenv("ROS_MASTER_URI"),
6    InetAddress.getLocalHost().getHostName());
7
8 // Create main application container
```

```
DEECoNode node = new DEECoNode(ROBOT_ID, wallTimer,
    new Network(),
    new BeeClick(),
    new DefaultKnowledgePublisher(),
    new KnowledgeInsertingStrategy(),
    rosServices, positionPlugin);

final String name = "Collector" + ROBOT_ID;

// Deploy DEECo node with robot specific plugins
Positioning positioning = new Positioning();

// Deploy Collector robot component
node.deployComponent(new CleanerRobot(name, positioning, wallTimer, GARBAGE));

// Deploy ensembles
node.deployEnsemble(DestinationAdoptionEnsemble.class);
node.deployEnsemble(AdoptedDestinationRemoveEnsemble.class);

wallTimer.start();
```

Listing 11.4: Example of the DEECo application deployment on a real Turtlebot using the JDEECo and the ROS integration plugin. The *CleanerRobot* component is visiting GARBAGE locations and perform group adaptation using ensemble provided data.

11.2 CDEECo

CDEECo++ was introduced in order to enable creation of distributed hard real-time DEECo based sCPSs. It is coded in C++ and its source code is located on GitHub[2]. It is not as feature complete as JDEECo and not as flexible as PyDEECo but implements bare minimum of features required for the bipartite ensembles to work with real-time processes. Internal DSL leveraging C++ templates, classes, and structures is used to capture components and ensembles. FreeRTOS is used to schedule internal tasks used by the framework, processes, and ensembles. CDEECo++ supports components with fixed size knowledge and processes triggered by knowledge change or periodically scheduled. A process is restricted to write only a single continuous block of the knowledge due to limitations that arise from hard real-time requirements on the framework.

11.2.1 Language Mapping

Code Listing 11.5 displays example of component definition. The whole component is wrapped in a namespace, Line 1, encapsulating definition of the knowledge, processes, and component itself. The knowledge is defined as a C++ structure *Knowledge*, Lines 3-15, containing definitions of knowledge fields and their types. *Knowledge* is based on the *CDEECO:Knowledge* for the purpose of a type compatibility. Processes are defined as separate classes within the namespace of the component. An example process, Lines 18-25, inherits from *CDEECO::PeriodicTask* in order to get scheduled properly. Template arguments of the base class define input and output knowledge of the process. Once the knowledge and the processes are defined, the whole component is defined as a separate class in a namespace, Lines 28-35. Based on *CDEECO::Component* with a template argument defining

[2]https://github.com/d3scomp/cdeeco

the knowledge type a component encompass its processes as fields. Reference to a *CDEECO::broadcaster* and component id is passed to base class in order to publish knowledge of the component with its id.

```cpp
namespace PortableSensor {
  // Knowledge
  struct Knowledge: CDEECO::Knowledge {
    CDEECO::Id coordId;

    struct Position {
      float lat;
      float lon;
    } position;

    struct Value {
      float temperature;
      float humidity;
    } value;
  };

  // Sense value process
  class Sense: public CDEECO::PeriodicTask<Knowledge, Knowledge::Value> {
  public:
    Sense(auto &component);

  private:
    SHT1x sensor = SHT1x(...);
    Knowledge::Value run(const Knowledge in);
  };

  // Component definition
  class Component: public CDEECO::Component<Knowledge> {
  public:
    static const CDEECO::Type Type = 0x00000001;

    Sense sense = Sense(*this); // Process assignment

    Component(CDEECO::Broadcaster &broadcaster, const CDEECO::Id id);
  };
}
```

Listing 11.5: Portable sensor component definition using CDEECo++ framework.

Example of a ensemble definition is given in Listing 11.6. As in the case of the component the whole ensemble definition is wrapped in a namespace, Line 1. First a complex type capturing input and output knowledge type for both *member* and *coordinator* is defined on Line 2. Then an ensemble is defined using the base ensemble type. A constant for period of the ensemble formation is set, Line 6. Lines 8-10, feature two constructors that setup ensemble instance on a node hosting *coordinator* or *member*. Each constructor is responsible for one direction of the knowledge transfer. Membership condition is defined as a method taking knowledge of both *member* and *coordinator* as a parameter, Line 13. Finally, on Lines 15-17, the knowledge exchange is defined as two separate methods that perform exchange each in one direction.

```cpp
1 namespace TempExchange {
2   typedef CDEECO::Ensemble<Alarm::Knowledge, Alarm::Knowledge::SensorData,
        PortableSensor::Knowledge, PortableSensor::Knowledge::CoordId> EnsembleType;
3
4   class Ensemble: EnsembleType {
5   public:
6     static const auto PERIOD_MS = 2027;
7
8     Ensemble(CDEECO::Component<Alarm::Knowledge> &coordinator, CDEECO::
          KnowledgeLibrary<PortableSensor::Knowledge> &library);
9
```

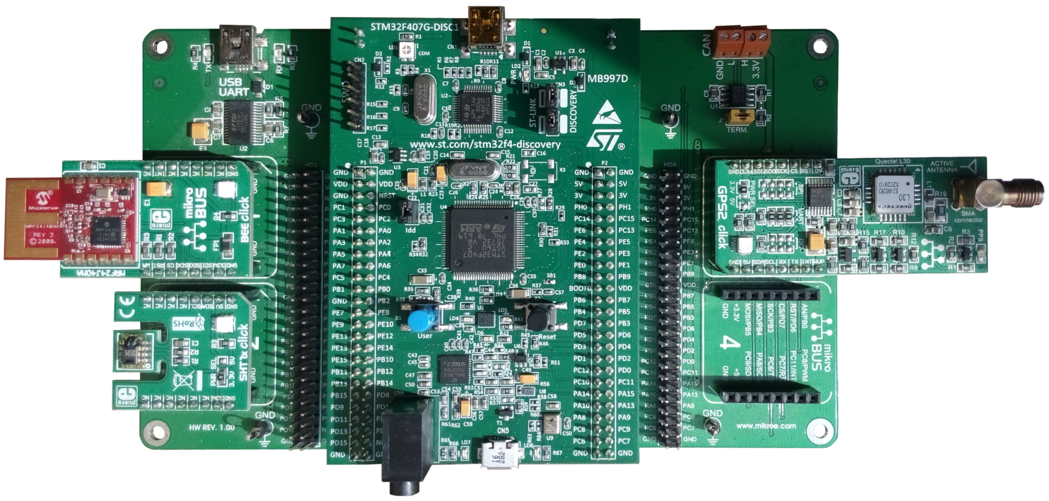


Figure 11.1: STM32F4 discovery shield with STM32F407G board, IEEE 802.15.4 radio, SHT1x temperature/humidity sensor, and GPS plugged in.

```
    Ensemble(CDEECO::Component<PortableSensor::Knowledge> &member, CDEECO::
        KnowledgeLibrary<Alarm::Knowledge> &library);

  protected:
    bool isMember(const CDEECO::Id coordId, const Alarm::Knowledge coordKnowledge,
        const CDEECO::Id memberId, const PortableSensor::Knowledge memberKnowledge)
        ;

    Alarm::Knowledge::SensorData memberToCoordMap(const Alarm::Knowledge coord,
        const CDEECO::Id memberId, const PortableSensor::Knowledge memberKnowledge)
        ;

    PortableSensor::Knowledge::CoordId coordToMemberMap(const PortableSensor::
        Knowledge member, const CDEECO::Id coordId, const Alarm::Knowledge
        coordKnowledge);

  private:
    std::default_random_engine gen;
  };
}
```

Listing 11.6: Example of CDEECo++ ensemble definition. Temp Exchange ensemble aggregates temperatures in Alarm component knowledge.

11.2.2 Platform

The CDEECo++ is supposed to run on an embedded board. The current version of the platform code uses FreeRTOS, a portable real-time operating system, for scheduling of tasks. Together with zero memory allocation during the execution of the platform and CDEECo++ framework code the process execution withing the CDEECo++ framework is hard real-time. The framework code is not specially tailored to work with particular embedded board. In theory any board supported by FreeRTOS should work. The current version of the framework includes driver for a IEEE 802.15.4 radio device. This driver has a well defined interface that can be implemented by drivers for different radio hardware, but no such drivers are provided yet. Thus the CDEECo++ is limited to the current platform by lack of radio drivers, but can be ported beyond the system formed by STM32F4 discovery shield with STM32F4 board and few peripherals displayed in Figure 11.1.

11.3 PyDEECo

Python implementation of DEECo is aiming to provide an environment for quick prototyping of new architecture constructs. Flexibility of Python language enable quick adoption of new ideas and integrated support for reflection based logging of system states and visualization provide immediate and detailed feedback on just implemented features. In order to enable quick prototyping the PyDEECo uses an internal DSL for capturing roles, components, and ensembles.

```python
class Rover(Role):
  def __init__(self):
    super().__init__()
    self.position = None
    self.goal = None
```

Listing 11.7: PyDEECo role specification.

An example of a component role definition is given in Listing 11.7. Role naturally maps to a Python class with role data records expressed as instance variables.

Listing 11.8 exemplifies a component definition. A component consist of a top level class that wraps component internals. The top level class inherits from a *Component* class that activates the included processes at runtime. Knowledge is defined as an internal class, Lines 3-6, that inherits from all roles the component is supposed to implement. The knowledge is initialized in a ___init___ method of the top level class, Lines 13-15. Process are defined as internal classes, Lines 18-20, 22-25, and 27-29. The scheduling of the process is determined using *@process* annotation. The processes access knowledge directly as instance variables using *self* reference. A *node* representing PyDEECo runtime is passed as second argument to the process in order to provide access to runtime data such as plugin interfaces and scheduler.

```python
1  class Robot(Component):
2    # Knowledge definition
3    class Knowledge(BaseKnowledge, Rover):
4      def __init__(self):
5        super().__init__()
6        self.color = None
7
8    # Component initialization
9    def __init__(self, node: Node):
10     super().__init__(node)
11
12     # Initialize knowledge
13     self.knowledge.position = node.positionProvider.get()
14     self.knowledge.goal = node.positionProvider.get()
15     self.knowledge.color = COLORS.Red
16
17   # Process definitions
18   @process(period_ms=10)
19   def update_time(self, node: Node):
20     self.knowledge.time = node.runtime.scheduler.get_time_ms()
21
22   @process(period_ms=1000)
23   def status(self, node: Node):
24     knowledge = self.knowledge
25     print(f"{knowledge.time}␣ms:␣{knowledge.id}␣at␣{knowledge.position}")
26
27   @process(period_ms=100)
28   def sense_position(self, node: Node):
29     self.knowledge.position = node.positionProvider.get()
```

Listing 11.8: PyDEECo component definition including knowledge definition, Component initialization, and definition of the component processes.

Definition of an ensemble is exemplified in Listing 11.9. Ensemble definition is wrapped in a class based on *EnsembleDefinition*. Current PyDEECo implements intelligent ensembles with experimental ensemble knowledge extension, thus the definition includes *fitness* function declaration, Lines 9-10 and ensemble knowledge definition, Lines 2-7. Membership function, Lines 12-15, is an ordinary method taking knowledge of member candidates as its arguments. The knowledge exchange is replaced by ensemble knowledge creation method, Lines 17-22.

```
1  class RobotGroup(EnsembleDefinition):
2    class RobotGroupKnowledge(BaseKnowledge, Group):
3      def __init__(self):
4        super().__init__()
5
6      def __str__(self):
7        return f"{self.__class__.__name__}␣centered␣at␣{self.center}"
8
9    def fitness(self, a: Robot.Knowledge, b: Robot.Knowledge):
10     return 1 / a.position.dist_to(b.position)
11
12   def membership(self, a: Robot, b: Robot):
13     assert type(a) == Robot.Knowledge
14     assert type(b) == Robot.Knowledge
15     return True
16
17   def knowledge(self, a: Robot.Knowledge, b: Robot.Knowledge):
18     knowledge = self.RobotGroupKnowledge()
19     knowledge.center = Position.average(a.position, b.position)
20     knowledge.members = [a, b]
21
22     return knowledge
```

Listing 11.9: PyDEECo ensemble and ensemble knowledge definition.

CHAPTER 12

Adaptation Test-bed

Evaluation of the experiments with systems in the field of the sCPS and adaptations is difficult due to the lack of common environments that enable comparison of results. These are sparse and usually do not target sCPS domain directly. Targeting goal G3 of this thesis focusing on the proper evaluation and with respect to the EBCS based sCPS it was decided to put together an easy to setup simulation environment featuring a model problem so that anyone in the scientific community and beyond can experiment with JDEECo based solution to the model problem and its adaptation. The choice of the model problem was inspired by the topic of this thesis, thus the model problem features cooperation of mobile robots that takes in account realistic network properties.

12.1 Model Problem

The reference problem provided by this test-bed is the Autonomous Cleaning Robots Coordination (ACRC) problem. In ACRC, a number of cleaning robots is deployed in in-door space consisting of corridors and multiple office rooms, see Figure 12.2.

Every robot is equipped with a 360 degrees camera which provides depth information. The robots uses the camera to observe obstacles (other robots, walls, etc.) and for navigation, by means of AMCL. Robots are equipped with a map of the place that they are supposed to clean. This map is used in the AMCL-based navigation, which works by comparing a depth scan with the map.

Robots are capable of limited communication using an IEEE 802.15.4 trans-receiver (with approx. 10 meter direct visibility range), which allows building MANETs. This means that robots can exchange data only when they are close to one another. Robots can extend the communication range by acting as proxies that rebroadcast messages further. Generally, however, no global communication can be assumed as situations when no proxy is close enough or too much interference exists are rather often.

The basic software of the robots is formed by ROS, which is the de-facto standard set of libraries and services for building open-source robotic platforms.

12.1.1 Operation and Adaptation Challenges

Each robot is initially given its own set of places it is supposed to visit and clean. In the naïve solution, which can be considered as the baseline, robots act completely independently of one another (i.e., they do not communicate nor coordinate) and visit places on their list in the given order.

Due to the complexity of the environment and the deficiencies in the ROS stack (which is considered as a black-box component that is given and one has to live with), the naïve solution gives rise to multiple problems:

- A robot has only an approximation of its position and orientation. Often, especially when other robots are present nearby, the AMCL localization fails as the depth scans (which include other robots) cannot be matched with the known map. As a result, the robot navigation becomes very imprecise and sometimes, when in dense traffic, fails completely and the robot stops.
- The navigation module in a robot sometimes fails to find a route to the destination because other robots moving by obstruct it. As the result the robot stops.
- Due to physical space constraints, robots often get to a deadlock situation – e.g., when one robot wants to enter the office and another wants to exit it. The result is again that the robots stop to avoid collision. (Note that this is a different situation to the previous point, where the failure to find a way is only transient. In this case, however, it persists until the deadlock is explicitly solved.)

Generally, each of these problems can be solved by pointing the robot to the right direction. However, it practically turns out to be quite difficult to (1) distinguish the cause of the problem, and (2) to know where to navigate the robot to recover it from the failed state.

Though these problems could be targeted by modifying ROS, our experience with extending and customizing ROS shows that a more practically viable solution is to regard ROS as a black-box and build an adaptation layer over it. As such, the robotic scenario constitutes an excellent case for adaptation. (Of course, this is by no way a criticism of ROS, which itself is the most comprehensive open-source solution for robotics. It is more an acknowledgement of the complexity inherently connected with developing systems that perform in and interact with real environments.)

To remediate the deficiencies of the baseline solution, the adaptation layer has generally free access to the robot navigation. In particular, it can obtain estimates of the position and can sense whether the robot moves. Based on this, it can:

- Manipulate the queue of locations to be visited (destinations).
- Pause the robot and command the robot to move to any place on the map.

Additionally, the adaptation layer on one robot may communicate with the adaptation layers of other robots to realize more complex adaptation strategies via cooperation.

The adaptation however comes with another set of problems, once not only recovery of the robots from failures and deadlocks is attempted, but also optimization of the overall performance of the system is important. Clearly, by reordering the locations to be visited and by transferring the responsibility of cleaning a place from one robot to another, the system can highly optimize itself. Theoretically, it can even get to a point when no collisions happen because robots exchange their destinations in such a way that they do not interfere. This is however subject to multiple problems, which can be regarded as additional adaptation challenges:

- The uncertainty in location makes planning not completely reliable.
- Communication range is limited, which means that robots in different rooms cannot communicate directly, but only through proxies (if present), which have to be located in the corridor close to the office entrances.
- The communication is subject to latencies and unreliability (due to interference) which makes it impossible for a robot to have an up-to-date knowledge of the global state of the system and disallows strong synchronization among robots.

12.1.2 Solution Comparison Dimensions

Having the adaptation logic in place, various metrics can be considered for evaluation and comparison of different adaptation strategies (solutions to ACRC). Below, metrics which were found to be useful in the experiments with ROS-controlled robots in the ACRC. Note that since ACRC contains random elements and non-determinism, the evaluation of a solution requires multiple simulation runs of ACRC and statistical evaluation (e.g. by statistical testing of sample means or quantiles).

Time to complete all the tasks (i.e., visit and clean all locations assigned to the robots at the start) can be regarded as the basic metric when it is assumed that the evaluated solution is able to make the robots complete all their tasks. Experience with ACRC shows that this is more difficult than it appears to be. For evaluating partial successes, we thus suggest the following metrics.

Number of cleaning tasks that were completed. This covers situations when time limit for completion expires or when the system itself realizes that certain locations cannot be cleaned – e.g., if a robot gets stuck in a room entrance and any attempts to move the robot out of the way fail.

Total running time till system completes or gives up. This can be used as a metric complementary to the above one, to reward solutions which possess the ability to recognize that certain problems cannot be solved. It can serve to resolve ties in case two solutions are statistically similar (e.g. a statistical test cannot reject the hypothesis of the two solutions have the same average number of cleaning tasks completed).

12.2 Test-bed

The test-bed provided as part of the artifact allows for easy experimentation with adaptation techniques and algorithms for the ACRC problem. It is built as a combination of ROS, the environment simulator and visualizer (Stage), network simulator (OMNeT++) and a component model for sCPS (DEECo). Details on the technical architecture are given in Section 12.2.3. The test-bed provides a

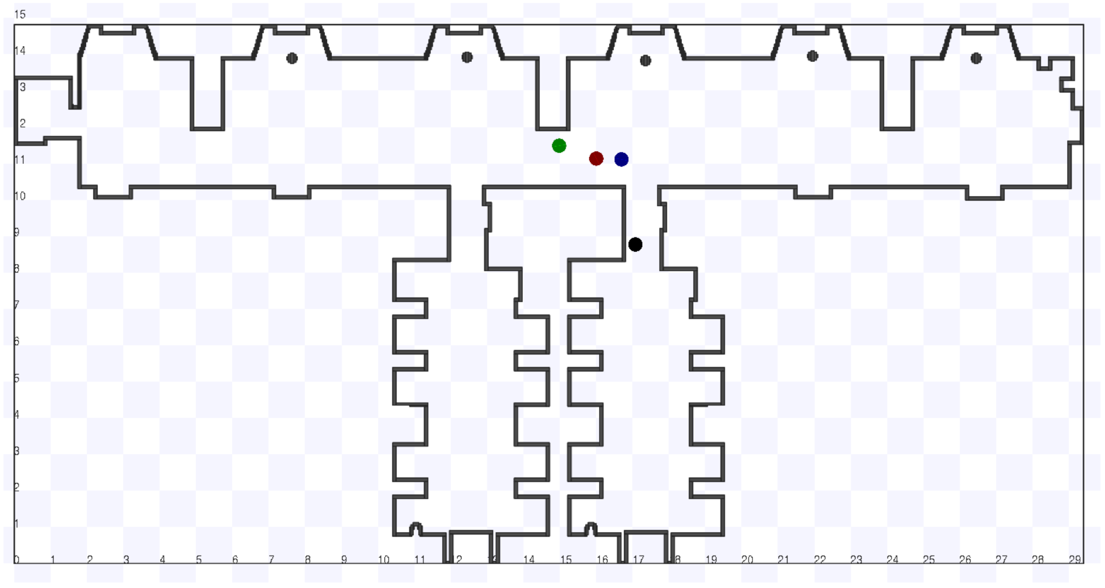


Figure 12.2: A visualization of the model problem. Green, red, blue, and black round represent robots in a office floor. Black line repents a wall and black dots are flower pots.

simulation of a swarm of Turtlebots. However, as it is developed on top of the ROS stack, it can be adapted to accommodate other ROS-controlled robots by configuring the Stage environment simulator to take into account different robots' dimension, their movement characteristics, sensors, etc.

12.2.1 User's Perspective

The test-bed models the ACRC problem via DEECo component model using bipartite ensembles described later in the text of this section). In particular, it represents each robot as an instance of *CleanerRobot* component and provides its baseline behavior (i.e. the base-level subsystem [51]) in Java. The test-bed provides a well-defined place in the component where the adaptation logic is to be plugged in (i.e. the reflective subsystem). Technically, this is done by introducing additional periodic processes to the *CollectorRobot* component, and additional ensemble specifications (e.g. see Section 12.3).

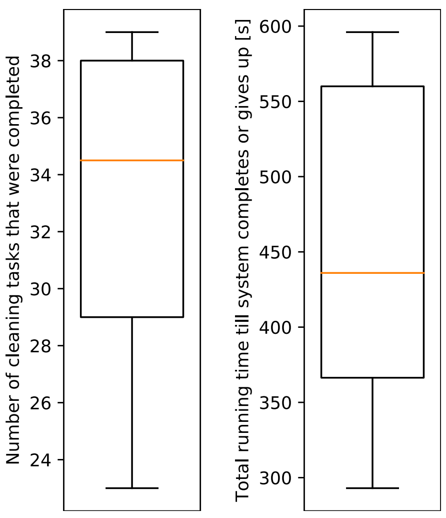


Figure 12.1: Box-plots of results from 10 experiment runs.

The actual ACRC simulation is configured by the number of robots and their initial positions. The test-bed comes with one map that comprises a corridor and two offices. Custom maps can be provided as Portable Network Graphics (PNG) files similar to the one shown in Figure 12.2.

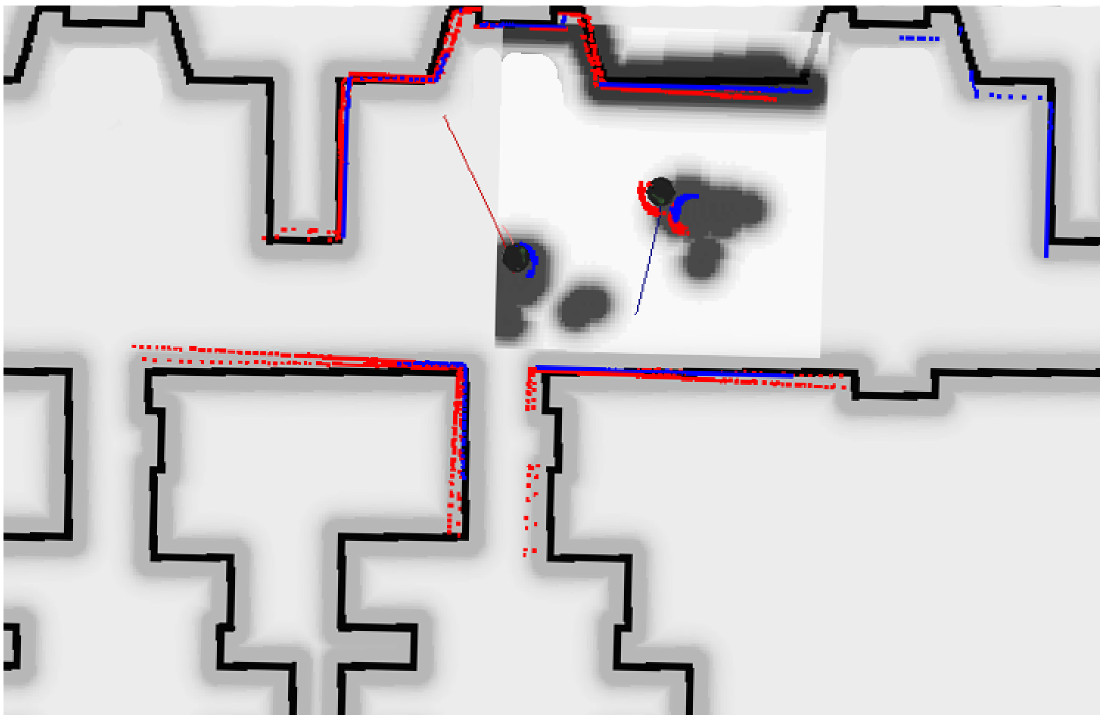


Figure 12.3: A Robots' perception of the environment. Back rounds are the robots. Red and blue dot clouds are the laser scanner reflections. Red and blue path is the path planner output. Shades of gray, forming also the black wall line, represent local and global costmap.

The primary output of the test-bed is the direct visualization of the scenario shown in Figure 12.2 (the visualizer itself comes with the Stage robot simulator – Section 12.2.3). Via it, the user can observe the movement of the robots at real time. Black lines represent walls and other obstacles impenetrable for the robots (i.e., the map provided to the test-bed). The colored dots represent the robots. Also, it is possible to observe the robots' view of their environment – Figure 12.3. The red and blue dots represent the robots' perception about the walls/obstacles. In Figure 12.3, there are two robots represented as the bigger black dots; the red dots are associated with the first robot, the blue ones with the second robot. The grey areas next to robots have a higher cost for the planning algorithm of the robot (i.e., robots try to avoid them).

Further, the test-bed comes with a script which computes statistics of the evaluation from the logs collected in multiple simulation runs. It generates box-plots of the results for the last two metrics defined in Section 12.1.2 (as in Figure 12.1).

12.2.2 Decentralized Coordination Modeling Concepts

The robots behavior is developed using DEECo and uses bipartite ensembles. In the artifact, a JDEECo is used as Java is generally easier for prototyping the components. In the real deployment usage of CDEECo++ implementation is envisioned.

Figure 12.1 shows a code skeleton of the baseline implementation of the *CleanerRobot* component in JDEECo. JDEECo constructs are further described in Chapter 11. The code example outlines public knowledge of a *CleanerRobot* on Lines 3-6. Note that knowledge marked *@Local*, Lines 8-11, is supposed only to be used internally by the processes of the *CleanerRobot* component and does not take part in any ensemble. Signature of basic *CleanerRobot* processes, defined in the baseline implementation of the *CleanerRobot* as provided by ACRC, are displayed on Lines 14-38. These are (i) setting the next destination, (ii) reading the position, (iii) reporting the status, and (iv) controlling the movement of the robot.

```
1  @Component
2  public class CleanerRobot {
3    public String id;
4    public Position destination;
5    public Position position;
6    public State state;
7
8    @Local public Long blockedCounter;
9    @Local public Long noPosChangeCounter;
10   @Local public Position oldPosition;
11   @Local public List<Position> route;
12   ...
13
14   @Process
15   @PeriodicScheduling(period = 500)
16   public static void setDestination(
17      @InOut("destination")  ParamHolder<Position> destination,...)
18   {...}
19
20   @Process
21   @PeriodicScheduling(period = 100)
22   public static void sense( @Out("position") ParamHolder<Position> position,
23      @In("positioning") Positioning positioning)
24   {...}
25
26   @Process
27   @PeriodicScheduling(period = 1000)
28   public static void reportStatus( @In("id") String id, ...)
29   {...}
30
31   @Process
32   @PeriodicScheduling(period = 2000)
33   public static void driveRobot(
34      @In("position") Position pos,
35      @In("positioning") Positioning positioning,
36      @In("destination") Position destination,
37      @InOut("curDestination") ParamHolder<Position> curDestination,...)
38   {...}
39 }
```

Listing 12.1: Model of ACRC baseline in JDEECo

Communication between components is in DEECo modeled by ensembles. Topologically, an ensemble in JDEECo is a star featuring one *coordinator* and multiple *members*.

The baseline implementation does not involve any ensembles. However, ensembles are to be exploited for decentralized coordination of adaptation across several robots. This is demonstrated in Figure 12.2, where an ensemble for location exchange is given. It is established between robots which are close to each other and both of them are stuck.

12.2.3 Technical Architecture

Figure 12.4 shows the architecture of the test-bed. Technically, it is a merger of four main existing modules. The contribution of the test-bed lies properly

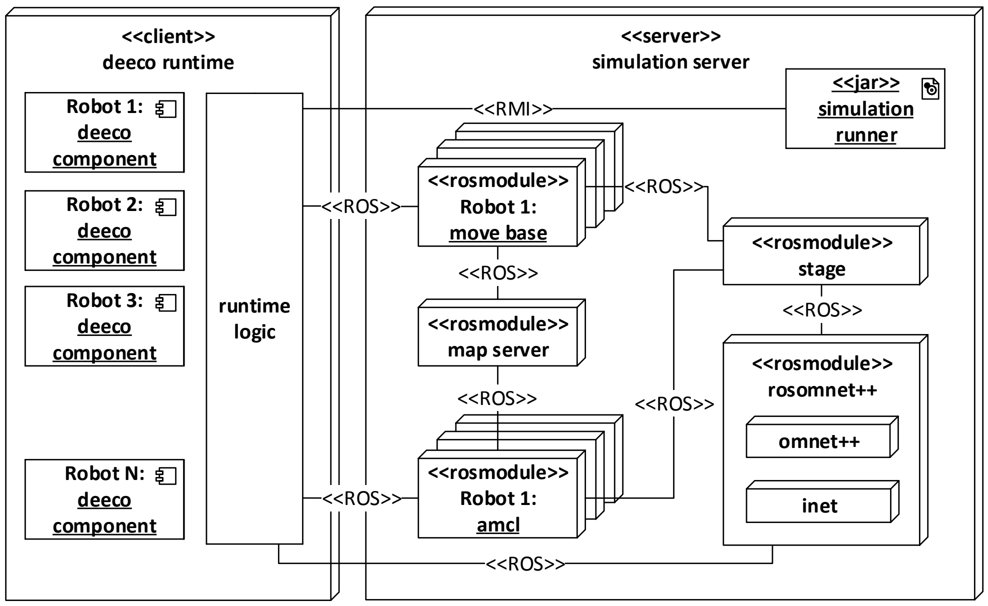


Figure 12.4: Test-bed deployment diagram

configuring them and bridging them by glue and synchronization code. The modules are:

- ROS Core – this module provides the basic software of the robot, implements the AMCL localization, navigation and low-level movement control of the robot.
- OMNeT++ – is a network simulator. It runs independently of ROS. A bridge between ROS Core publish/subscribe mechanism and OMNeT++ was implemented, which exposes the MANET transceiver as a ROS topic. This allows modules connected to ROS to communicate. OMNeT++ simulates the latency, physical range and interference of the communication based on robots' positions.
- Stage – is a robot simulator, which controls the simulation. It connects to ROS Core and simulates sensors and actuators of the robot given the simulated robot position and the map of the environment.
- JDEECo – provides the component abstraction and concepts for decentralized coordination as described in Section 12.2.2. It abstracts ROS topics on location, navigation and exposes them to DEECo components to allow for adaptation. It further exploits the ROS topic on MANET-based communication (backed by OMNeT++) to implement inter-component communication via ensembles. JDEECo again runs independently of ROS and is synchronized with it by a bridge that was developed as part of the test-bed.

12.3 Example Adaptation Logic

The model problem specification and the test-bed are complemented with an example adaptation logic as part of the ACRC model problem. It provides a comprehensive example of the modeling concepts (described in Section 12.2.2) and also serves as evaluation of the test-bed to perform simulation of physical, mobility, networking and coordination concerns.

In the example adaptation, the problems described in Section 12.1.1 are tackled in the following way:

(a) A process (on each robot) is introduced, which periodically detects the situation when a robot is stuck. This is done by checking whether the robot is moving and whether the robot has a destination set. The robot that is not moving and wants to move is considered stuck.

(b) If a robot is detected to be stuck, a random location from its queue of destinations is selected and set as its current one. This resets the navigation module in the robot and typically gets the robot to move. The outcome via the process described in (a) is monitored and repeat if no visible outcome is detected.

(c) If another robot is stuck in close proximity (up to 1.5m), an ensemble with it is established. Within the ensemble, one robot adopts the current destination of the other robot and vice-versa. This solves the (deadlock) situations when two robots meet in the office entrance and cannot proceed. The strategy (c) is illustrated in Listing 12.2. Ensemble membership is defined on Lines 7-11, and destination adoption is defined on Lines 13-24.

```
1  @Ensemble
2  @PeriodicScheduling(period = 3000)
3  public class DestinationAdoptionEnsemble {
4    double MAX_DIST_M = 3.0;
5    long BACKOFF_MS = 10000;
6
7    @Membership
8    bool membership(@In("coord.id") String coordId, ...) {
9      return coordState == Block && mbrState == Block && !coordId.equals(mbrId) &&
10         coordPos.distTo(mbrPos) < MAX_DIST_M;
11   }
12
13   @KnowledgeExchange
14   void exchng(@InOut("mbr.destination") destination, ...) {
15     if (now-lastAdoption.value)<BACKOFF_MS) {
16       return;
17     }
18
19     mbrAdoptedDestinations.value.add(coordDestination);
20     mbrDestination.value = coordDestination;
21     mbrRoute.value.add(coordDestination);
22     mbrBlockCnt.value = 0l;
23     lastAdoption.value = now;
24   }
25 }
```

Listing 12.2: Excerpt from example ACRC adaptation strategy

12.4 Lessons Learned and Limitations

The experience with development of the test-bed on top of ROS led to several observations, which are generally interesting.

Generally, a relatively big surprise was the overall immaturity of the frameworks. This most likely stems from the fact that ROS is primarily used as a platform for controlling a single robot at real-time. Though it has very flexible architecture, which allows running multiple robots within a single ROS system and allows connecting different environment simulators (e.g. Stage), the practice shows that these setups work out of the box only for trivial examples. Deploying multiple robots without careful configuration of the environment would make ROS or the Stage simulator crash. Similar story applies for OMNeT++, which is a mature and production-ready network simulator used in many applications. Nevertheless, when it comes to complex exercising of the MANET, the simulator again becomes very fragile and without careful configuration and patching, it crashes for no obvious reason. From this perspective, it seems that even without the DEECo abstraction layer, the pre-configured test-bed provided can save a couple of months of painful debugging.

Another class of problems comes from the fact that, though ROS has been used in simulations, it is not a discrete event simulator. It consists of a number of modules, which just run in wall-clock time. This means that (1) the simulation is non-deterministic, and (2) if extra care is not taken, the system crashes because the simulator, ROS, OMNeT++ and DEECo are not synchronized. This problem was solved by introducing explicit synchronization at critical places, but still one has to keep in mind that this solution does not result in fully deterministic simulations.

Surprisingly enough, experience with developing the sample adaptation logic has shown that the wall-clock timed simulation has certain advantages over a standard off-line discrete-event simulation. Since the system is live (and behaves as if the robots were moving in real time), one can watch the system as it runs, inspect the laser scans, etc. Additionally, it is possible to modify the system while it is running – e.g., a robot can be dragged by mouse to another location. While this is not important in classical batch simulations which focus on statistical comparison of different algorithms, it is very useful in debugging and especially in prototyping (which in fact is one of the primary goals of the test-bed and the reason why it was equipped with the DEECo abstractions).

12.5 Artifact Structure

The ideas described in this chapter are supported by an artifact[1]. The artifact contains the source code of the test-bed, together with installation and usage instructions. Moreover, a pre-configured virtual machine image is included in order to enable rapid hands-on experience without the hassle of installing tons of libraries. The artifact is formed by a single archive which contains all the necessary files. Instructions on how to use the artifact are located inside of the archive in *index.html*.

[1]`http://d3s.mff.cuni.cz/projects/components_and_services/deeco/files/seams-2016-artifact.zip`

CHAPTER 13

Evaluation

Evaluation of the ideas presented in this thesis, particularly in Chapters 8, 10, and 9, is covered in Sections 13.2, 13.1, and 13.3 of this chapter respectively.

Evaluation strategy is to introduce simulations of scenarios focusing on different aspects of the sCPS while trying to measure applicability of the proposed approach. Different sections describe experiments conducted with different DEECo implementations and scenarios as described in chapters describing an evaluated approach.

13.1 Safety Critical Communication

In this section, the analysis presented in Chapater 10 is validated by means of simulation. To this end, an OMNeT++ simulation [52] using *INET* hardware models was created.

An OMNeT++ simulation was setup by manually implementing DEECo components as OMNeT++ modules. In particular, OMNeT++ module for each vehicle at the intersection and for the ICS was implemented. While the ICS is stationary, vehicles and their corresponding modules in OMNeT++ move with given speeds. The modules generate network traffic that emulates the communication of vehicles entering and exiting the ICS's region of influence. This reflects the knowledge propagation for our DEECo-based ICS, from which end-to-end communication latencies are collected for a large set of simulated packet transmissions.

Our network topology consists of one ICS host connected by a full-duplex switch to three AP – see Figure 10.5. Vehicles connect dynamically to the AP adjusting

Priority levels	7
Message length	1024 bits
Packet send interval	7 ms ($\hat{p}_{pro}$ from the analysis)
ICS response delay	$\hat{p}_{pro} + \hat{p}_{ens} + r_{ICS} = 28\,\text{ms}$
A car response delay	$\hat{p}_{pro} + \hat{p}_{ens} + r_{car} = 21.18\,\text{ms}$
Bandwidth (Car to AP)	100 Mbps
Bandwidth (ICS to AP)	1 Gbps

Table 13.1: Simulation parameters

message priorities as they get closer to the intersection. The communication from the switch to the ICS host is performed under message prioritization according to the IEEE 802.1Q standard. The simulation scenario spans different numbers of vehicles (20, 50 and 70 correspondingly) exchanging packets with the ICS. Table 13.1 summarizes the most important simulation parameters considered in the evaluation.

Figure 13.1 and Table 13.2 show the results of the simulation with respect to closed-loop reaction time – i.e., the Car-ICS-Car delay – and for an increasing number of consecutive packet losses at the communication channel. In case that no packets are lost, this figure shows that our $D_{max} = 50\,\text{ms}$ – computed at the end of Section 10.4 – is safe. That is, in this case, all delay values in the system are always less than 50 ms even for 70 cars, i.e., two more cars than what it is considered and allowed by the analysis presented in the above sections.

13.1.1 Evaluation under Unreliable Communication

Simulation results for a varying number of consecutive packet losses either from the car to the ICS or from the ICS to the car are discussed here. As it can be observed in Figure 13.1, the system operates properly – i.e., the Car-ICS-Car delay is below $t_{1\text{m}} = 72\,\text{ms}$ – for up to 3 consecutive packet losses, which validates our analysis in Section 10.5. Clearly, the more packets are lost, the higher the Car-ICS-Car delay is; however, this is always less than the computed threshold $t_{1\text{m}}$ and, hence, the system can remain in automatic mode.

For case of 4 packets lost, also depicted in Figure 13.1, the Car-ICS-Car delay starts exceeding the threshold $t_{1\text{m}} = 72\,\text{ms}$ – even when considering only 20 cars at the intersection. As a result, the system cannot tolerate more than 3 consecutive packet losses without switching to manual mode. This again is in accordance with the computed upper bound on packet losses given in Equation (10.22).

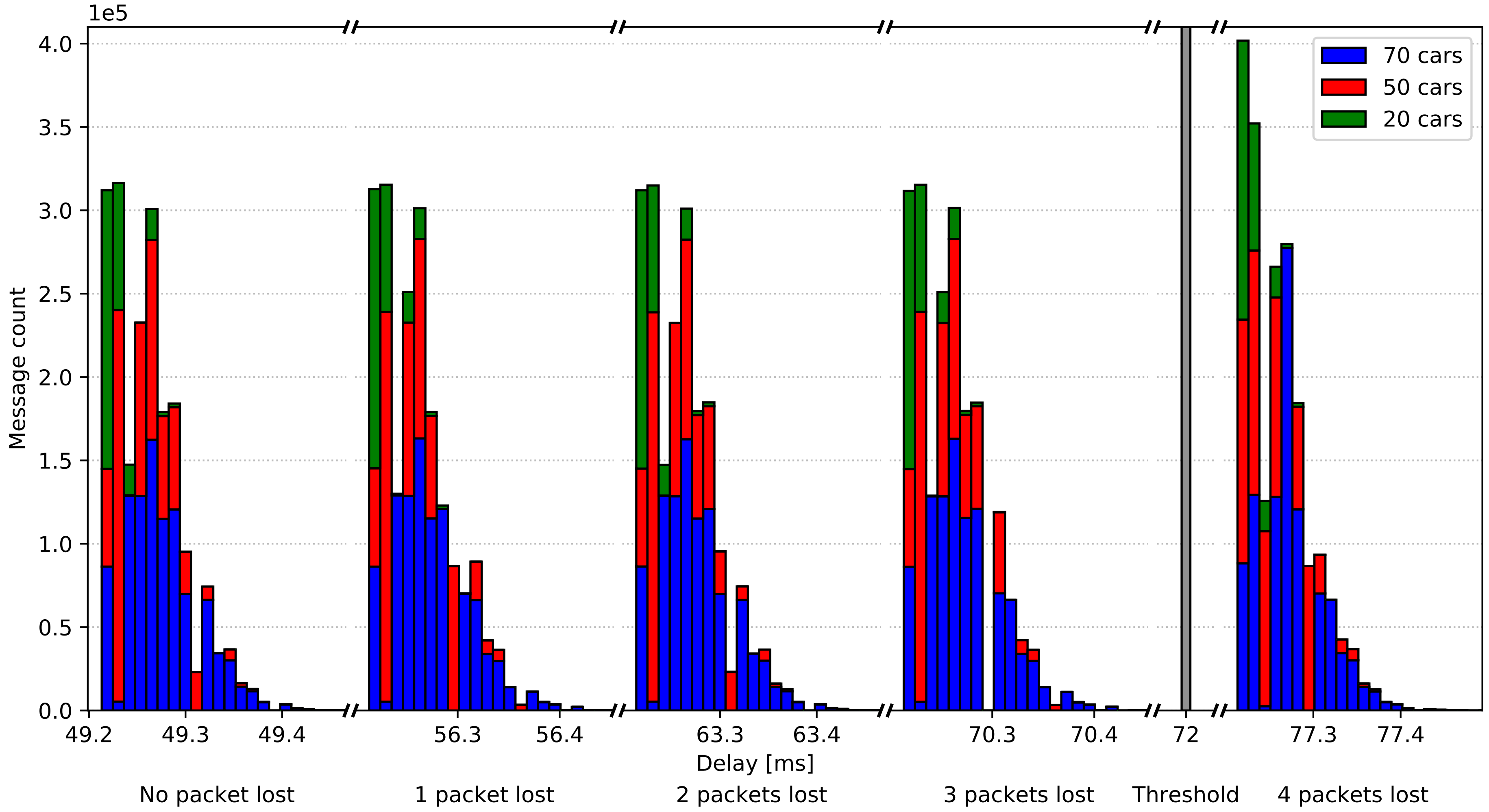


Figure 13.1: Car-ICS-Car closed-loop reaction times in milliseconds

13.1.2 Realism of the Evaluation

The presented results are based on a simulation and, thus, they may differ in reality. In particular, a number of assumptions which may not hold were made and, hence, have an impact on our evaluation. In the following text, this is discussed in more detail.

- The computed t_{1m} may not hold. This is based on the assumption that cars/vehicles can have speeds of up to 50 Km/h – see (10.14). However, in reality, it may happen that one or more cars exceed this speed limit by some amount. A solution to this is to consider a safety margin and, for example, compute a new t_{1m} for 60 Km/h instead. However, it now may happen that the ICS cannot meet this deadline anymore. To overcome this problem, the number of cars at the intersection can be restricted to a safe value. If more cars than safe enter the ICS's region of influence, it will switch to manual mode. Clearly, this higher speed limit can also be exceeded. In this case, the ICS can directly switch to manual mode.

- The computed maximum number of cars at the intersection n may also not hold. This is based on assumptions on the minimum length of cars and on the maximum possible distance between any two cars at the intersection – see (10.15). If these assumptions do not hold in practice, the maximum number of cars at the intersection may potentially increase. This has impact on the WCRT of the ICS r_{ICS} and on the worst-case communication delay from a car to the ICS c_{car}. As a result, the ICS may probably not be able to meet deadlines any longer and, hence, it will have to switch to manual mode to guarantee safety, if more cars than the maximum expected enter its region of influence.

- The WCET of processes at the cars and at the ICS may be greater than the assumed $e_i = 50\,\mu$s. This will have direct impact on the WCRT at the car r_{car} and at the ICS r_{ICS}. As a consequence, the ICS may not be able to meet deadlines anymore and, again, it will have to switch to manual mode, if a given number of cars is exceeded at the intersection.

- The bandwidths assumed for the different segments (either from the car to the AP or from the AP to the ICS) are less than those assumed in Table 13.1. This leads to increased communication delays in both directions from the car to the ICS and vice versa. The ICS may stop being able to meet deadlines and, thus, it will have to restrict the number of cars at the intersection in the automatic mode.

From the above discussion, it should be clear that discrepancies between our simulated and a real-life ICS was accounted for by taking a conservative estimate on the maximum number of cars that the ICS can simultaneously handle. If, in practice, this number is exceed, the ICS will switch to manual mode preserving safety at the cost of restricting service.

No packet lost	20 vehicles	50 vehicles	70 vehicles
Mean	49.2245	49.2507	49.2694
Std. Dev.	0.0157	0.00298	0.03723
Median	49.2182	49.2449	49.2582
Max	49.3117	49.4453	49.5121
1 packet lost			
Mean	56.2245	56.2507	56.2692
Std. Dev.	0.0155	0.00299	0.0371
Median	56.2182	56.2449	56.2582
Max	56.3117	56.4319	56.4987
2 packets lost			
Mean	63.2246	63.2508	63.2694
Std. Dev.	0.0156	0.00298	0.0372
Median	63.2182	63.2449	63.2582
Max	63.3117	63.4453	63.5121
3 packets lost			
Mean	70.2246	70.2509	70.2693
Std. Dev.	0.0156	0.00298	0.0371
Median	70.2182	70.2449	70.2582
Max	70.3117	70.4453	70.4987
4 packets lost			
Mean	77.2245	77.2507	77.2694
Std. Dev.	0.0156	0.00298	0.0372
Median	77.2182	77.2449	77.2582
Max	77.3117	77.4319	77.5388

Table 13.2: Reaction time statistics (values given in milliseconds)

13.2 Vehicle coordination in a platoon

This section provides experimental simulation of the platoon scenario that serves as an evaluation of the approach described in Chapter 8.

As a proof of concept a simulation of the platoons scenario was conducted including several experiments, allowing to assess the applicability of the method[1]. Total number of messages exchanged in the system was used as a metric for expressing communication efficiency. The simulation, conducted with use of MATSim [53], was focused on optimization of emergency vehicles' routings across realistic road network of the Prague city provided by OpenStreetMap [54]. Firefighter, police, and ambulance vehicles were considered as the emergency vehicle types. The locations of ambulance, police and firefighter bases were set according to their real locations. For simplicity, all non-road objects and several minor roads were removed from the original map which yielded a road network covering the area of approximately $100km^2$.

The simulation comprises three groups of experiments: (i) emergency call response by 3 vehicles, (ii) emergency call response by 5 vehicles, and (iii) single large platoon (convoy with the right of the way). The groups (i) and (ii) encompass experiments differentiated by number of concurrent emergency calls (1, 2, 3, 5, 10, 15, 20), while (iii) encompasses experiments with several platoon sizes (3, 5, 10, 15, 20).

As to (i) and (ii), when an emergency call is issued (e.g. a serious car crash), vehicles are dispatched to the accident site (destination). In the simulation, the emergency vehicles heading to the same destination aim at forming a platoon to make it easier to clear their path in heavy traffic by driving closely behind each other. The emergency vehicles are dispatched from the emergency service bases as close to the destination as possible. Specifically it is assumed that: in (i) one of each emergency vehicle type is sent to every destination, in (ii) two ambulance, two firefighter and one police vehicles are sent to every destination, in (iii) emergency vehicle types are not distinguished.

Once a vehicle is on its way to the accident site, it aims at following another emergency vehicle heading to the same destination. A platoon is established, when the distance between two solo vehicles heading the same destination is negligible in a street. A vehicle is allowed to join a platoon only when its prolongation of its route to the destination is minor and has the ability to increase its speed temporarily.

In order to show that the results do not depend on particular routing and destination choice, 10 different simulation runs parametric in the destination choice were executed.

13.2.1 Results

In (i) and (ii) a key result of these sets of experiments is the proof of communication complexity reduction (from quadratic to linear). Recall that this complexity metric is the number of infrastructure network messages – in our case those were the IP messages. For the group (i) the number of IP messages was measured (Figure

[1]Source code of the scenario implementation used in the experiments is available at: urlhttp://github.com/d3scomp/cbse-2015-tutorial

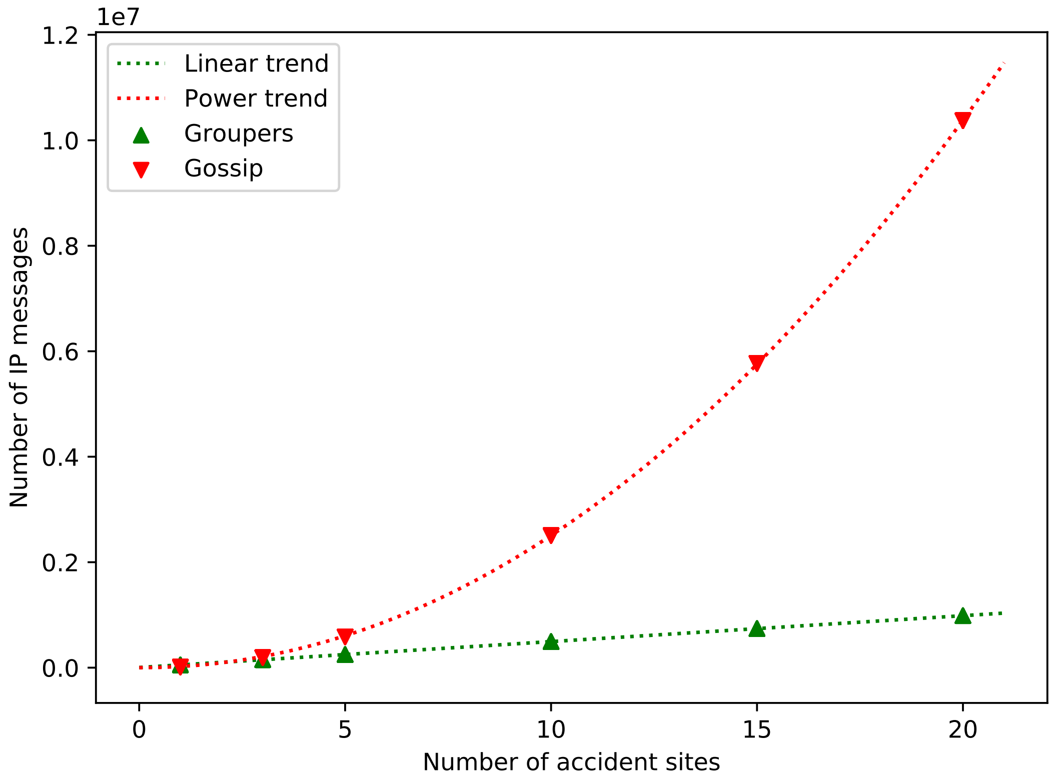


Figure 13.2: Communication complexity comparison of Gossip and groupers; experiment group (i) – 3 vehicles per accident

13.2); for the group (ii) this measurements are in (Figure 13.3). The reason for not considering MANET messages is that these are local and thus not influencing the infrastructure network load, even though small fraction of these is inherently rebroadcasted in MANET network. From these figures, it follows that when just Gossip is applied, the number of IP messages grows quadratically with the number of vehicles. This is caused by the fact that IP messages from a vehicle are sent to all of other vehicles. On the contrary, when groupers are applied the IP messages are sent only to the vehicles sharing a particular destination, the number of IP messages is linear in number of groups while assuming the size of the group is constant. Moreover, here it is also visible that the effect of improvement starts at a minimal number of destinations (such as 3 in Figure 13.2), since there is an overhead of communication among groupers.

Note that a system that enables message passing between the MANET and infrastructure networks (such as JDEECo originally) needs to be configured in such a way that messages from different communication groups do not leak from one communication group to another, otherwise this would harm the positive effect of communication groups. As an aside, in the simulation this was ensured by preventing rebroadcasting of IP messages by MANET.

Finally, domain specific knowledge can be further exploited by deciding ensemble membership conditions in groupers. Such a feature would enable distribution of knowledge only to those nodes that host components satisfying a particular ensemble membership condition. In the platoons scenario (Figure 8.2), a vehicle that is a member of a platoon is not a member of an instance of *SameDestination*

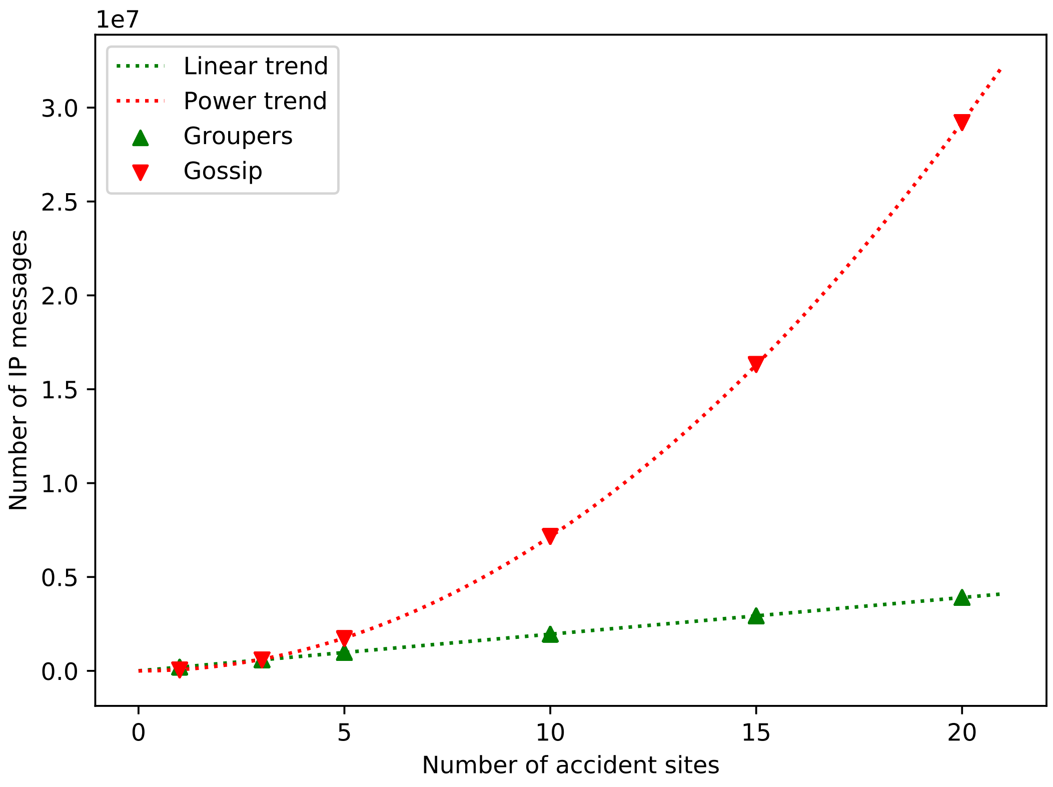


Figure 13.3: Communication complexity comparison of Gossip and groupers; experiment group (ii) – 5 vehicles per accident

any more, thus not being subject to the respective knowledge exchange, since only the "solo" vehicles and platoon leaders need to communicate via infrastructure network. Therefore, thanks to ensemble membership condition evaluation in groupers, it is possible to exclude those vehicles from communication group. The effect of this optimization would be minimal in experiment groups (i) and (ii), since the platoons considered are relatively short. In order to study this effect, the group (iii) was introduced. From Figure 13.4 it is clearly visible that introducing groupers deciding ensemble membership condition further reduces the number of IP messages for larger platoons.

13.3 Ensemble parameters and system utility

Aiming to show the effect of different network parameters on the overall systems utility, a series of experiments on the scenario described in Chapter 9 was conducted. Note that this is meant to explain the problem, not to suggest one particular setting of parameters, as the dependency between a particular parameter and network load and system utility is heavily problem specific.

The problem was modeled as a DEECo application and implemented in the JDEECo framework while the scenario specific environment and ensemble instantiation heuristic were implemented as reusable plugins to JDEECo.

All the experiment setups were based on the example scenario while maximizing distance from beacon to robot at ensemble instantiation time. In order to narrow

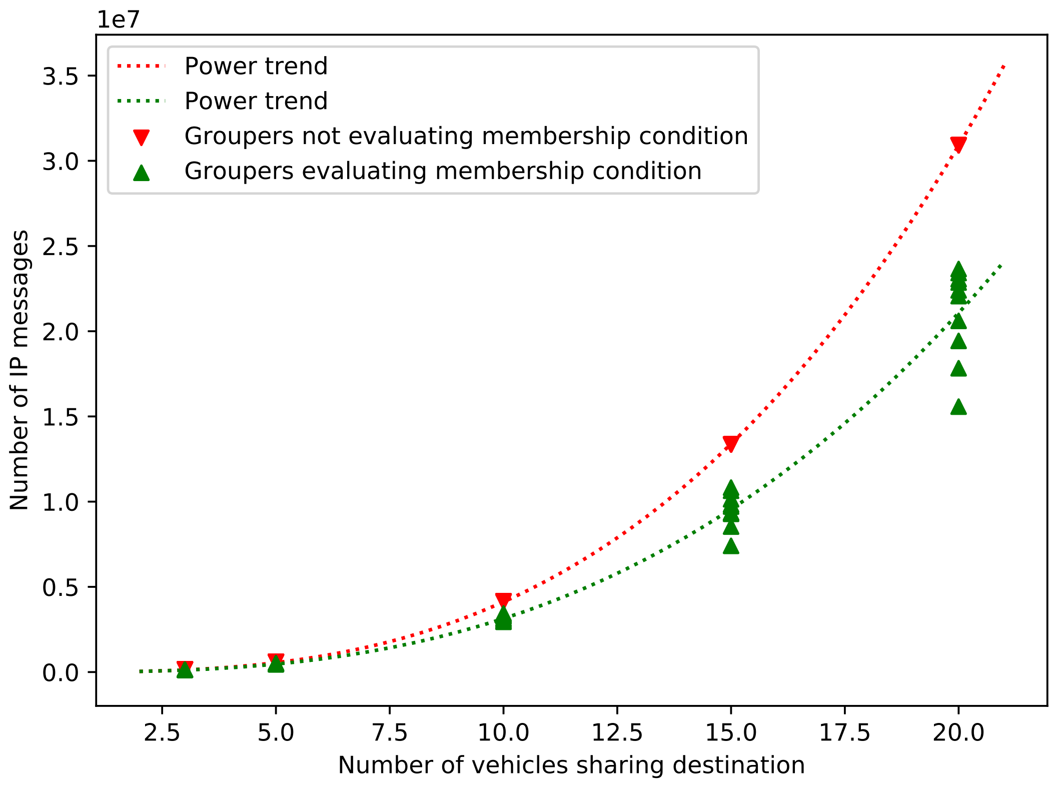


Figure 13.4: The effect of introducing groupers deciding ensemble membership condition; experiment group (iii) – single large platoon

down a huge number of candidate parameters influencing system utility, an initial round of experiments was conducted. In this run, parameters with the potential to optimize system utility were selected for further analysis. In the end, an experiment setup covering a range of values of the parameters (i), (ii), and (iii) mentioned in Section 9.4.3 was employed in the scenario simulation.

The simulation was conducted in a custom environment of the size $30x30$ meters, further determined by robot movement, beacon touching, and two radio models. The radio models include a simple one featuring delivery delays between $15ms$ and $35ms$ in the limited range of 5 meters that was initially used for the evaluation. The second model used is based on the OMNeT++ framework. It features precise radio simulation, but it slows down the simulation in an extent of several orders of magnitude. Due to the possibility of two radio models the results also serve the purpose of comparing the two models in terms of usability and precision.

13.3.1 Experiment Setup

The experiments were conducted in the following settings: The number of robots was 6, and the number of beacons was 4. This implied that the ensemble (specified in Figure 9.1 at Lines 26-48) existed in at most 3 instances, even though their average number of instances was 1.4. Basically, the experiments were executed for different values of parameters (i), (ii), and (iii) described in Section 9.4.3. In detail, a series of experiments were conducted for different values of single parameter, while all the other parameters were fixed in the series. For each parameter value setting (configuration), an experiment was executed 100 times with different random seeds of robot and beacon positions. This arrangement helped determine the influence of noise in the number of underlying messages and system utility. The whole set of experiments was executed twice. The first execution was including simple network simulation while the second one was using precise OMNeT++ network simulator.

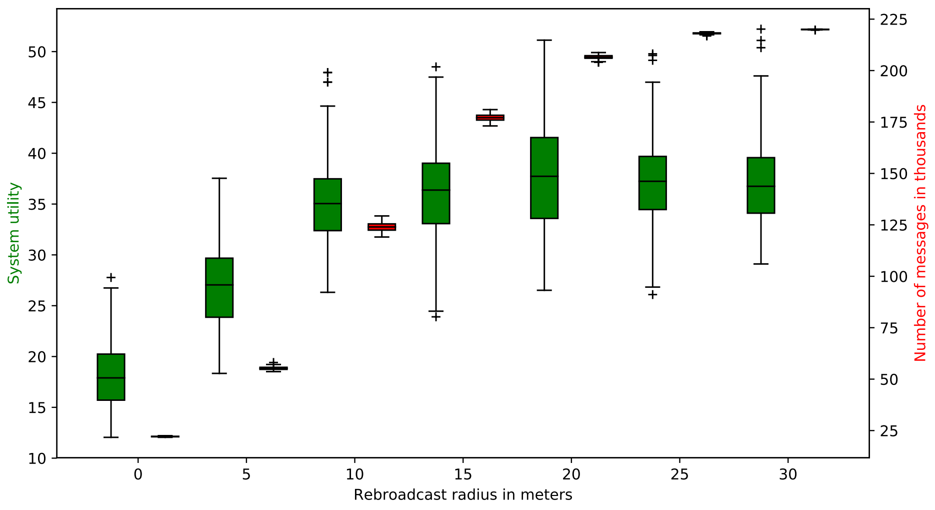
13.3.2 Simple Network Results

The actual result of the simple network based simulation runs is depicted in Figure 13.5. The effect of rebroadcast period modification is depicted in Figure 13.5b where there is a clear sweet spot between the rebroadcast period of 10 and 20 seconds. Thus setting a lower period results in producing excessive number of messages without any significant improvement in the system utility.

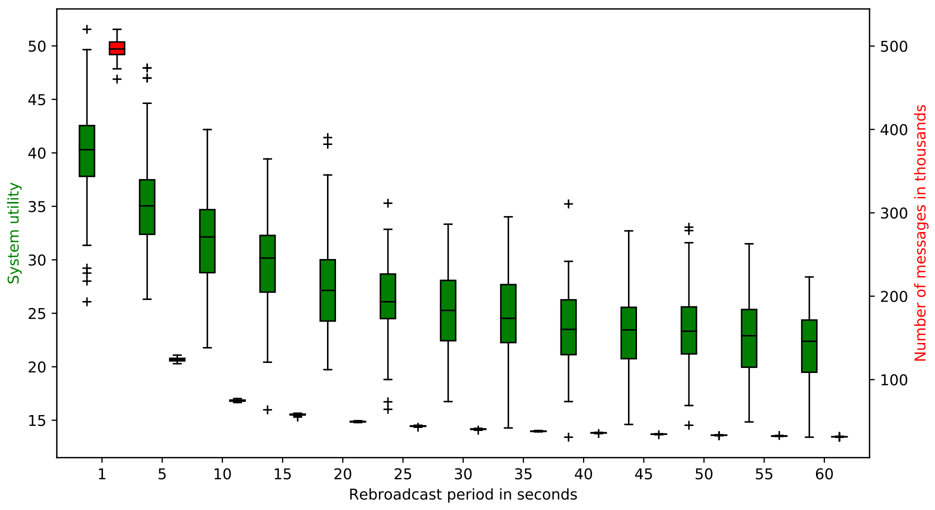
The impact of the rebroadcast radius modification, displayed in Figure 13.5a shows that the system utility is saturated at the range of 10 meters. Further extension of the range limit just implies more messages to be sent, while the system utility remains intact. Finally, in an effect of modifying max staleness is presented. Essentially, removing the messages older than the desired max staleness may cause minor reduction in number of messages sent and even slightly improve the system utility. Since the simulation was done for a simplified radio model, the actual position of the sweet spots and other interesting points in figures may not be accurate, depending upon the properties of the real network, such as latency, throughput, and congestion. Furthermore, the setting of the parameters has to respect its effect on the *smallestRadius* and *max staleness* communication constraints specified in the ensemble. Here, the relation is that the rebroadcast radius has to be greater or equal to the *smallestRadius* specification. Further, the rebroadcast period, Figure 13.5b, and max. packet age, Figure 13.5c, are positively correlated with the data staleness, which has to be kept under the specified max staleness.

Overall, the results indicate that it is possible to optimize system utility via settings of communication parameters and, at the same time, minimize the number of messages necessary to honor the communication constraints imposed in an ensemble specification. However, it is important to keep in mind that the profit for a single ensemble instance may impact the system utility influenced by another ensemble instance.

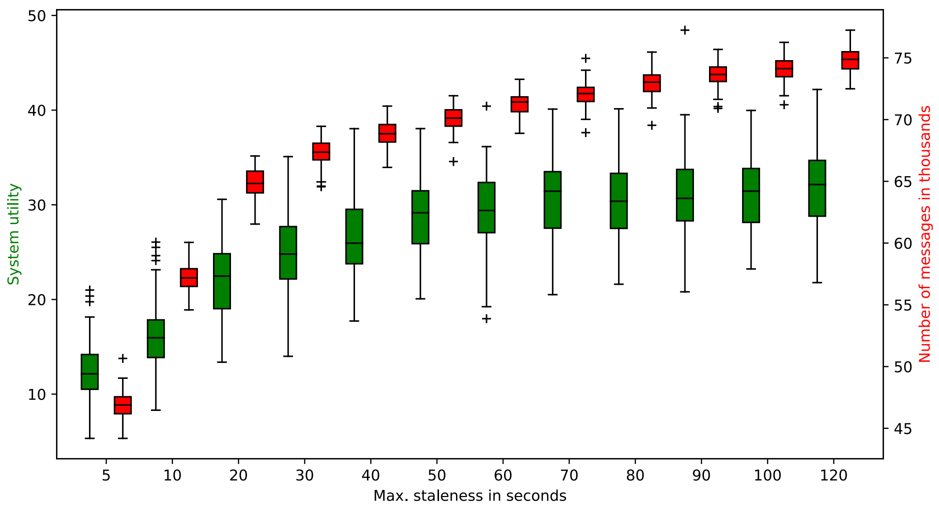
Naturally, the exact sweet spots are specific to a particular scenario, so that the results above cannot be directly applied to a different one. Nevertheless, the simulation shows that the relationships between communication parameters and system utility can be applied in a MANET to scenarios similar in communication constraints. Usage of an optimization algorithm to continuously adjust the communication parameters at runtime while trying to stay within the communication constraints is envisioned. Generally, a viable starting point for the optimization is the worst-case setting of the parameters (i.e. assuming the minimal radius and maximal staleness. From these, the optimization can try to extend the range of communication and increase the number of messages transmitted until it hits the saturation of the network.



(a) Prefer distant beacons - rebroadcast radius influence



(b) Prefer distant beacons - rebroadcast period influence



(c) Prefer distant beacons - max staleness influence

Figure 13.5: Communication parameter impact on system utility, simplified network simulation

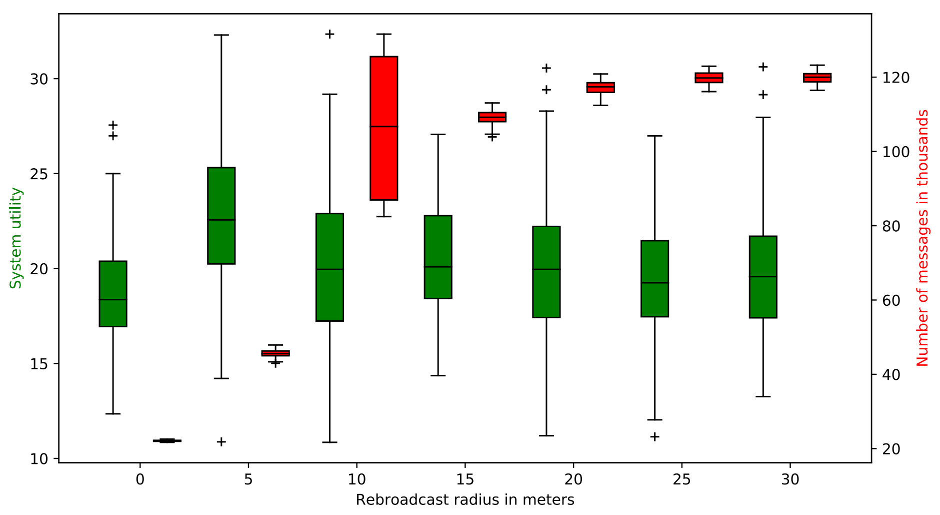
13.3.3 Precise Network Results

Originally only the simple network simulation results were used to evaluate the experiment. This was due to lack of time to conduct experiments that run for a week using an OMNeT++. Later the precise simulation results were obtained. These results serve the two purposes. First the results can be used for a more detailed evaluation of this particular scenario. Second the comparison of results show general pitfalls of the simple and fast simulation that can be used as an advice for future evaluation in the field of the sCPS.

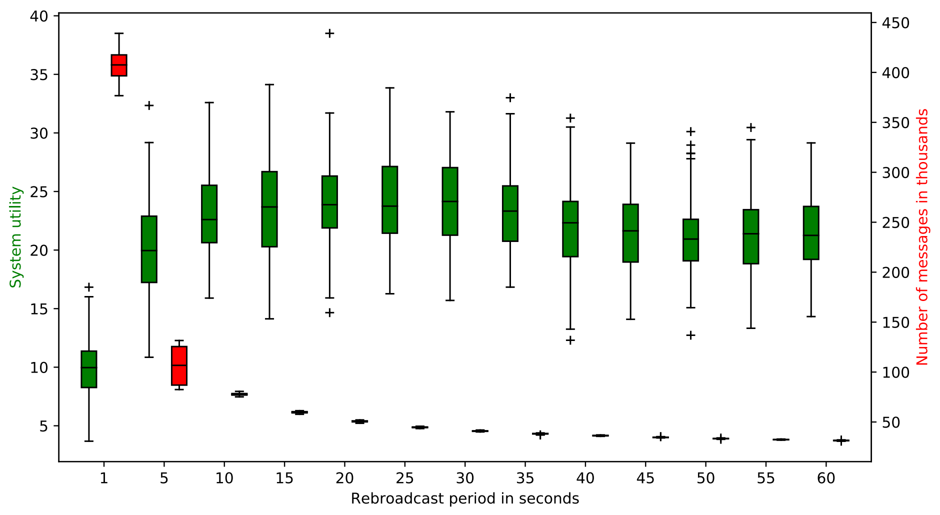
Considering the values obtained from the OMNeT++ based simulation the sweet spots are more visible and slightly shifted when compared to the simple simulation model. From the Figure 13.6a follows that the system utility is maximal at the 5 meter range. Further extension do not only bring zero benefits, as displayed in simple model results, but also overload the network and possibly slightly lower the system utility. Rebroadcast period influence displayed in 13.6b now seems to have a sweet spot somewhere around 20 seconds. This one is caused by balance between network congestion at low periods and data aging at high periods. The effect of maximum staleness enforcement displayed in 13.6c is the same as in case of a simple network model.

Concluding the differences in network model used the overall utility is reduced by $10-15\%$ when a precise network simulation is used. This is most probably due to fact that the network gets congested and packets dropped or delayed. In general the simple simulation provided results that show saturation points for different parameters. The range of parameters can be limited using the simple simulation output. The precise simulation added hard limits on network congestion caused by sending too many packets too often. These can be used to limit pentameter range in such a way that potentially dangerous system configurations that cause network overload can be avoided.

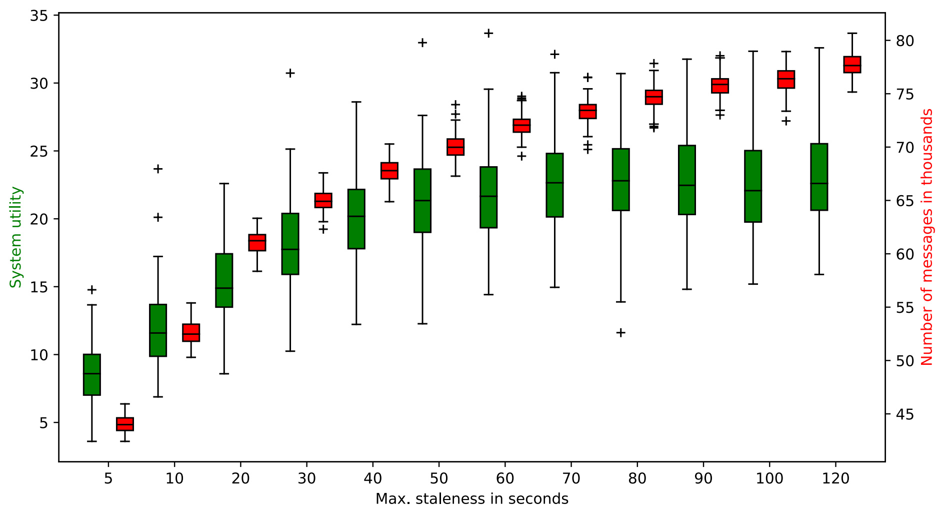
All in all the combination of a simple and OMNeT++ based simulations showed benefits and pitfalls of both. Simple simulation quickly reveals saturation points where further change in parameter value do not bring significant benefit. The precise simulation reveals the values of parameters that cause problem with network including packet drop and delays due to congestion.



(a) Prefer distant beacons - rebroadcast radius influence



(b) Prefer distant beacons - rebroadcast period influence



(c) Prefer distant beacons - max staleness influence

Figure 13.6: Communication parameter impact on system utility, precise network simulation

CHAPTER 14

Related Work

This chapter deals with works similar to this thesis in the filed of sCPS and beyond. Different approaches are grouped together in sections of this chapter based on the challenges addressed in this thesis. In particular works dealing with data propagation limits and related to Challenge C1 are described in Section 14.1; works relevant to sCPS networking and related to Challenge C2 are discussed in Section 14.2; works related to membership evaluation and related to Challenge C3 are introduced in Section 14.3; works relevant to coordination and related to Challenge C4 are discussed in Section 14.4; and relevant means of evaluation relevant to Challenge C5 are listed in Section 14.5. Some of the works seems fit for multiple challenges, but are listed only in the most relevant section.

The related work listed addresses, among others, two fundamental fields of communication and component based architectures. The approach taken in this thesis differentiate from others by putting these two together by choosing which network properties should be reflected in architecture. This way the abstractions necessary to develop complex software systems such as sCPS and IoT are used while the necessary details are provided that help to optimize network behavior and connect different layers of the system that belong together in case of the sCPS and IoT.

14.1 Knowledge propagation limits

This section deals with related work directly of indirectly addressing Challenge C1 of this thesis. The related work encompasses mostly different routing techniques, WSN, and edge cloud. All these solve the problems of (i) choosing where to propagate particular piece of information, and (ii) how to do it efficiently.

14.1.1 Wireless Sensor Networks

WSNs are systems composed of many nodes covering an area of interest and collectively sensing required values. Motivated by use cases where the nodes are powered by batteries or solar cells, the main research direction in the field aims on optimization of energy consumption. The WSN, as outlined in [55], present a network architecture that attempts to reduce energy consumption and wireless network bandwidth usage by processing the sensed values directly on the node that captured those and by employing a multihop routing scheme.

The effort to limit data that needs to be transmitted is very similar to the one in the field of EBCS based sCPS. Moreover, similarly to the sCPS, requirements on resilience and node fault tolerance motivate researchers to think of a decentralized WSN [56]. The similarity between WSN and sCPS is further manifested in [57], where the authors propose Distributed Sensing and Control (DSC) middleware that enables dynamic node addition and removal as well as resource usage management. Using the middleware it is possible to balance utility of the system and resource usage. In context of sCPS this concept is very similar to the intelligent ensemble and the *fitness* function as described in Chapter 7.

The connection between WSN and CPS is also recognized in [58] where the authors propose evolution of the WSN from pure value sensing towards bidirectional interaction between physical and virtual worlds. Further the authors propose deployment of large decentralized WSN based CPS that meet the criteria to be called sCPS.

In general WSN solve similar problems with communication as this thesis addresses in the field of the sCPS. The difference is primary focus on data in WSN while this thesis primary aims on dynamic grouping using ensembles, that also requires data propagation in order to work properly.

14.1.2 Context aware routing

Keeping the abstractions high and dealing with network traffic regardless of the packet content limits possibility to optimize routing and packet delivery. In order to further optimize it is necessary to exploit domain specific knowledge and analyze data passing through the network in order to deliver the packets efficiently and limit delivery of irrelevant data. In general this technique is called context-aware routing. An example of this is [59] where context is used to discover delivery paths based on the social behavior.

The concept of adaptive routing is similar to the communication groups presented in Chapter 8 where the context is used to route knowledge data to the destination where the similar knowledge is processed together. Further the technique similar to the context aware routing is used in Chapter 9 where packet delivery is optimized in such a way that the overall system utility is maximized using adaptive routing of knowledge packets based on their content. The work covered by this thesis goes beyond context aware routing and even drop packets based on the context. This is possible due to the architecture that combines network and component grouping, thus enabling application specific network optimization.

14.1.3 Geographic routing

In a mesh network or a MANET, building a good routing path is significant challenge. In [60] usage of a node location is proposed to optimize routing. This concept is further extended in geographic routing, [61].

Geonetwork is a type of networking system relying on a geographic rounting protocol. It is most often used in ad-hoc networks. Taking advantage of network node geographic position in addition to its address when delivering packets geographic routing based protocols enable packet delivery to network node at a particular location or to all nodes in a chosen area. There are many ad-hoc geographical routing protocols proposed different in an addressing scheme, route calculation, performance, and resiliency to network node mobility. Some of these, including Shortest path, Greedy, and Compass, are described in [39].

Geonetworking has already gained popularity in WSN and continue to advance towards vehicle to vehicle communication. Due to this it is also important for sCPS. The [62] already defines usage of geographic routing protocols in VANETs. The concept of geographic routing is related to Challenge C1 of this thesis due to ability to exploit packet context, its source, destination, or other location information in its delivery. In Chapter 9 a similar concept of packet delivery based on physical locations and distances is used. Comparing to the approach presented in this thesis the difference is in level of abstraction where geographic routing is considered a low level network while this thesis solves networking at application level.

14.1.4 Zone Routing Protocol

Zone routing protocol, as described in [63], is a hybrid wireless network protocol that combines proactive and reactive routing protocols to deliver packets.

Each node following the network protocol maintains a table of nodes in its neighborhood. This table is used by the proactive routing protocol to deliver packets to a local zone. When the proactive routing fails the sender asks nodes at the edge of its neighborhood to establish a route. If necessary, the route request can be processed recursively. Once the route is found the sender uses the discovered route to send the packet.

The combination of proactive local zone and inter-zone reactive routing makes the protocol especially suitable for vehicular systems and systems with node mobility in general. These properties make the zone routing important technique for the sCPS. In this thesis the zone based routing is related to the combination of local and global knowledge packet delivery. The local packet delivery using MANET can be seen as in-zone packet delivery while global routed packet delivery using an infrastructure based mobile data connection is responsible for packet delivery beyond zone limits. On comparison, an approach presented in this thesis attempts to spread data rather than route them to particular destination. Moreover, if directed communication is required the system already knows whenever to do local or global delivery.

14.1.5 Large scale Cyber-Physical Systems

Another field closely related to the sCPS and network awareness are the large scale CPS as outlined in [64] and [65]. Building on the concept of Machine To Machine (M2M) communication, traditional CPS, and WSN an autonomous behavior based on the in-network processing of data is envisioned. Thinking of the future in the field of M2M and CPS, massive deployment is expected with networks of millions of nodes. Current approaches that consists mainly from centralized solutions needs significant redesign in order to withstand such a widespread usage.

Concepts such as local data processing and local coordination without necessity to spread data to the whole network are related to the concepts of adaptive communication and knowledge propagation using a dynamic boundary condition. Comparing both approaches the large scale CPS lacks the complexity of sCPS in terms of dynamic grouping and coordination.

14.1.6 Edge Cloud

Edge cloud or Fog computing is an approach where computation resources are moved from data centers closer to the users in order to save latency and gain additional benefits [66], [67]. Usually the edge cloud servers are located at the edge of the network - i.e at the mobile network cells. Servers located at the edge of the network gain low latency when talking to the end user devices as well as they have access to some extra information about the end user device. This extra information enables optimizing the service based on the available bandwidth, mutual location, device type, and connection history.

This thesis stresses similar concepts of locality in data processing and adaptation of knowledge propagation based on the condition of the underlying network infrastructure. In addition to the edge cloud, the approach described in this thesis does not rely on deployed servers as it assumes computation capacities to be spread across the network nodes hosting the components.

14.2 Network architectures of the sCPS

In this section some of the architectural styles used to design sCPS, related to the topic of this thesis, in particular Challenge C2, are described. The listing is not aiming to be complete, but rather focuses on the most relevant architectures and provides brief comparison.

14.2.1 Ensemble Based Component Systems

Ensemble Based Component Systems, further described in Chapter 2, are systems similar to the DEECo, the component model used in this thesis. Another models featuring autonomous components and ensembles include SCEL [11] and Helena [16]. The Helena is covered separately in Section 14.2.3.

The SCEL is a language used to capture the whole system at a high level of abstraction. It is used to model behavior and aggregations parametric in knowledge and policies. SCEL does not enable creation of runnable applications on its own. A jRESP [17], a runtime environment and framework for SCEL based programs, is necessary to develop an executable application. Compared to DEECo used in this thesis, the SCEL based approach focuses on adaptation and analysis of the systems while it does not consider network layer in detail.

14.2.2 Kevoree

Kevoree[1] is a toolbox for creating distributed systems using the model@runtime approach. It provides a component model suitable for defining components together with their input and output connectors. The components are used as blocks to

[1]`http://kevoree.org`

build a complex system by connecting their instances to the runtime model of a system. The model is responsible for component interaction and also defines the deployment of different components.

The Kevoree aims to simplify adaptations of the system by enabling adaptation at the model level. This approach simplifies system introspection due to the runtime availability of the model. Also the adaptation execution is made easier as the system can be modified via changes made to its runtime model. The nature of Kevoree is based on interaction through an asynchronous model. This feature enables interaction of components using even a limited network connection. A communication failure does not lead to imminent system failure, instead the changes to the model are propagated once the connection is restored.

A framework that implements Kevoree functionality is available for Java and JavaScript (NodeJS and Web browsers). Components defined in different languages and using different platforms can still form a single system. A Kevoree script, or KevScript is a DSL for describing model to model transformations used to update and build the system. The KevScript sources can be used as for both deployment and adaptation description. Transformations that transform empty model to the model of a target system can be considered deployment scripts. Transformations that just alter the system can describe adaptation execution.

Communication among Kevoree components is performed using *communication channels* that can have various implementations ranging from serial links to Skype calls[2].

Comparing to the JDEECo and the approach described in this thesis to Kevoree, both provide roughly the same concepts of components, communication channels, and connectors. The difference is in DEECo based systems having ensemble based connectors that are dynamic by concept while Kevoree relies on more static and traditional connectors. Another difference is in handling of communication. Kevoree maintains communication channels that require quite reliable media and continuous communication. The DEECo based systems employ periodic knowledge publication that brings resilience and support dynamic systems.

14.2.3 Helena

Handling massively distributed systems with ELaborate ENsemble Architectures, in short Helena [16], is an EBCS based approach to handle distributed systems. Helena leverages formal definition of component interaction in ensembles based on roles and role connectors.

Helena role and ensemble models are described either using a graphical tool provided by an Eclipse SDK[3] integration plugin or using a HelenaText DSL. Both definitions are processed by a code generator that generates a Java code reflecting Helena roles as classes. The Java code can be compiled together with jHelena execution framework into an executable application. The implementation of the tool is available at GitHub[4].

[2]`https://heads-project.github.io/methodology/heads_methodology/extend_kevoree_to_support_a_new_communication_channel.html`

[3]`http://www.eclipse.org`

[4]`https://github.com/aklarl/Helena`

Formal nature of Helena model definition enables to generate Promela code instead of Java and verify the system properties using a Spin verification tool.

Helena relates to this thesis in its EBCS nature. Comparing the network awareness of Helena and DEECo based systems, Helena aims on the formal modeling and verification of the data flows while approach in this thesis aims on awareness and optimization towards realistic network properties.

14.2.4 Palladio Component Model

Palladio component model (PCM)[5] [68] is a part of a collection of software development tools called Palladio. The component model is the core of the tool-set focusing on the analysis and simulation of the systems in the early development phases. Palladio model captures components, connectors, interfaces, service behavior, middleware, deployment, and network while focusing on the separation of concerns of developers working at different levels of abstraction. Based on the model specification the tools provide analysis of performance, maintainability, and QoS.

PCM uses Eclipse Modeling Framework (EMF) as a backend for its models. Available graphical editors based on the Eclipse IDE enable interactive creation of system models. PCM together with tools from Palladio suite form a Palladio-Bench environment suitable for interactive design and analysis of various systems in early development phase.

Palladio is related to this thesis in performance modeling of the network utilization. Comparing to the DEECo based evaluation presented in this thesis, the PCM focuses on design time analysis based on the component description while this thesis measures results based on realistic simulations of real code running in a virtual environment.

14.2.5 AUTOSAR

AUTomotive Open System ARchitecture (AUTOSAR) [69] is the component model used by many brands in automotive industry. It has been created as a response to the growing complexity of the on-board electronics in modern cars. The AUTOSAR architecture is split into three layers (i) application software; (ii) Runtime Environment (RTE); and (iii) Basic Software (BSW). These are connected using well defined interfaces that enable composition of components from different vendors. The BSW encompasses the core of the real-time OS that is customized to run on a particular Electronic Control Unit (ECU). The RTE is a glue that holds different applications and BSW together. The application software is composed of components responsible for control of various hardware present in a car. These are expected to be provided by different vendors together with the hardware. The core AUTOSAR can be extended in order to upgrade its features. For instance, TIMing MOdel (TIMMO) [70] is Timing Augmented Description Language (TADL) based model used to capture timing constraints.

AUTOSAR differs from EBCS based component models in its static binding between components. Although AUTOSAR provides abstractions that could enable dynamic binding, the application components and BSW are bound together

[5]`https://sdqweb.ipd.kit.edu/wiki/Palladio_Component_Model`

at compile time using the RTE. The relevance of AUTOSAR to this thesis is based on wide adoption of AUTOSAR in the automotive industry, the main use case source of this thesis, and its real-time abilities.

14.2.6 BlueArX

BlueArX [71] is a component framework that together with AUTOSAR aims on the automotive domain. The BlueArX components are of two types (i) atomic; and (ii) structural. Atomic components are backed by a C implementation. The structural components are composed of atomic and structural components while some of the interfaces are exported. This way BlueArX expresses hierarchical components. The BlueArX supports predetermined dynamics using operational modes. Mode switching as well as component operation is real-time.

Even though the BlueArX supports some dynamics using modes it is not a true dynamic and open-ended system such as the systems described in this thesis. Its relevance is motivated, as in the case of AUTOSAR, by its target domain and real-time support.

14.2.7 ProCom

Progress Component Model (ProCom) [72] is a component model that emphasizes difference between loosely synchronized large components and fine grained tightly synchronized small components with real-time requirements. The ProCom has two layers. The top layer, called ProSys, is modeled as a set of concurrent subsystems communicating via message passing. The ProSys components can form hierarchies and consists of subsystems residing on different nodes. The low layer, called ProSave, also support component composition into hierarchies, but the low level components are passive and only react to activation by external entity. Communication between ProSave components is following pipes and filters paradigm.

ProCom is similar with models presented in this thesis in its real-time awareness, but lacks support for dynamism and open-endedness.

14.2.8 Rubus

Rubus [73] is a component model aiming on resource constrained vehicular systems with mixed real-time requirements. A Software Circuit (SWC) is a base building block defined by its behavior, interface, and internal state. It serves as a unit of encapsulation of functions. Different SWCs are connected using ports of two types used for (i) data flow; and (ii) triggering. All synchronization is restricted to the explicit synchronization visible at the model level, the behavior specific code is not allowed to provide extra synchronization.

Rubus based systems are composed of hierarchies of SWCs using assemblies and composites. These have no semantics but serve only for structural decomposition of the system. Assemblies and composites differ in deployment where assemblies cannot be split while composites can be split to reside on multiple nodes.

Rubus does not naively support dynamic and open ended systems but has native support for real-time properties and their analysis which make it relevant to this thesis. Each SWC can be accompanied by a run-time profile containing its execution time and memory consumption. Based on these Rubus supports real-time requirements on completion deadline, offset and period jitter.

14.2.9 Mechatronic UML

Mechatronic UML[6] [74] is an extension to the standard UML that allows to capture model of the complex distributed safety-critical real-time systems. It enables model driven development, verification, and code generation by employing hybrid mechatronic components [75] and real-time coordination patterns [76]. The model consists of two views (i) structural; and (ii) behavioral. Structural view (i) describes component instances connected using message exchange that can be distributed to different nodes. Behavioral view (ii) captures behavior of a single component. Both views are presented as platform independent models that can be converted to a platform dependent source code. The conversion from platform independent model to final source code is performed step by step while each step is verified in terms of real-time requirements on the system. As the system is build from verified model using properly verified steps the final code is proven to be correct by construction.

Relevance of Mechatronic UML towards this thesis is based on the fact that it captures some dynamics in the system using reconfiguration as well as it captures real-time properties. Moreover the model can capture even wireless connections and their real-time properties such as throughput and latency. Comparing to this thesis, Mechanization UML lacks support for truly dynamic and open-ended systems although some level of dynamic is supported.

14.2.10 Behavior, Interaction, Priority

Behavior, Interaction, Priority (BIP) [77] is a general framework, language, and toolset targeting systems that require rigorous design and real-time analysis. Using BIP the complex systems are composed from atomic components by coordination of their behavior. The model describes components using three layers: (i) transitions that capture component behavior using Petri nets and C functions; (ii) interactions between the transitions captured by a set of connectors; and (iii) priority rules that select interactions.

The components are obtained by superposition of the three layers. Composed components are created by composition of individual layers of the source components. BIP supports transformations, composed of elementary transformations, that work in a behavior, interaction, priority space. These can be used to convert models to timed ones or synchronous ones. Construction by composition of both components and transformations enable step by step verification of the design and ultimately provides corecness by construction for the final system.

Regarding this thesis the relevance of BIP is based on its rigorous timing specification and analysis. Compared to DEECo based systems described in this thesis the BIP based systems are static, not allowing to build open-ended systems.

[6]`http://www.mechatronicuml.org`

14.3 Membership evaluation

This section summarize work related to the Challenge C3 addressed in this thesis. The following text deals with related group formation techniques and communication organization technologies such as Mesh and Gossip. These are important due to the fact that communication technology used actually influences membership evaluation in the system. Therefore also some of the works mentioned in Section 14.1 could have been listed here.

14.3.1 Maude

Maude[7] [78] is a tool that enable capturing systems state using rewriting logic. Maude language leverage equations and rewriting rules to define rewriting systems. Once the system is defined it is possible to ask a Maude solver to reduce a particular assignment using the rules and equations present in the system.

Maude is relevant to the topic as one way of viewing dynamic groups in sCPS is based on transactions. These can be seen as actions performed on the top of an existing group, or ensemble, that modify (i) ensemble state; (ii) modify knowledge of the members; (iii) or even change component membership. A simple example of Maude usage is given in Listing 14.1. The exemplified code captures three components and an ensemble. The rewriting rules define component enter and leave action for an ensemble. Comparing to Maude, an approach used in DEECo goes beyond membership evaluation, where Maude could be applicable, but encompasses also networking, data exchange, and coordination.

```
fmod DEECo is
  sort Component .
  sort Ensemble .

  op A : -> Component .
  op B : -> Component .
  op C : -> Component .

  op E : -> Ensemble .
  op _ADD_ : Ensemble Component -> Ensemble .
  op _REM_ : Ensemble Component -> Ensemble .

  vars c : Component .
  vars e : Ensemble .
  eq e ADD c REM c = e .
endfm

reduce in DEECo : E ADD A ADD B ADD C REM C REM B.
```

Listing 14.1: A simple example of a DEECo based model captured using the Maude.

14.3.2 Coalition Formation Using Auctions

Another approach to tackle dynamic grouping, that can be used to tackle the problems of the sCPS, is to employ auction based mechanisms [79]. The auction based approaches use rules of the market to assign agents and tasks to coalitions while taking price of involved decisions into account.

[7]http://maude.cs.illinois.edu

The general idea is as follows. The agents form coalitions that compete for tasks while offering their price per task. Auction manager evaluates offers, verifies that the coalition is capable of performing the task and assigns the cheapest coalition to particular task. The process is repeated in rounds until there are no more tasks to be assigned.

Although the solution is not optimal the protocol usually offers a good approximation of the optimal solution. In a sense the auctioning system can be seen as greedy optimization algorithm where the auctioneer is doing the greedy choice of cheapest partial solution. A very nice property of the auction protocol from the point of sCPS is the fact that is is naturally distributed. Even a straightforward distributed implementation is naturally resilient to communication and agent failures. If the agent coalition fails or just falls apart due to the network failure the auction of the assigned task is repeated.

Compared to the ensemble formation in DEECo, auction based coalition formation requires handshaking among the nodes hosting the components in order to run the auctions while DEECo requires only knowledge spreading. Even so, incorporation of auction based ensemble formation into EBCS is an interesting idea that is worth further investigation.

14.3.3 Mesh networking

Mesh network [80] is a form of a MANET. Similarly to MANET the Mesh networks rely on usage of end devices as routers and require no or very little dedicated infrastructure to work. The main difference between MANET and Mesh network is envisioned usage of the technology. The MANET is a term used to address custom application specific network while the Mesh network is aiming to provide standard customer grade network infrastructure.

There are two basic types of Mesh networks. The first one, based on generally unavailable hardware, is used universities and other organizations in order to provide flexible wireless network. While the other type encompasses devices available to general public that enable building of Mesh network in general indoor or outdoor environments.

Mesh networks are relevant to membership evaluation Challenge C3 due to the natural connection of membership evaluation, data availability and connection stability. If an open-ended and dynamic system is taken into account the network infrastructure needs to be as open-ended and dynamic as the system itself. Based on current network structure the data necessary to evaluate membership may or may not be available. Moreover the knowledge of current and near future network structure can take part in group membership decision in cases where the group stability is required. These requirements make underlying protocols of mesh network relevant to membership evaluation. Compared to the overall approach presented in this thesis Mesh networking addresses just a part of the problem and do not provide architecture that combines both network and control systems.

14.3.4 Gossip Communication

Gossip [81] is a technique of information spreading used mainly on top of the MANETs. The data spreading in Gossip is inspired by human gossip when new data are spread by the nodes that are already aware of them. The benefit of the Gossip is simple implementation and by design distributed operation. Moreover the Gossip is quite reliable, robust and scalable.

There are multiple Gossip based protocols different in concrete implementation. The basic difference in Gossip implementation is determined by underling network. In the IP networks, each node usually maintains connection to N other nodes and uses such connection for gossiping. In wireless networks, or other networks that support broadcast, the gossiping takes place locally using link broadcast. With the different communication links different gossip limiting techniques are employed. The basic two limiters stop gossiping particular information once (i) it was received enough times; (ii) it originates far enough.

Relevance of the Gossip is based on the fact that knowledge spreading in systems described in this thesis is usually Gossip based. Moreover, the data spreading algorithm has huge effect on membership decision as the data availability is a precursor for membership establishment. Finally, awareness of the future connection quality is sometimes required to guarantee group membership stability.

14.3.5 Distributed Hash Tables

The communication groups as well as general knowledge spreading in DEECo systems is actually a special case of a distributed database. This makes Distributed Hash-Tables (DHT) relevant topic with respect to the ideas presented in this thesis. The main difference is that systems based on DEECo architecture utilize distributed database somewhere in its runtime, as it is an architectural concept, while the systems using DHT use it as a middleware.

DHT consists of the two fundamental parts. First one is partitioning of the data in a database, formed by key-value pairs, to network nodes using the key value [82], [83]. The second part is formation of an overlay network [84] among the network nodes that allows to determine where is the particular key-value pair stored.

The DHT based databases are resilient to node failures due to the possibility of data redundancy. Moreover some changes in the topology can be tolerated. With respect to the scalability the DHT scales well when used as standard database, but it cannot be directly used in systems where information is restricted to be stored locally by a poor long distance connection. Comparing to the approach presented in this thesis, DHT does not allow for data locality, thus DHT based solutions are vulnerable to the congestion and other connection problems that go beyond redundancy.

14.4 Component Coordination

In this section works related to the Challenge C4 addressed in this thesis are discussed. The prominent Multi agent systems, discussed in Section 14.4.1, and distributed constraint optimization problem, Section 14.4.3, are accompanied by AkKa framework, dynamic coalition formation, and RoboCup challenges described in Sections 14.4.2, 14.4.4, and 14.4.5 respectively.

14.4.1 Multi-Agent Systems

Multi-Agent Systems (MAS) are systems composed of multiple interacting intelligent autonomous agents. The idea of MAS is to think of and program individual agents rather than the system as whole. Considering target of this thesis, the MAS are relevant to the applications in fields of vehicular and robotic systems.

Classic MASs are centering on individual entities, but more complex coordination scenarios covered in [85] bring MAS quite close to the EBCS. There are various coalition formation algorithms emplyed in MAS such as [86], [87], and [88]. The difference to the topic of this thesis is lack of inclusion of network at the architecture level and explicit usage of coordination as a middleware rather than capturing cooperation using the system architecture.

14.4.2 AkKa

AkKA[8] [89] is actor based framework Scala and Java. AkKa is aiming on concurrent distributed systems while it uses messages to achieve asynchronous concurrency and thus removes necessity to use locks for synchronization. AkKa actors interact transparently when running on the same network node or using a network. Aiming on servers and data processing AkKa adopted a few patterns known from Erlang such as "Let it crash" that ensure high availability of services achieved via monitoring of running actor state and possibly restarting crashed actors. AkKa employs, so called, parental supervision where parent actors are responsible for monitoring on their child actors.

The AkKa framework aims on concurrent distributed systems that are in contrast with research presented in this thesis running on servers. Because of this the feature set of AkKa with respect to the network and its expected properties is quite different. Also, AkKa does not support open-ended systems directly.

14.4.3 Distributed Constraint Optimization Problem

Distributed Constraints Optimization Problem, or in short DCOP, is a distributed version of a standard constrain optimization problem. The constraint optimization problem is formulated for a set of variables, variable domains, and a function. The task is to pick a value for each variable from a matching domain so that the value of the function is minimal or maximal. The distributed version differs in the fact that the controlled variables are spread across a set of nodes forming a network. The computation is distributed and the nodes need to coordinate via messages in order to solve the problem. There are Distributed Constraints Optimization Problem (DCOP) solvers, such as FRODO 2.0 [90], DisChoco2 [91], and DCOPolis [92], available that help implementation of systems that need to tackle the DCOP.

An example of such problem is the distributed traffic light control. There are N traffic lights, each with a standalone controller, in a city. Each traffic light has a set of variables such as split of the green light among each controlled direction, switch frequency, or pedestrian preference. The network of such traffic lights forms a distributed system where each traffic light controller is in charge

[8] `https://github.com/akka/akka`

of a local crossing and controls the local variables. All the N controllers form a distributed system and use messages to collectively optimize the values so that a general function expressed using all the variables in the system is maximized. For instance such system can maximize overall throughput or minimize delay time for an ambulance car passing though the city.

The DCOP is relevant to the topic as the problem of forming dynamic groups or ensembles can be formulated as a DCOP instance. Providing that the free variables in the DCOP control assignment of components to the ensembles. The optimization can result in creating as many ensembles as possible or maximizing the overall utility of the system. Compared to DCOP this thesis does not seek complete optimality in ensemble formation. Instead, the ensembles formation process attempts to put together the best ensembles it can considering the current network conditions.

14.4.4 Dynamic Coalition Formation

Dynamic coalition formation is similar problem to the formation of ensembles in EBCS and sCPS. The problem was tackled in [93] where a MAS approach is used to build the system. Based on the dynamic coalition formation success and system utility the authors outline adaptation in communication of the agents that help to build better coalitions.

To distinguish from approach taken in this thesis, the authors do not take realistic network properties into account, but rather use a model of connection to the neighbor node. Also the work does not propose new architecture of the system, but rather argues necessity to optimize connections to other entities based on the coalitions that needs to be formed.

14.4.5 RoboCup Challenges

Interesting source of related approaches to the component coordination are the systems created to compete in RoboCup [94] challenges. naturally the teams are driven by success and do not spare much time to generalize the concepts, but high complexity forced some of them to take a systematic approach. Authors of [95] present their work towards formation of rescuer teams that is directly related to dynamic coordination. Another team developed a middleware and a language aiming on dynamic coalition formation using an unreliable communication in dynamic environment.

14.5 Evaluation of the sCPS Experiments

This section contains work related to the evaluation of the sCPS experiments as defined in Challenge C5 of this thesis. The following sections describe related test-beds, exemplars, and RoboCup challenges that can also be used to evaluate sCPS algorithms and architectures.

14.5.1 RoboCup

RoboCup [94] is an international scientific initiative that promotes state of the art intelligent robots. What makes RoboCup related to sCPS evaluation are four different professional and one education leagues that describe rules and real or simulated environment in which different teams can measure their abilities. The RoboCup professional leagues include RoboCupSoccer, RoboCupRescue, RoboCup@Home, and RoboCupIndustrial while the first two mentioned include simulation environment that enable evaluation comparable to the one described in this thesis.

RoboCupSoccer

As the name suggests, RoboCupSoccer, the original RoboCup league is all about football. There are four physical environments and rule sets defined that include Humanoid, Standard Platform, Middle size, and Small size. These are used to conduct matches between teams formed by real robots. The last sub-league, called Simulation, describes virtual environment in which teams of controllers of virtual robots can compete.

With respect to the sCPS evaluation challenge the simulation is the most relevant part of the RoboCupSoccer as the real robot competitions aim on different challenges than the ones addressed in this thesis. The Simulation league does provide a standard simulator and libraries used to interact with the simulator, but no advanced runtime to be used by the teams is available. Even when some of the teams publish their runtime, lack of well structured baseline implementation makes evaluation of particular technique a difficult task.

RoboCupRescue

RobopCupRescue league aims on a disaster rescue scenario where real or simulated robots are competing in their ability to rescue victims of a disaster. The league has several sub-leagues aiming on different scales of disaster ranging from detailed rescue of a limited number of persons from a hostile building to the scenario where a simulated city is on fire and virtual agents are set to extinguish the fires.

The city-wide fire extinguishing simulation is actually the most relevant to the topic of this thesis as it requires large scale coordination of agents belonging to several classes (Police, Ambulance, Fire brigade) while only limited communication is available in place. The main difference between this RoboCup league and the approach described in Chapter 12 is the scale of the simulation and readiness to the experiments. The approach mentioned in this thesis aims to provide a test-bed that is much easier to setup and start programming with.

14.5.2 Health CPS

Health CPS [96] presents a health-care CPS assisted by cloud and big data. Motivated by the recent progress in medical data collection and processing, the authors presents a three layer CPS that leverages cloud and big data to address challenges that arise from nature of the data the system needs to process. The specifics of the medical data are: (i) large scale operation with many wearable devices; (ii) rapid data generation as many devices provide continuous stream of data; (iii) various structure of the source data as different hardware provide data in different format; and (iv) deep value hidden in the data needs combination of multiple data sources and careful processing to be extracted.

As a part of the outlined architecture the authors also describe ROCHAS a test-bed aiming on the medical CPS systems. The test-bed is composed of real hardware and an environment setup. The focus is applied on the robot user interface, health-care cloud assisted system, and cloud based storage and processing. With respect to the test-bed presented in this thesis the main difference is usage of classical CPS with reliable access to centralized cloud services while approach in this thesis aims mostly on decentralized systems connected using a wireless unreliable network.

14.5.3 SCADA security test-bed

Another test-bed [97] is aiming on assessment of security of Supervisory Control And Data Acquisition (SCADA) based CPS. The authors have create their test-bed composed of real devices used in typical SCADA systems while they deployed those in a single location in order to simplify maintenance.

Aiming on CPS makes this test-bed related to the Challenge C5 of this thesis, but the target system is a static industrial SCADA that is not dynamic, open-ended or using an unreliable network.

14.5.4 Cyber-Physical Security Test-beds

Another SCADA based test-bed is described in [98]. This test-bed aims on security in CPS controlling a power grid. Authors deployed a real SCADA system together with realistic emulation and simulation of the rest of the power grid in order to enable experiments with results of attacks on the power grid.

Similarly to the test-bed described in Section 14.5.3 this test-bed is aiming on static industrial systems, but similarly to the approach presented in this thesis relies on simulation of the hardware.

14.5.5 Automated Traffic Routing Problem

The Automated Traffic Routing Problem (ATRP) [99] is model problem designed to evaluate different self-adaptation techniques. The problem encompasses cars traveling on a map using a predetermined schedule. A traffic condition on the road are subject to speed regulation, traffic jams, and closures due to construction and accidents. The traffic routing system that is being evaluated on the model problem can optimize qualities such as total travel time, worst case vehicle travel time, and pollution. The systems solving the ATRP can be adaptive, non-adaptive, centralized or distributed while also their resource consumption, resilience, and stability can be evaluated together with resulting traffic optimality.

Together with the ATRP comes the Adasim, a discrete event simulator that is capable of simulation of the ATRP and evaluate the proposed traffic routing system. Adasim knows six entity types: (i) map; (ii) vehicle; (iii) routing agent; (iv) sensors that observe the environment, (v) uncertainty filters; and (vi) privacy policies.

Simulation and evaluation of a large system from vehicular domain that could easily be addressed as a sCPS makes this work related to this thesis. The difference is in the strong accent on the adaptation and simplified communication and sensor noise compared to the work presented in Chapter 12 where precise network and environment simulation is used but the general scope of the test-bed is limited to a few robots and a single floor in a building.

14.5.6 Tele Assistance System

Tele Assistance System (TAS) [100] is an exemplar of a system providing medical services. It encompasses a scenario where patients with chronic illnesses are taken care of using a tele-assistance service encompassing sensors embedded in a wearable device and third party services providing medical analysis, drug supplies, and critical alarm response. The services have different reliability rates and prices providing space for optimization of overall cost and reliability.

The TAS is aiming on adaptation, an important technique used to manage complex and large systems. It is relevant to the test-bed presented in this thesis as it is also aiming on adaptation. The difference is in the target system where this thesis mentions adaptation of robotic CPS while the TAS is adapting service based system.

CHAPTER 15

Conclusion and Open Challenges

15.1 Summary

In this thesis, there are multiple approaches that improve EBCS behavior under realistic network conditions while aiming on applications in sCPS. First, the possible use cases in multiple domains that include robotic systems, vehicular systems and Industry 4.0 were listed. Special focus was applied on vehicular use cases that feature a common base for scenarios that arise in multiple fields. Similarities of different situations occurring in traffic to situations in swarm robotics and Industry 4.0 were discussed in order to prove importance of selected use case source. Each use case characterized by a brief description of a scenario. In total 19 scenarios were described in two categories representing systems using mostly local and global communication techniques.

The vehicular use cases were analyzed in terms of possible implementation while a possible representation of different entities as DEECo components and their relations as DEECo ensembles were given based on the experience with DEECo based system design. Based on the representation, ensemble membership conditions, a key to system specification using ensembles and components, were drafted while different extra features not yet present in the current DEECo model were pointed out. This way possible modes of DEECo system operation with (i) shared or exclusive components; (ii) exclusive component binding to a role or domain.

Use case scenarios and their drafted implementation were transformed into a set of requirements on communication while the operational challenges were discussed. Different stages of system operation were covered starting with determining of what a typical component is like, what kinds of processes is it supposed to include, and what should a knowledge of a car component include with respect to different scenarios. Once the components were identified requirements on their grouping using ensembles were listed. In particular: (i) the purpose of the ensemble in different scenarios was identified to be one of data exchange, coordination, and shared state maintenance; (ii) requirements on network throughput, latency and reliability with respect to centralized and distributed ensemble formation were discussed while summarizing the most suitable ensemble formation technique per scenario; (iii) usage of mobile broadband and short range packet radio as well as the requirements on real-time communication were described per scenario. The specifics of the network requirements on consistency, latency, and throughput in currently supported ensemble formation techniques were discussed in dedicated chapters focusing on bipartite ensembles and intelligent ensembles respectively.

Selected approaches were evaluated by means of draft implementation and simulation of system operation. The evaluation was based on the work published in reviewed conference proceedings and a journal. Frameworks implementing DEECo developed or heavily modified as part of an effort to write this thesis such as JDEECo, CDEECo++, and PyDEECo were used to run the draft implementation. The results presented were captured using different simulators including OMNeT++, SUMO, MATSim, and Stage. Some of the simulations were designed to be reusable and one of them was published as an easy to run artifact aiming to help other scientists in the community to verify their ideas.

15.2 Open Issues

Work on this thesis proved applicability of the EBCS based techniques in the field of sCPS by means of small to mid scale simulation. What is still yet to be analyzed is the applicability of the approach and proposed techniques in the large scale simulation or real life deployment. Also a closer to reality implementation of the business logic could be employed in the simulation in order to prove real-life usability of the chosen approaches.

One of the possible future work directions is to extend Turtlebot based testbed presented in Chapter 12. An output of simulation that encompasses at last several tenths of robots would shed more light on the scalability of the ensemble based approach. Moreover the current state of the JDEECo framework is fit for deployment on real Turtlebots a comparison of simulation results with data obtained from real robots would further support correctness of the approach.

Another direction of the future work is towards the vehicular systems. Regarding the use case scenarios described in Chapter 4 just ICS and platoon were covered in evaluation. Using the SUMO, OMNeT++, and JDEECo or PyDEECo it should be possible to conduct realistic network aware simulation that can include almost realist business logic solving different car to car interactions in the traffic. These would also further supported usage of EBCS in vehicular sCPS.

Ultimately some of the long range communication use cases listed in Chapter 4 Section 4.5 could be brought to reality in a form of a mobile application. Relative independence of the remaining systems of a car in present in the scenarios like Car sharing (VS14) or parking place registry (VS12) can be leveraged to easily bring these to experimental deployment in a real traffic.

Bibliography

[1] M. Kit, F. Plasil, V. Matena, T. Bures, and O. Kovac, “Employing domain knowledge for optimizing component communication,” in *Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering*, ser. CBSE ’15, Montréal, QC, Canada: ACM, 2015, pp. 59–64, ISBN: 978-1-4503-3471-6. DOI: 10.1145/2737166.2737172. [Online]. Available: http://doi.acm.org/10.1145/2737166.2737172.

[2] V. Matena, A. Masrur, and T. Bures, “An ensemble-based approach for scalable QoS in highly dynamic CPS,” in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Aug. 2017, pp. 234–238, ISBN: 978-1-5386-2141-7. DOI: 10.1109/SEAA.2017.62. [Online]. Available: https://dx.doi.org/10.1109/SEAA.2017.62.

[3] T. Bures, V. Matena, R. Mirandola, L. Pagliari, and C. Trubiani, “Performance modelling of smart cyber-physical systems,” in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE ’18, Berlin, Germany: ACM, 2018, pp. 37–40, ISBN: 978-1-4503-5629-9. DOI: 10.1145/3185768.3186306. [Online]. Available: http://doi.acm.org/10.1145/3185768.3186306.

[4] O. Štumpf, T. Bureš, and V. Matěna, “Security and trust in data sharing smart cyber-physical systems,” in *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ser. ECSAW ’15, Dubrovnik, Cavtat, Croatia: ACM, 2015, 18:1–18:4, ISBN: 978-1-4503-3393-1. DOI: 10.1145/2797433.2797451. [Online]. Available: http://doi.acm.org/10.1145/2797433.2797451.

[5] A. Masrur, M. Kit, V. Matěna, T. Bureš, and W. Hardt, “Component-based design of cyber-physical applications with safety-critical requirements,” *Microprocessors and Microsystems*, vol. 42, pp. 70–86, 2016, ISSN: 0141-9331. DOI: 10.1016/j.micpro.2016.01.007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933116000107,

Impact Factor: 1.025, CiteScore: 1.11, SCImago Journal Rank: 0.238

Statistics captured on 05/09/2018 from https://www.journals.elsevier.com/microprocessors-and-microsystems.

[6] V. Matena, T. Bures, I. Gerostathopoulos, and P. Hnetynka, “Model problem and testbed for experiments with adaptation in smart cyber-physical systems,” in *Proceedings of the 11th International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS ’16, Austin, Texas: ACM, 2016, pp. 82–88, ISBN: 978-1-4503-4187-5. DOI:

10.1145/2897053.2897065. [Online]. Available: http://doi.acm.org/10.1145/2897053.2897065.

[7] T. Bures, P. Hnetynka, F. Krijt, V. Matena, and F. Plasil, “Smart coordination of autonomic component ensembles in the context of ad-hoc communication,” in *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques: 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10–14, 2016, Proceedings, Part I*, T. Margaria and B. Steffen, Eds. Cham: Springer International Publishing, 2016, pp. 642–656, ISBN: 978-3-319-47166-2. DOI: 10.1007/978-3-319-47166-2_45. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-47166-2_45.

[8] J. Rodriguez, *Fundamentals of 5G Mobile Networks*, 1st. Wiley Publishing, 2015, ISBN: 978-1-118-86752-5. [Online]. Available: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118867521.html.

[9] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, “A survey of mobile phone sensing,” *IEEE Communications Magazine*, vol. 48, no. 9, pp. 140–150, Sep. 2010, ISSN: 0163-6804. DOI: 10.1109/MCOM.2010.5560598. [Online]. Available: https://doi.org/10.1109/MCOM.2010.5560598.

[10] T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil, “DEECO: An ensemble-based component system,” in *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, ser. CBSE ’13, Vancouver, British Columbia, Canada: ACM, 2013, pp. 81–90, ISBN: 978-1-4503-2122-8. DOI: 10.1145/2465449.2465462. [Online]. Available: http://doi.acm.org/10.1145/2465449.2465462.

[11] M. Wirsing, M. Hölzl, M. Tribastone, and F. Zambonelli, “ASCENS: Engineering autonomic service-component ensembles,” in *Formal Methods for Components and Objects: 10th International Symposium, FMCO 2011, Turin, Italy, October 3-5, 2011, Revised Selected Papers*, B. Beckert, F. Damiani, F. S. de Boer, and M. M. Bonsangue, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–24, ISBN: 978-3-642-35887-6. DOI: 10.1007/978-3-642-35887-6_1. [Online]. Available: https://doi.org/10.1007/978-3-642-35887-6_1.

[12] K. Galvin, M. Hieb, and C. Blais, *Coalition battle management language (c-bml) study group report*, 2005. [Online]. Available: https://calhoun.nps.edu/handle/10945/31179.

[13] “IEEE standard for wireless access in vehicular environments (WAVE) – networking services,” *IEEE Stdandard 1609.3*, 2016. [Online]. Available: http://standards.ieee.org/findstds/standard/1609.3-2016.html.

[14] F. Cunha, L. Villas, A. Boukerche, G. Maia, A. Viana, R. A. F. Mini, and A. A. F. Loureiro, “Data communication in VANETs: Protocols, applications and challenges,” *Ad Hoc Networks*, vol. 44, pp. 90–103, 2016, ISSN: 1570-8705. DOI: 10.1016/j.adhoc.2016.02.017. [Online].

Available: http://www.sciencedirect.com/science/article/pii/S1570870516300580.

[15] R. Tachet, P. Santi, S. Sobolevsky, L. I. Reyes-Castro, E. Frazzoli, D. Helbing, and C. Ratti, “Revisiting street intersections using slot-based systems,” *PloS one*, vol. 11, no. 3, e0149607, 2016. [Online]. Available: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0149607.

[16] R. Hennicker and A. Klarl, “Foundations for ensemble modeling – the helena approach,” in *Specification, Algebra, and Software: Essays Dedicated to Kokichi Futatsugi*, S. Iida, J. Meseguer, and K. Ogata, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 359–381, ISBN: 978-3-642-54624-2. DOI: 10.1007/978-3-642-54624-2_18. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-54624-2_18.

[17] (2018). jRESP – runtime environment for SCEL programs, ASCENS, [Online]. Available: http://www.ascens-ist.eu/jresp.html (visited on 06/26/2018).

[18] “IEEE standard for low-rate wireless networks,” *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pp. 1–709, Apr. 2016. DOI: 10.1109/IEEESTD.2016.7460875. [Online]. Available: https://dx.doi.org/10.1109/IEEESTD.2016.7460875.

[19] “IEEE standard for low-rate wireless networks–amendment 4: Higher rate (2 Mb/s) physical (phy) layer,” *IEEE Std 802.15.4t-2017 (Amendment to IEEE Std 802.15.4-2015 as amended by IEEE Std 802.15.4n-2016, IEEE Std 802.15.4q-2016, and IEEE Std 802.15.4u-2016*, pp. 1–25, Apr. 2017. DOI: 10.1109/IEEESTD.2017.7900315. [Online]. Available: https://dx.doi.org/10.1109/IEEESTD.2017.7900315.

[20] T. Sun, N.-C. Liang, L.-J. Chen, P.-C. Chen, and M. Gerla, “Evaluating mobility support in ZigBee networks,” in *Embedded and Ubiquitous Computing: International Conference, EUC 2007, Taipei, Taiwan, December 17-20, 2007. Proceedings*, T.-W. Kuo, E. Sha, M. Guo, L. T. Yang, and Z. Shao, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 87–100, ISBN: 978-3-540-77092-3. DOI: 10.1007/978-3-540-77092-3_9. [Online]. Available: https://doi.org/10.1007/978-3-540-77092-3_9.

[21] S. Gräfling, P. Mähönen, and J. Riihijärvi, “Performance evaluation of IEEE 1609 wave and IEEE 802.11p for vehicular communications,” in *2010 Second International Conference on Ubiquitous and Future Networks (ICUFN)*, Jun. 2010, pp. 344–348. DOI: 10.1109/ICUFN.2010.5547184. [Online]. Available: http://dx.doi.org/10.1109/ICUFN.2010.5547184.

[22] J. Hoebeke, I. Moerman, B. Dhoedt, and P. Demeester, “An overview of mobile ad hoc networks: Applications and challenges,” *Journal of the the Communications Network*, vol. 3, no. 3, pp. 60–66, 2004, ISSN: 1477-4739. [Online]. Available: http://cwi.unik.no/images/Manet_Overview.pdf.

[23] J. Loo, J. L. Mauri, and J. H. Ortiz, *Mobile Ad Hoc Networks: Current Status and Future Trends*, 1st. Boca Raton, FL, USA: CRC Press, Inc., 2011, ISBN: 9781439856505. [Online]. Available: https://www.crcpress.com/Mobile-Ad-Hoc-Networks-Current-Status-and-Future-Trends/Loo-Lloret-Mauri-Ortiz/p/book/9781439856512.

[24] A. Autolitano, C. Campolo, A. Molinaro, R. M. Scopigno, and A. Vesco, "An insight into decentralized congestion control techniques for VANETs from ETSI TS 102 687 V1. 1.1," in *Wireless Days (WD), 2013 IFIP*, IEEE, Nov. 2013, pp. 1–6. DOI: 10.1109/WD.2013.6686471. [Online]. Available: http://ieeexplore.ieee.org/document/6686471.

[25] T. ETSI, "Intelligent transport systems (ITS); decentralized congestion control mechanisms for intelligent transport systems operating in the 5 ghz range; access layer part," *ETSI TS*, vol. 102, no. 687, p. V1, 2011.

[26] F. Rezaei, M. Hempel, and H. Sharif, "LTE PHY performance analysis under 3GPP standards parameters," in *Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2011 IEEE 16th International Workshop on*, IEEE, Jun. 2011, pp. 102–106. DOI: 10.1109/CAMAD.2011.5941095.

[27] F. Krijt, Z. Jiracek, T. Bures, P. Hnetynka, and I. Gerostathopoulos, "Intelligent ensembles: A declarative group description language and java framework," in *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '17, Buenos Aires, Argentina: IEEE Press, 2017, pp. 116–122, ISBN: 978-1-5386-1550-8. DOI: 10.1109/SEAMS.2017.17. [Online]. Available: https://doi.org/10.1109/SEAMS.2017.17.

[28] T. Bures, F. Krijt, F. Plasil, P. Hnetynka, and Z. Jiracek, "Towards intelligent ensembles," in *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ser. ECSAW '15, Dubrovnik, Cavtat, Croatia: ACM, 2015, 17:1–17:4, ISBN: 978-1-4503-3393-1. DOI: 10.1145/2797433.2797450. [Online]. Available: http://doi.acm.org/10.1145/2797433.2797450.

[29] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, Dec. 2012. [Online]. Available: http://sumo.dlr.de/pdf/sysmea_v5_n34_2012_4.pdf.

[30] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, Jan. 2011. DOI: 10.1109/TMC.2010.133. [Online]. Available: https://doi.org/10.1109/TMC.2010.133.

[31] F. Hagenauer, F. Dressler, and C. Sommer, “Poster: A simulator for heterogeneous vehicular networks,” in *2014 IEEE Vehicular Networking Conference (VNC)*, Dec. 2014, pp. 185–186. DOI: 10.1109/VNC.2014.7013339. [Online]. Available: https://doi.org/10.1109/VNC.2014.7013339.

[32] M. Wirsing, M. Hölzl, N. Koch, and P. Mayer, *Software Engineering for Collective Autonomic Systems: The ASCENS Approach*, ser. Lecture Notes in Computer Science. Springer, 2015, vol. 8998, ISBN: 978-3-319-16309-3. DOI: 10.1007/978-3-319-16310-9. [Online]. Available: http://dblp.uni-trier.de/db/series/lncs/lncs8998.html.

[33] M. Hausknecht, T. C. Au, and P. Stone, “Autonomous intersection management: Multi-intersection optimization,” in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011, pp. 4581–4586. DOI: 10.1109/IROS.2011.6094668. [Online]. Available: https://dx.doi.org/10.1109/IROS.2011.6094668.

[34] A. L. O. Paraense, K. Raizer, and R. R. Gudwin, “A machine consciousness approach to urban traffic control,” *Biologically Inspired Cognitive Architectures*, vol. 15, pp. 61–73, 2016, ISSN: 2212-683X. DOI: 10.1016/j.bica.2015.10.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2212683X15000614.

[35] T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil, “Gossiping components for cyber-physical systems,” in *Software Architecture: 8th European Conference, ECSA 2014, Vienna, Austria, August 25-29, 2014. Proceedings*, P. Avgeriou and U. Zdun, Eds., Cham: Springer International Publishing, 2014, pp. 250–266, ISBN: 978-3-319-09970-5. DOI: 10.1007/978-3-319-09970-5_23. [Online]. Available: https://doi.org/10.1007/978-3-319-09970-5_23.

[36] (2018). jDEECo: Java framework implementing the DEECo component system, Charles University, Department of Distributed and Dependable Systems, [Online]. Available: https://github.com/d3scomp/JDEECo (visited on 06/26/2018).

[37] S. Voulgaris, M. Jelasity, and M. van Steen, “A robust and scalable peer-to-peer gossiping protocol,” in *Proceedings of the Second International Conference on Agents and Peer-to-Peer Computing*, ser. AP2PC’03, Melbourne, Australia: Springer-Verlag, 2004, pp. 47–58, ISBN: 978-3-540-24053-2. DOI: 10.1007/978-3-540-25840-7_6. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-25840-7_6.

[38] R. Friedman, D. Gavidia, L. Rodrigues, A. C. Viana, and S. Voulgaris, “Gossiping on manets: The beauty and the beast,” *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 5, pp. 67–74, Oct. 2007, ISSN: 0163-5980. DOI: 10.1145/1317379.1317390. [Online]. Available: http://doi.acm.org/10.1145/1317379.1317390.

[39] I. Stojmenovic, “Position-based routing in ad hoc networks,” *IEEE Communications Magazine*, vol. 40, no. 7, pp. 128–134, Jul. 2002, ISSN: 0163-6804. DOI: 10.1109/MCOM.2002.1018018. [Online]. Available: https://doi.org/10.1109/MCOM.2002.1018018.

[40] (2014). OMG: MDA guide revision 2.0, Object Management Group, [Online]. Available: `http://www.omg.org/cgi-bin/doc?ormsc/14-06-01` (visited on 06/26/2018).

[41] S. Zeadally, R. Hunt, Y.-S. Chen, A. Irwin, and A. Hassan, "Vehicular Ad Hoc networks (VANETs): Status, results, and challenges," *Telecommunication Systems*, vol. 50, no. 4, pp. 217–241, Aug. 2012, ISSN: 1018-4864. DOI: `10.1007/s11235-010-9400-5`. [Online]. Available: `http://dx.doi.org/10.1007/s11235-010-9400-5`.

[42] A. Masrur, M. Kit, T. Bures, and W. Hardt, "Towards component-based design of safety-critical cyber-physical applications," in *2014 17th Euromicro Conference on Digital System Design*, Aug. 2014, pp. 254–261. DOI: `10.1109/DSD.2014.87`. [Online]. Available: `https://doi.org/10.1109/DSD.2014.87`.

[43] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973, ISSN: 0004-5411. DOI: `10.1145/321738.321743`. [Online]. Available: `http://doi.acm.org/10.1145/321738.321743`.

[44] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, Sep. 1993. [Online]. Available: `http://igm.univ-mlv.fr/~masson/pdfANDps/audsley93applying.pdf`.

[45] *IEEE 802.11p standard: Wireless LAN MAC and PHY specifications amendment 6: Wireless access in vehicular environments*, `http://standards.ieee.org/findstds/standard/802.11p-2010.html`.

[46] *IEEE 802.1Q standard: LANs and WANs – MAC bridges and virtual bridged LANs*, `http://standards.ieee.org/findstds/standard/802.1Q-2011.html`.

[47] N. Shah, F. Bastani, S. Kumar, and I. L. Yen, "Real-time car-to-car communication protocol for intersecting roads," in *Proceedings of the International Conference on ITS Telecommunications (ITST)*, Oct. 2008, pp. 412–417. DOI: `10.1109/ITST.2008.4740297`. [Online]. Available: `https://doi.org/10.1109/ITST.2008.4740297`.

[48] N. Shah, S. Kumar, F. Bastani, and I.-L. Yen, "Optimization models for assessing the peak capacity utilization of intelligent transportation systems," *European Journal of Operational Research*, vol. 216, no. 1, pp. 239–251, 2012, ISSN: 0377-2217. DOI: `10.1016/j.ejor.2011.07.032`. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0377221711006643`.

[49] S.-Y. Pyun, H. Widiarti, Y.-J. Kwon, D.-H. Cho, and J.-W. Son, "TDMA-based channel access scheme for V2I communication system using smart antenna," in *Proceedings of the IEEE Conference on Vehicular Networking (VNC)*, Dec. 2010, pp. 209–214. DOI: `10.1109/VNC.2010.5698227`. [Online]. Available: `https://doi.org/10.1109/VNC.2010.5698227`.

[50] S.-Y. Pyun, H. Widiarti, Y.-J. Kwon, J.-W. Son, and D.-H. Cho, "Group-based channel access scheme for a V2I communication system using smart antenna," *IEEE Communications Letters*, vol. 15, no. 8, pp. 804–806, Aug. 2011, ISSN: 1089-7798. DOI: 10.1109/LCOMM.2011.060811.110324. [Online]. Available: https://doi.org/10.1109/LCOMM.2011.060811.110324.

[51] D. Weyns, S. Malek, and J. Andersson, "Forms: A formal reference model for self-adaptation," in *Proceedings of the 7th International Conference on Autonomic Computing*, ser. ICAC '10, Washington, DC, USA: ACM, 2010, pp. 205–214, ISBN: 978-1-4503-0074-2. DOI: 10.1145/1809049.1809078. [Online]. Available: http://doi.acm.org/10.1145/1809049.1809078.

[52] A. Varga and R. Hornig, "An overview of the /omnet++(simulation environment," in *In Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, 2008, ISBN: 978-963-9799-20-2. DOI: 10.1.1.231.4511. [Online]. Available: https://omnetpp.org/doc/workshop2008/omnetpp40-paper.pdf.

[53] (2018). Multi-agent transport simulation, MATSim Community, [Online]. Available: http://www.matsim.org (visited on 06/26/2018).

[54] (2018). Openstreetmap, OpenStreetMap contributors, [Online]. Available: http://www.openstreetmap.org (visited on 06/26/2018).

[55] G. J. Pottie and W. J. Kaiser, "Wireless integrated network sensors," *Commun. ACM*, vol. 43, no. 5, pp. 51–58, May 2000, ISSN: 0001-0782. DOI: 10.1145/332833.332838. [Online]. Available: http://doi.acm.org/10.1145/332833.332838.

[56] M. MARIN-PERIANU, N. Meratnia, P. Havinga, L. M. S. D. Souza, J. Muller, P. Spiess, S. Haller, T. Riedel, C. Decker, and G. Stromberg, "Decentralized enterprise systems: A multiplatform wireless sensor network approach," *IEEE Wireless Communications*, vol. 14, no. 6, pp. 57–66, Dec. 2007, ISSN: 1536-1284. DOI: 10.1109/MWC.2007.4407228. [Online]. Available: https://doi.org/10.1109/MWC.2007.4407228.

[57] N. Cai, M. Gholami, L. Yang, and R. W. Brennan, "Application-oriented intelligent middleware for distributed sensing and control," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 947–956, Nov. 2012, ISSN: 1094-6977. DOI: 10.1109/TSMCC.2011.2174982. [Online]. Available: https://doi.org/10.1109/TSMCC.2011.2174982.

[58] F.-J. Wu, Y.-F. Kao, and Y.-C. Tseng, "From wireless sensor networks towards cyber physical systems," *Pervasive and Mobile Computing*, vol. 7, no. 4, pp. 397–413, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574119211000368.

[59] C. Boldrini, M. Conti, and A. Passarella, “Social-based autonomic routing in opportunistic networks,” in *Autonomic Communication*. Boston, MA: Springer US, 2009, pp. 31–67, ISBN: 978-0-387-09753-4. DOI: 10.1007/978-0-387-09753-4_2. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-09753-4_2.

[60] Y. Yang, J. Wang, and R. H. Kravets, “Designing routing metrics for mesh networks,” in *IEEE WiMesh*, 2005, pp. 1–9. [Online]. Available: http://mobius.cs.uiuc.edu/system/files/wimesh05.pdf.

[61] M. Musolesi and C. Mascolo, “Car: Context-aware adaptive routing for delay-tolerant mobile networks,” *IEEE Transactions on Mobile Computing*, vol. 8, no. 2, pp. 246–260, Feb. 2009, ISSN: 1536-1233. DOI: 10.1109/TMC.2008.107. [Online]. Available: https://ieeexplore.ieee.org/iel5/7755/4731215/04585387.pdf.

[62] T. ETSI, *102 636-4-1:” intelligent transport systems (ITS); vehicular communications; geonetworking; part 4: Geographical addressing and forwarding for point-to-point and point-to-multipoint communications; sub-part 1: Media-independent functionality” v1. 1.1 (2011-06)*, 2011. [Online]. Available: http://www.etsi.org/deliver/etsi_ts/102600_102699/1026360401/01.01.01_60/ts_1026360401v010101p.pdf.

[63] Z. J. Haas, M. R. Pearlman, and P. Samar, *The zone routing protocol (ZRP) for Ad Hoc networks*, IETF Internet Draft, Jul. 2002.

[64] I. Stojmenovic, “Machine-to-machine communications with in-network data aggregation, processing, and actuation for large-scale cyber-physical systems,” *IEEE Internet of Things Journal*, vol. 1, no. 2, pp. 122–128, 2014. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6766661&tag=1.

[65] J. Wan, M. Chen, F. Xia, L. Di, and K. Zhou, “From machine-to-machine communications towards cyber-physical systems,” *Computer Science and Information Systems*, vol. 10, no. 3, pp. 1105–1128, 2013. [Online]. Available: http://www.doiserbia.nb.rs/Article.aspx?ID=1820-02141300018W&AspxAutoDetectCookieSupport=1#.VsTPZUmsxpg.

[66] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC ’12, Helsinki, Finland: ACM, 2012, pp. 13–16, ISBN: 978-1-4503-1519-7. DOI: 10.1145/2342509.2342513. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342513.

[67] L. M. Vaquero and L. Rodero-Merino, “Finding your way in the fog: Towards a comprehensive definition of fog computing,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 27–32, Oct. 2014, ISSN: 0146-4833. DOI: 10.1145/2677046.2677052. [Online]. Available: http://doi.acm.org/10.1145/2677046.2677052.

[68] S. Becker, H. Koziolek, and R. Reussner, "The palladio component model for model-driven performance prediction," *Systems and Software*, vol. 82, no. 1, pp. 3–22, Jan. 2009, ISSN: 0164-1212. DOI: 10.1016/j.jss.2008.03.066. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2008.03.066.

[69] (2018). AUTOSAR: Layered software architecture, AUTOSAR consortium, [Online]. Available: http://autosar.org/download/R4.0/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf (visited on 06/26/2018).

[70] K. Klobedanz, C. Kuznik, A. Thuy, and W. Mueller, "Timing modeling and analysis for AUTOSAR-based software development - a case study," in *Proceedings of Conference on Design, Automation, and Test in Europe (DATE)*, Mar. 2010, pp. 642–645. DOI: 10.1109/DATE.2010.5457125. [Online]. Available: https://doi.org/10.1109/DATE.2010.5457125.

[71] J. E. Kim, O. Rogalla, S. Kramer, and A. Hamann, "Extracting, specifying and predicting software system properties in component based real-time embedded software development," in *Proceedings of the International Conference on Software Engineering (ICSE)*, May 2009, pp. 28–38. DOI: 10.1109/ICSE-COMPANION.2009.5070961. [Online]. Available: https://doi.org/10.1109/ICSE-COMPANION.2009.5070961.

[72] T. Bures, J. Carlson, I. Crnkovic, S. Sentilles, and A. V. Feljan, "ProCom – the progress component model reference manual, version 1.0," Mälardalen University, Tech. Rep., Jun. 2008, ISRN: MDH-MRTC-230/2008-1-SE. [Online]. Available: http://www.es.mdh.se/publications/1279-.

[73] K. Hänninen, J. Mäki-Turja, M. Nolin, M. Lindberg, J. Lundbäck, and K. Lundbäck, "The Rubus component model for resource constrained real-time systems," in *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, Jun. 2008, pp. 177–183. DOI: 10.1109/SIES.2008.4577697. [Online]. Available: https://doi.org/10.1109/SIES.2008.4577697.

[74] S. Burmester, H. Giese, and M. Tichy, "Model-driven development of reconfigurable mechatronic systems with mechatronic UML," in *Model Driven Architecture*, U. Aßmann, M. Aksit, and A. Rensink, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 47–61, ISBN: 978-3-540-31819-4. DOI: 10.1007/11538097_4. [Online]. Available: https://doi.org/10.1007/11538097_4.

[75] H. Giese, S. Burmester, W. Schäfer, and O. Oberschelp, "Modular design and verification of component-based mechatronic systems with online-reconfiguration," in *Proceedings of the 12th ACM SIGSOFT Twelfth International Symposium on Foundations of Software Engineering*, ser. SIGSOFT '04/FSE-12, Newport Beach, CA, USA: ACM, 2004, pp. 179–188, ISBN: 1-58113-855-5. DOI: 10.1145/1029894.1029920. [Online]. Available: http://doi.acm.org/10.1145/1029894.1029920.

[76] H. Giese, M. Tichy, S. Burmester, W. Schäfer, and S. Flake, "Towards the compositional verification of real-time uml designs," in *Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. ESEC/FSE-11, Helsinki, Finland: ACM, 2003, pp. 38–47, ISBN: 1-58113-743-5. DOI: `10.1145/940071.940078`. [Online]. Available: `http://doi.acm.org/10.1145/940071.940078`.

[77] A. Basu, M. Bozga, and J. Sifakis, "Modeling heterogeneous real-time components in BIP," in *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, ser. SEFM '06, Washington, DC, USA: IEEE Computer Society, Sep. 2006, pp. 3–12, ISBN: 0-7695-2678-0. DOI: `10.1109/SEFM.2006.27`. [Online]. Available: `http://dx.doi.org/10.1109/SEFM.2006.27`.

[78] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada, "Using maude," in *Fundamental Approaches to Software Engineering*, T. Maibaum, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 371–374, ISBN: 978-3-540-46428-0. DOI: `10.1007/3-540-46428-X_27`. [Online]. Available: `https://dx.doi.org/10.1007/3-540-46428-X_27`.

[79] S. Kraus, O. Shehory, and G. Taase, "Coalition formation with uncertain heterogeneous information," in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '03, Melbourne, Australia: ACM, 2003, pp. 1–8, ISBN: 1-58113-683-8. DOI: `10.1145/860575.860577`. [Online]. Available: `http://doi.acm.org/10.1145/860575.860577`.

[80] R. Bruno, M. Conti, and E. Gregori, "Mesh networks: Commodity multihop ad hoc networks," *IEEE Communications Magazine*, vol. 43, no. 3, pp. 123–131, Mar. 2005, ISSN: 0163-6804. DOI: `10.1109/MCOM.2005.1404606`. [Online]. Available: `https://dx.doi.org/10.1109/MCOM.2005.1404606`.

[81] R. Friedman, D. Gavidia, L. Rodrigues, A. C. Viana, and S. Voulgaris, "Gossiping on manets: The beauty and the beast," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 5, pp. 67–74, Oct. 2007, ISSN: 0163-5980. DOI: `10.1145/1317379.1317390`. [Online]. Available: `http://doi.acm.org/10.1145/1317379.1317390`.

[82] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking up data in p2p systems," *Commun. ACM*, vol. 46, no. 2, pp. 43–48, Feb. 2003, ISSN: 0001-0782. DOI: `10.1145/606272.606299`. [Online]. Available: `http://doi.acm.org/10.1145/606272.606299`.

[83] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, ser. STOC '97, El Paso, Texas, USA: ACM, 1997, pp. 654–663, ISBN: 0-89791-888-6. DOI: `10.1145/258533.258660`. [Online]. Available: `http://doi.acm.org/10.1145/258533.258660`.

[84] K. Dhara, Y. Guo, M. Kolberg, and X. Wu, "Overview of structured peer-to-peer overlay algorithms," in *Handbook of Peer-to-Peer Networking*. Boston, MA: Springer US, 2010, pp. 223–256, ISBN: 978-0-387-09751-0. DOI: 10.1007/978-0-387-09751-0_9. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-09751-0_9.

[85] O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation," *Artif. Intell.*, vol. 101, no. 1-2, pp. 165–200, May 1998, ISSN: 0004-3702. DOI: 10.1016/S0004-3702(98)00045-9. [Online]. Available: http://dx.doi.org/10.1016/S0004-3702(98)00045-9.

[86] T. Michalak, J. Sroka, T. Rahwan, M. Wooldridge, P. McBurney, and N. R. Jennings, "A distributed algorithm for anytime coalition structure generation," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, ser. AAMAS '10, Toronto, Canada: International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 1007–1014, ISBN: 978-0-9826571-1-9. [Online]. Available: http://dl.acm.org/citation.cfm?id=1838206.1838342.

[87] T. Rahwan, T. Michalak, M. Wooldridge, and N. R. Jennings, "Anytime coalition structure generation in multi-agent systems with positive or negative externalities," *Artif. Intell.*, vol. 186, no. C, pp. 95–122, Jul. 2012, ISSN: 0004-3702. DOI: 10.1016/j.artint.2012.03.007. [Online]. Available: http://dx.doi.org/10.1016/j.artint.2012.03.007.

[88] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé, "Coalition structure generation with worst case guarantees," *Artificial Intelligence*, vol. 111, no. 1, pp. 209–238, 1999, ISSN: 0004-3702. DOI: 10.1016/S0004-3702(99)00036-3. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370299000363.

[89] (2018). Akka, Lightbend, [Online]. Available: http://akka.io (visited on 06/26/2018).

[90] T. Léauté, B. Ottens, and R. Szymanek, "FRODO 2.0: An open-source framework for distributed constraint optimization," in *Proceedings of the IJCAI" 09 Distributed Constraint Reasoning Workshop (DCR" 09)*, 2009, pp. 160–164. [Online]. Available: https://infoscience.epfl.ch/record/146585.

[91] M. Wahbi, R. Ezzahir, C. Bessiere, and E.-H. Bouyakhf, "DisChoco 2: A platform for distributed constraint reasoning," *Proceedings of DCR*, vol. 11, pp. 112–121, 2011. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.208.4895.

[92] E. A. Sultanik, R. N. Lass, and W. C. Regli, "DCOPolis: A framework for simulating and deploying distributed constraint optimization algorithms," 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.86.6632.

[93] M. E. Gaston and M. desJardins, "Agent-organized networks for dynamic team formation," in *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '05, The Netherlands: ACM, 2005, pp. 230–237, ISBN: 1-59593-093-0. DOI: 10.1145/1082473.1082508. [Online]. Available: http://doi.acm.org/10.1145/1082473.1082508.

[94] (Jul. 2018). Robocup, RoboCup initiative, [Online]. Available: http://www.robocup.org (visited on 06/26/2018).

[95] J. Parker, E. Nunes, J. Godoy, and M. Gini, "Exploiting spatial locality and heterogeneity of agents for search and rescue teamwork*," *J. Field Robot.*, vol. 33, no. 7, pp. 877–900, Oct. 2016, ISSN: 1556-4959. DOI: 10.1002/rob.21601. [Online]. Available: https://doi.org/10.1002/rob.21601.

[96] Y. Zhang, M. Qiu, C. W. Tsai, M. M. Hassan, and A. Alamri, "Health-cps: Healthcare cyber-physical system assisted by cloud and big data," *IEEE Systems Journal*, vol. 11, no. 1, pp. 88–95, Mar. 2017, ISSN: 1932-8184. DOI: 10.1109/JSYST.2015.2460747. [Online]. Available: https://dx/dio.org/10.1109/JSYST.2015.2460747.

[97] A. Hahn, B. Kregel, M. Govindarasu, J. Fitzpatrick, R. Adnan, S. Sridhar, and M. Higdon, "Development of the powercyber scada security testbed," in *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, ser. CSIIRW '10, Oak Ridge, Tennessee, USA: ACM, 2010, 21:1–21:4, ISBN: 978-1-4503-0017-9. DOI: 10.1145/1852666.1852690. [Online]. Available: http://doi.acm.org/10.1145/1852666.1852690.

[98] A. Hahn, A. Ashok, S. Sridhar, and M. Govindarasu, "Cyber-physical security testbeds: Architecture, application, and evaluation for smart grid," *IEEE Transactions on Smart Grid*, vol. 4, no. 2, pp. 847–855, Jun. 2013, ISSN: 1949-3053. DOI: 10.1109/TSG.2012.2226919. [Online]. Available: https://dx.doi.org/10.1109/TSG.2012.2226919.

[99] J. Wuttke, Y. Brun, A. Gorla, and J. Ramaswamy, "Traffic routing for evaluating self-adaptation," in *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Jun. 2012, pp. 27–32. DOI: 10.1109/SEAMS.2012.6224388. [Online]. Available: https://doi.org/10.1109/SEAMS.2012.6224388.

[100] D. Weyns and R. Calinescu, "Tele assistance: A self-adaptive service-based system examplar," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '15, Florence, Italy: IEEE Press, 2015, pp. 88–92. [Online]. Available: http://dl.acm.org/citation.cfm?id=2821357.2821373.

List of Figures

List of Tables

Glossary

Beyond Control Entity is an abstraction of an object that is out of control of the system in question. It allows to plug such an object into an existing system in its native way. With respect to the EBCS an external object can be seen as a component that lack processes and its knowledge is based on the observations of such object. Such component can take part in the ensembles. 44

bipartite ensemble is an ensemble specified using the classical *member - coordinator* model. See Chapter 7. 65, 67, 68, 70, 72, 73, 75, 76, 79, 96, 109, 113, 122, 123, 160

Bluetooth is a technology standard for wireless exchanging data over short distances. It uses short-wavelength Ultra High Frequency (UHF) radio waves and random frequency hopping. 4, 17

CDEECo++ [1] is a C++ framework using DEECo architecture focusing on embedded applications with hard real-time requirements. 34, 93, 97, 100, 109, 113–115, 123, 160

component is a software package hosted on a computation node that includes knowledge defining its interface to other components, and processes implementing the business logic. 5, 14, 15, 20, 35, 36, 38, 40, 44, 49–51, 53, 54, 58–62, 66–82, 85, 86, 88–91, 95–99, 104, 110–116, 122, 124, 125, 135, 146, 151, 152, 155, 159

costmap represents a cost for passing through a particular area in space. Usually it is kept in a form of bitmap. The pixel value represents the cost. 123

depth camera is a device similar to camera that can capture not only color but also distance of every pixel in the view to the camera. 4, 12

Duck-Typing is a dynamic type system based on the features of an object. i.e. the object is considered an instance of a Duck if it Quacks like a duck and walks like a duck. 67

edge cloud is a deployment of cloud services, i.e. servers, databases, on network nodes located at the edge of the network in order to lower response time and maximize throughput towards client devices. 59, 60, 72, 143

ensemble is a dynamic connector of a group of components described by a membership condition. See bipartite ensemble and intelligent ensemble. 5,

[1] `https://github.com/d3scomp/cdeeco`

7, 14, 15, 20, 32, 34–38, 40–47, 49–62, 65–76, 78–82, 85, 86, 88–92, 96–100, 104, 105, 110–117, 122, 124–126, 135–138, 144, 146, 147, 151, 152, 155, 159, 160

ensemble instance is a group of components following particular ensemble specification. 52, 54, 58, 60, 62, 69, 73–75, 89–92, 104, 114, 138

FreeRTOS or Free Real-Time Operating System[2] is a simple OS implementing real-time task scheduling and synchronization on embedded platforms.. 100, 113, 115

fully-autonomous car is a car with self control ability capable of standalone navigation through the street network entirely without a driver. 35, 38, 42

Gossip is a network protocol inspired by an epidemic spread of information in a social network. Using a gossip protocol the information is spread by nodes that are already aware of the information. 80–82, 86, 91, 92, 97, 135, 136

hard real-time system operation is subject to deadlines. Missing a deadline is considered a total failure of the system. An example of such system is an airbag inflation system which needs to inflate the airbag exactly on time, doing so earlier or later can cause harm or kill the passenger. 7, 16, 34, 41–43, 63, 65, 93, 97, 109, 113, 115

IEEE 802.11p is a wireless technology similar to Wi-Fi operating on a licensed band of 5.9 GHz that is used as a physical layer of the WAVE protocol. 16, 17

IEEE 802.15.4 is a wireless technology focusing on short range low bandwidth data broadcast. It shares 2.4 GHz band with Wi-Fi and is used as physical layer for ZigBee. 16, 17, 26, 34, 37, 38, 61, 75, 112, 115, 119

IEEE 802.1Q or VLAN tagging is a protocol that allows to operate a set of logical networks on top of a physical one. This protocol also enable setting priorities on logical traffic. 101, 130

IEEE 802.3 Ethernet or simply Ethernet is a wired networking technology adopted in networks ranging from local to global. 15, 16, 26, 101, 103, 105

Industry 4.0 called also smart factory or advanced manufacturing is a current trend in manufacturing that encompasses improved data exchange, CPS, IoT, and widespread digitization. iii, 5, 6, 9–11, 34, 35, 37, 38, 59, 159

intelligent ensemble is a form of ensembles using declarative ensemble specification supporting multiple component roles and ensemble fitness specification. See Chapter 7 for details. 34, 65, 67, 72–76, 109, 117, 144, 160

[2]`https://www.freertos.org`

JDEECo [3] is a Java framework using DEECo architecture. Currently it is the most feature complete framework based on DEECo. It has broad support for simulation execution and also supports real device deployment.. 14, 15, 34, 66–68, 70, 71, 80–83, 109–113, 119, 123–125, 135, 136, 147, 160

knowledge of the component is set of data fields that define its public interface. The knowledge is accessed by component processes and is subject to exchange defined by ensemble. 5, 14, 15, 34, 49–51, 57, 59–62, 68–78, 80–82, 85, 86, 89, 90, 97–100, 104–106, 110–117, 124, 129, 135, 144–147, 151–153, 159

knowledge exchange is a process of mapping knowledge among ensemble members once the ensemble is formed. 5, 14, 15, 32, 35, 42, 46, 49, 50, 59, 60, 65, 67, 70–72, 75, 76, 79, 81, 89, 90, 96–99, 104, 105, 110, 114, 136

model@runtime is an approach where architecture models are preserved at runtime in order to provide self reflection used mainly for software adaptations. 146

NED or Network Description, is a DSL used to capture OMNeT++ modules. 25

OMNeT++ or Objective Modular Network Testbed in C++[4] is an extensible network simulator written in C++ with support for precise simulation of different network hardware. 25, 26, 109, 111, 121, 125, 127, 129, 137, 138, 140, 160

OpenStreetMap [5] is an open map of the world created by a community and shared under a permissive license. The fact that the source data are available in XML makes it suitable for creation of various customized maps that can be used as inputs for traffic simulators. 23, 24, 26, 36, 134

platoon or road train is a chain of cars on highway that drive close enough to reduce air drag while allowing easier autonomous driving for cars internal to the train. 6, 12, 13, 40, 41, 44, 45, 53, 54, 58, 62–64, 77, 79–81, 134–137, 160

process is a piece of business logic running on top of a knowledge of a component. Usually it encompasses a method that is periodically scheduled or triggered by knowledge change. 5, 14, 49, 50, 71, 72, 78, 95, 99–101, 104, 105, 107, 110, 112–116, 122, 124, 126, 159

Promela [6] is a C like language used to capture model of the system so that it cab be verified with Spin. 148

PyDEECo [7] is a Python framework using DEECo architecture. 34, 109, 113, 116, 117, 160

[3]`https://github.com/d3scomp/JDEECo`
[4]`https://www.omnetpp.org`
[5]`http://www.openstreetmap.org`
[6]`http://spinroot.com/spin/Man/Quick.html`
[7]`https://github.com/d3scomp/pydeeco`

real-time system operation is subject to deadlines. Missing a deadline is considered a problem for the system operation. Either the value produced by the system can be lowered in case of the soft real-time systems or the system may fail due to deadline miss in case of hard real-time systems. 7, 15, 16, 21, 32, 33, 35, 37, 42, 43, 49, 55, 59, 61–63, 65, 72, 93, 95, 97, 99, 100, 113, 115, 127, 148–150, 159

ROSViz is a graphical tool that can visualize operation of a ROS system. It can display the scene in 3D and display map, model of the robot, different types of particles, and different messages of various types. 21

semi-autonomous car is a car with partial self control ability. In context of this thesis a car that can follow lane on a highway without drivers attention is assumed. 35, 38

soft real-time system operation is subject to deadlines. Missing a deadline is considered a penalty to the system operation and may reduce utility of the system. An example of such system is a public transport. When a bus comes a bit later than scheduled the overall utility of the system is reduced, but the system remains in operation with reduced utility. 14, 43, 63

Spin [8] is tool for software verification. 148

Stage is a multi-robot simulator focusing on simple, but high performance simulation of high quantities of robots. Stage is part of Player-Stage project[9]. For details see Chapter 2 Section 2.5.2. 22, 35, 109, 121–123, 125, 127, 160

test-bed is a piece of software intended to provide test scenario and matching simulation environment used to practice development of a particular system. 7, 33, 34, 65, 66, 109, 119, 121–123, 125–127, 155–158, 160

Turtlebot [10] is a vacuum cleaner based model of robot capable of carrying a laptop and other equipment. It is designed as a experimentation platform for robotic systems.. 21, 109, 112, 113, 122, 160

Wi-Fi (WiFi) is a wireless local area networking technology with devices based on the IEEE 802.11 standard. 4, 15–17, 32, 37, 63, 75

ZigBee is a wireless communication technology build on top of IEEE 802.15.4 standard radio. It focuses on PAN networking while using multi-hop ad-hoc routing protocol to deliver messages to out-of-sight nodes. 17, 32

[8] http://spinroot.com
[9] http://playerstage.sourceforge.net
[10] https://www.turtlebot.com

Acronyms

4G The 4$^{\text{th}}$ Generation mobile networks. 4, 19, 61, 62, 71

5G The 5$^{\text{th}}$ Generation mobile networks. 4, 13, 19, 38, 61, 62, 71

ACE Autonomic Component Ensembles. 87

ACRC Autonomous Cleaning Robots Coordination. 119, 121, 122, 124, 126

AMCL Adaptive Monte-Carlo Localization. 21, 22, 119, 120, 125

AP Access Point. 101, 103, 108, 129, 132

ATRP Automated Traffic Routing Problem. 157

AUTOSAR AUTomotive Open System ARchitecture. 148, 149

BIP Behavior, Interaction, Priority. 150

BML Battle Management Language. 10

BNEP Bluetooth Network Encapsulation Protocol. 17

BSW Basic Software. 148

C2I Car to Infrastructure. 93, 94

CNC Computer Numerical Control. 3

CPS Cyber-Physical System. 4, 16, 144–146, 156–158

DCOP Distributed Constraints Optimization Problem. 154, 155

DEECo Dependable Emergent Ensembles of Components. 14, 15, 32–35, 49, 50, 57, 65, 67, 69, 74, 75, 78–80, 85, 88, 89, 93, 95, 97–100, 105–107, 109–111, 113, 116, 121–125, 127, 129, 136, 146–148, 150–153, 159, 160

DHT Distributed Hash-Tables. 153

DSC Distributed Sensing and Control. 144

DSL Domain Specific language. 21, 25, 88, 89, 91, 95, 109, 113, 116, 147

EBCS Ensemble Based Component Systems. 5–7, 9, 14, 32–35, 61, 65, 69, 71, 93, 119, 144, 147, 148, 152, 154, 155, 159, 160

ECU Electronic Control Unit. 148

EDL Ensemble Definition Language. 20, 73, 109

EMF Eclipse Modeling Framework. 148

FDD Frequency Division Duplexing. 19

FIFO First In First Out. 17

GPS Global Position System. 4, 36, 38, 49–51

GUI Grahical User Interface. 26

Helena Handling massively distributed systems with ELaborate ENsemble Architectures. 14, 147, 148

ICS Intelligent Crossroad System. 65, 93–95, 98–107, 129, 130, 132, 160

IoT Internet of Things. 4, 6, 9, 10, 19, 35, 143

IP Internet Protocol. 13, 15, 17, 19, 61, 71, 75, 134–136, 153

jRESP Java Run-time Environment for SCEL Programs. 14, 146

LIDAR Light Detection And Ranging. 4, 12, 21, 49

LTE Long-Term evolution. 13, 19, 26, 27, 38

M2M Machine To Machine. 145

MAC Media Access Control. 61, 63

MANET Mobile ad-hoc network. 18, 61, 62, 77, 80, 86, 87, 91, 92, 119, 125, 127, 135, 138, 144, 145, 152

MAS Multi-Agent Systems. 154, 155

MATSim Multi-Agent Transport Simulation. 23, 24, 36, 111, 134, 160

MDA Model driven Architecture. 88

MIMO Multiple Input Multiple Output. 19

OBU On-Board Unit. 17

OS Operating System. 95, 97, 100, 148

PAN Personal Area Network. 17

PCM Palladio component model. 148

PNG Portable Network Graphics. 122

ProCom Progress Component Model. 149

QoS Quality of Service. 11, 19, 34, 72, 148

RADAR Radio Detection and Ranging. 12

ROS Robot Operating System. 20–23, 35, 109, 111–113, 119–122, 125, 127

RSU Road Side Unit. 17

RTE Runtime Environment. 148, 149

SCADA Supervisory Control And Data Acquisition. 157

SCEL Service Component Ensemble Language. 14, 146

sCPS smart Cyber-Physical Systems. iii, 3–7, 9, 14, 18, 19, 24, 31–33, 38, 65, 66, 68, 71, 77, 79, 86, 87, 93, 109, 113, 119, 121, 129, 140, 143–146, 151, 152, 155, 156, 158–160

SDMA Space Division Multiple Access. 101

SLAM Simultaneous Localization and Mapping. 10

SMT Satisfiability Modulo Theories. 20, 73, 74

SUMO Simulation of Urban MObility. 24–26, 36, 160

SWC Software Circuit. 149, 150

TADL Timing Augmented Description Language. 148

TAS Tele Assistance System. 158

TCP Transmission Control Protocol. 24

TDD Time Division Duplexing. 19

TDMA Time Division Multiple Access. 72, 102

TIMMO TIMing MOdel. 148

TraCI Traffic Control Interface. 24

UAV Unmanned Aerial Vehicle. 9, 10, 36

UGV Unmanned Ground Vehicle. 9, 36

UUV Unmanned Underwater Vehicle. 36

V2V Vehicle to Vehicle communication. 72

VANET Vehicular ad-hoc network. 18, 80, 93, 101, 145

WAVE Wireless Access in Vehicular Environments. 12, 17, 38, 61, 63, 72, 75, 76

WCET Worst-Case Execution Time. 100, 104, 132

WCRT Worst-Case Response Time. 99, 132

WSN Wireless Sensor Network. 16, 143–145

List of publications

[1] M. Kit, F. Plasil, V. Matena, T. Bures, and O. Kovac, "Employing domain knowledge for optimizing component communication," in *Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering*, ser. CBSE '15, Montréal, QC, Canada: ACM, 2015, pp. 59–64, ISBN: 978-1-4503-3471-6. DOI: `10.1145/2737166.2737172`. [Online]. Available: `http://doi.acm.org/10.1145/2737166.2737172`.

[2] V. Matena, A. Masrur, and T. Bures, "An ensemble-based approach for scalable QoS in highly dynamic CPS," in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Aug. 2017, pp. 234–238, ISBN: 978-1-5386-2141-7. DOI: `10.1109/SEAA.2017.62`. [Online]. Available: `https://dx.doi.org/10.1109/SEAA.2017.62`.

[3] T. Bures, V. Matena, R. Mirandola, L. Pagliari, and C. Trubiani, "Performance modelling of smart cyber-physical systems," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18, Berlin, Germany: ACM, 2018, pp. 37–40, ISBN: 978-1-4503-5629-9. DOI: `10.1145/3185768.3186306`. [Online]. Available: `http://doi.acm.org/10.1145/3185768.3186306`.

[4] O. Štumpf, T. Bureš, and V. Matěna, "Security and trust in data sharing smart cyber-physical systems," in *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ser. ECSAW '15, Dubrovnik, Cavtat, Croatia: ACM, 2015, 18:1–18:4, ISBN: 978-1-4503-3393-1. DOI: `10.1145/2797433.2797451`. [Online]. Available: `http://doi.acm.org/10.1145/2797433.2797451`.

[5] A. Masrur, M. Kit, V. Matěna, T. Bureš, and W. Hardt, "Component-based design of cyber-physical applications with safety-critical requirements," *Microprocessors and Microsystems*, vol. 42, pp. 70–86, 2016, ISSN: 0141-9331. DOI: `10.1016/j.micpro.2016.01.007`. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0141933116000107`,

Impact Factor: 1.025, CiteScore: 1.11, SCImago Journal Rank: 0.238

Statistics captured on 05/09/2018 from `https://www.journals.elsevier.com/microprocessors-and-microsystems`.

[6] V. Matena, T. Bures, I. Gerostathopoulos, and P. Hnetynka, "Model problem and testbed for experiments with adaptation in smart cyber-physical systems," in *Proceedings of the 11th International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '16, Austin, Texas: ACM, 2016, pp. 82–88, ISBN: 978-1-4503-4187-5. DOI:

10.1145/2897053.2897065. [Online]. Available: http://doi.acm.org/10.1145/2897053.2897065.

[7] T. Bures, P. Hnetynka, F. Krijt, V. Matena, and F. Plasil, "Smart coordination of autonomic component ensembles in the context of ad-hoc communication," in *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques: 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10–14, 2016, Proceedings, Part I*, T. Margaria and B. Steffen, Eds. Cham: Springer International Publishing, 2016, pp. 642–656, ISBN: 978-3-319-47166-2. DOI: 10.1007/978-3-319-47166-2_45. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-47166-2_45.