Real-Time Coordination of Mobile Autonomous Entities

Mélanie Bouroche

A thesis submitted to the University of Dublin, Trinity College

in fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

December 2007

Declaration

I, the undersigned, declare that this work has not previously been submitted to this or any other University, and that unless otherwise stated, it is entirely my own work. I agree that Trinity College Library may lend or copy this thesis upon request.

Mélanie Bouroche

Dated: 28th May 2008

Acknowledgements

I have been waiting for this moment for so long. At last, the thesis done, only the acknowledgements to write! The acknowledgements – being the only place where the writing can be a bit more creative and can allow for feelings to be expressed – would, I believed, show that I too can write moving words when given the space and opportunity. Alas, now that the moment has arrived, all the great ideas and inspirations that came over the years are gone; most probably killed by the stress-versus-sleep war of these last few weeks. Well, my acknowledgements will be simple, but to the point.

First and foremost, my thanks to Prof. Vinny Cahill, my supervisor, for letting me explore my own ideas; for trying to understand them even when I could not explain them; and for helping me control my stress levels.

To Ashley, for his help on the implementation of ComhorMod's GUI.

To Aline, Barbara, Cormac, Andronikos, Ray, Alan and Stefan, for having proof-read chapters of this thesis.

To my parents, for letting me do what I want and for supporting me in the best way they can, despite the distance.

To all the people in my research group, DSG, for their support; in particular to Andronikos, Kulpreet, Tim and David for hugs when I needed them and for making sure I took breaks, and to Aline for moral support and scientific advice.

To my sister Gaëlle and all my friends, for keeping me sane; in particular, to Emeline for always being there, to Eoin and Mathieu for helping me stay in contact with the real world, and to Astride, for having been there all along.

Mélanie Bouroche
University of Dublin, Trinity College
December 2007

Abstract

Progress in miniaturisation of computing devices, wireless communication, and sensing technology are encouraging the deployment of autonomous mobile computer systems ("entities") in our everyday environment. Examples of mobile entities that operate without human control include automated guided vehicles and other mobile service robots, as well as robots for disaster rescue, unmanned aerial vehicles, and, in the future, autonomous cars. To ensure safe operation, such entities must coordinate their behaviour both with each other and with their environment. This means that their aggregated behaviour must respect some system-wide safety constraints. As the safety of humans and possibly crucial or expensive infrastructure is at stake, the coordination of these entities is safety-critical, i.e., a violation of the safety constraints could result in a catastrophe. Because entities interact with their environment, they need to cope with its pace; hence, ensuring these system-wide safety constraints implies stringent real-time requirements on coordination.

Mobile entities may use direct communication to coordinate their behaviour. Because they are mobile, they typically communicate over wireless (possibly ad hoc) networks. In wireless networks, however, communication, and in particular real-time communication, is highly unreliable and the achievable timeliness varies greatly over time and location. Environment-mediated communication, in which entities communicate by changing their environment and detecting changes made by other entities using sensors, can be used to supplement direct communication. The range and accuracy of sensors, however, are inherently limited and sensor information might be affected by environmental conditions (e.g., luminosity or temperature). All of these limitations imply that the information that entities have about other entities and their environment varies significantly over time and space. Therefore entities might not be able to reach a consensus on their collective behaviour, or even to consult each other, and might not have a complete view of their environment. Hence entities need to take decisions independently of each other, using only limited information, while still guaranteeing system-wide safety constraints.

Ensuring that system-wide safety constraints are respected by entities that take decisions independently while having access to only limited information is particularly challenging. Existing coordination models are typically consensus-based and either assume continuous real-time connectivity, or offer only best-effort guarantees. This thesis investigates an alternative to relying on consensus that exploits real-time feedback on currently available information provided by novel real-time sensing and

communication models. In this approach, entities adapt their behaviour depending on currently available information, hence making progress when it is safe to do so, while ensuring that safety will never be compromised.

This thesis presents Comhordú, a new coordination model supporting the development of entities that use this adaptive approach to coordination. Comhordú defines a formalism in which to express system-wide safety constraints and a notion of responsibility to allow enforcement of these constraints to be distributed amongst entities. The notion of consensus is replaced by contracts that bind entities a priori and allow them to make predictions about other entities' behaviours even when they do not have current information about them. In addition, the thesis describes a systematic process that allows developers to use Comhordú to translate system-wide safety constraints into requirements on the behaviour of individual entities. These requirements capture the necessary information for an entity to safely begin or continue performing any given action. During execution, each entity can, at any time, derive the set of actions that it can safely undertake, given the information that it currently has about its environment. A tool supporting the development of such entities by automating the systematic steps of the process is also presented. Finally, this thesis demonstrates that the model and associated development process can be used to solve scenarios that are not solvable using existing coordination models.

Publications Related to this Ph.D.

- Aline Senart, Mélanie Bouroche and Vinny Cahill, *Modelling an Emergency Vehicle Early-Warning System using Real-Time Feedback.* In International Journal of Intelligent Information and Database Systems (IJIIDS), volume 2, issue 1, to appear 2008.

- Mélanie Bouroche, Barbara Hughes and Vinny Cahill. *Real-time Coordination of Autonomous Vehicles.* In IEEE Intelligent Transportation Systems Conference (ITSC'06), Toronto, Canada, September 2006.

- Mélanie Bouroche and Vinny Cahill. *Coordination of Autonomous Mobile Entities.* In 4th MiNEMA Workshop, Lisbon, Portugal, July 2006.

- Mélanie Bouroche, Barbara Hughes and Vinny Cahill. *Building Reliable Mobile Applications with Space-Elastic Adaptation.* In international workshop on Mobile Distributed Computing (MDC'06), in conjunction with WoWMoM 06, Niagara Falls, New York, USA, June 2006.

- Aline Senart, Mélanie Bouroche, Barbara Hughes and Vinny Cahill. *Coordination of Safety-Critical Mobile Real-Time Embedded Systems.* In workshop on research directions for security and networking in Critical Real-Time and Embedded Systems (CRTES 06), in conjunction with RTAS 2006, San Jose, California, USA, April 2006.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis presents Comhordú, a real-time coordination model for autonomous mobile computer systems, or "entities". Using Comhordú, developers can translate system-wide safety constraints into requirements on the behaviour of autonomous mobile entities. If the behaviours of all entities fulfil these requirements, system-wide safety constraints can be ensured while entities act independently, despite having access to only limited information from sensors and unreliable wireless communication.

This chapter first describes the motivations for this work, defines the notion of coordination, outlines the limitations of wireless communication and sensor information, and then gives an overview of existing related work. Following this, the challenges to be overcome by the work described in this thesis are presented. In addition, the approach chosen for this work, its goal, contributions, and scope are outlined. Finally, a road map of the thesis and a summary of this chapter are presented.

1.1 Motivations

Progress in miniaturisation of computing devices, wireless communication, and sensing technology is encouraging the deployment of mobile entities in our everyday environment. Many of these computer systems operate without any human control, i.e., are autonomous. Examples of these autonomous entities include pet robots (Kubota et al. 2000), automated guided vehicles (AGVs) (Cawkwell 2000), and other mobile service robots (Schraft 1994), as well as robots for disaster rescue (Hirose & Fukushima 2002), unmanned aerial vehicles (UAVs) (Alighanbari et al. 2003), space robots (Fong & Nourbakhsh 2005), and, in the future, autonomous cars (Baber et al. 2005, Bouraoui et al. 2006).

These entities typically operate in the same environment as humans, other entities, and potentially expensive infrastructure. To ensure the safety of all parties, entities must coordinate their behaviour both with each other and with their environment. For example, AGVs used in factory automation need to collaborate while ensuring that they do not collide with each other or with factory workers. This means that the aggregated behaviour of all these entities must respect some system-wide safety constraints. As the safety of humans and possibly crucial or expensive infrastructure is at stake, the

coordination of entities is often safety-critical (Kopetz 1997), i.e., a violation of the safety constraints could result in a catastrophe. Furthermore, because entities interact with their environment, ensuring these system-wide safety constraints typically implies stringent real-time requirements on coordination. For example, the time available to an AGV to react to avoid another AGV is bounded depending on their speeds and trajectories. The work described in this thesis caters for this emerging class of applications composed of autonomous mobile entities and exhibiting strong reliability and timeliness requirements.

Entities need to have information about the behaviour of other entities and their environment in order to coordinate their behaviour. In mobile settings, however, the quality of the information provided by both direct communication and sensor data (under real-time requirements) is limited and varies significantly over space and time. Therefore, entities might not be able to reach consensus on their collective behaviour, or even to consult each other, and might not have a complete view of the environment.

Ensuring that system-wide safety constraints are respected by entities that take decisions independently, while having access to only limited information, is particularly challenging. This thesis investigates a new approach to this problem where entities adapt their behaviour depending on currently available information about the behaviour of other entities and their environment.

1.2 Background information

For entities to coordinate, they need to have information about their environment and the behaviour of other entities. For this purpose, entities can use both wireless communication and sensor data. This section first defines the concepts of coordination and coordination model. It then provides some background information about wireless communication and sensor data.

1.2.1 Coordination

Coordination is a relatively new concept. It was first considered between different processes or programs (see, for example, Arbab 1998). Coordination in this context was defined as a construct to form a single application from multiple components, and the purpose of coordination models and associated languages has been stated (Papadopoulos & Arbab 1998) as:

> "to provide a means of integrating a number of possibly heterogeneous components together, by interfacing with each component in such a way that the collective set forms a single application that can execute on, and take advantage of parallel and distributed systems".

Closer to our interest, Keil & Goldin (2003) define *coordination* in a multiagent system as:

> "the management of interaction among computing entities in multiagent systems, including creation and destruction of such entities and of communication links among them",

where *interaction* is:

> "the ongoing two-way or multiway exchange of data among computational entities, such that the output of one entity may causally influence the later outputs of another entity".

This definition, however, is more suited for software agents where the creation and destruction of entities is an easy and common occurrence. Furthermore, as the quality of communication between entities varies over time and distance, the notion of explicit creation and destruction of communication links seems inappropriate. Inspired by the definition of the Oxford English Dictionary of coordination (Definition 4 in *Oxford English Dictionary Online* 1989):

> "harmonious combination of agents or functions toward the production of a result",

we add the notion of a result that the entities are trying to ensure. So our definition of *coordination* is:

> "the management of interactions both amongst entities, and between entities and their environment, towards the production of a result",

where we define *interaction* as:

> "any action that may causally influence the action of other entities".

Note that our definition of interaction is more generic than the one from Keil & Goldin in that, in our view, interaction does not require the exchange of data. In this work, the result towards which entities interact includes the safety of entities and their environment.

We adopt the definition of a *coordination model* by Ciancarini (1996):

> "a triple $\{E, M, L\}$ where E is the set of coordinable entities, M is the set of coordination media, and L is the set of coordination laws that dictates how entities coordinate themselves through the given coordination media, and using a number of coordination primitives".

In the following, we review the possible coordination media.

1.2.2 Wireless communication

Mobile entities typically communicate over a wireless network. Wireless networks can be either infrastructure-based, in which case communication is managed by a set of predefined dedicated nodes, or ad hoc, where nodes spontaneously create a network with other nodes in their vicinity. Communication over a wireless link is impaired by fading, as well as diffraction, scattering, reflection, and refraction (Neskovic et al. 2000). In addition, node mobility means that connectivity and network topology change dynamically, as nodes move into and out of range of other nodes (Wang & Li 2002).

These factors make routing messages in ad hoc networks particularly challenging (McDonald & Znati 1999). Furthermore, message collisions are hard to avoid in wireless, and in particularly in ad hoc, networks, and cause unpredictable latency (Hughes & Cahill 2003). These reasons imply that communication, and in particular real-time communication, in wireless networks is highly unreliable and that the achievable timeliness varies greatly over time and location (Gaertner & Cahill 2004).

1.2.3 Sensor information and environment-mediated communication

Entities can also obtain information about their environment and other entities through sensor information. In particular, entities can use environment-mediated communication instead, or in addition to, wireless communication. Environment-mediated communication (EMC) (Gellersen et al. 1999) is a form of indirect communication, where agents communicate by changing their environment and detecting these changes. An example of environment-mediated communication in everyday life is the use of sirens by emergency vehicles: an emergency vehicle acts on its environment by producing a sound that can be detected by drivers who then know that it is arriving.

EMC has been characterised in the study of stigmergy, in which an agent's actions leave signs in the environment, that it and other agents sense and that determine their subsequent actions (Parunak 2003). The term stigmergy was first coined to refer to the nest building behaviour of termites (Grassé 1959), and the concept has since been applied to a variety of domains ranging from combinatorial optimisation (Dorigo et al. 1999) to communications networks (Schoonderwoerd et al. 1996, Caro & Dorigo 1998), peer-to-peer application design (Babaoglu et al. 2002) and robotics (Holland & Melhuish 1999).

EMC relies on the use of sensors. The range and accuracy of sensors, however, are inherently limited, so sensors can only give information about their immediate vicinity, and this information is necessarily approximate. Furthermore, sensor range and accuracy might be affected by environmental conditions (e.g., luminosity or temperature) and will therefore vary over time. This implies that, as the quality of sensor information and EMC varies over time, entities cannot safely rely only on them to get information for coordinating their behaviour.

In the following, we refer to EMC as indirect communication, and to wireless communication as direct communication to simplify the discussion.

1.3 Context

The limitations of both data communication and sensing mean that the information that entities have about their environment varies significantly over time and space. In this section, we give a brief overview of existing work in the area of real-time coordination of autonomous mobile entities, and assess how they deal with the unreliability of real-time communication and sensor data in mobile settings. A number of applications composed of autonomous mobile entities have already been deployed,

mostly in the field of robotics, using application-specific coordination mechanisms, which we review in Section 1.3.1. In contrast to these application-specific mechanisms, a number of generic coordination approaches have also been proposed for autonomous mobile entities. Most of these approaches rely on the notion of consensus. Section 1.3.2 reviews these consensus-based approaches, while Section 2.4.2 discusses existing non-consensus based approaches to coordination.

1.3.1 Application-specific coordination mechanisms

In the robotics community, a number of projects have investigated the coordination of autonomous robots. While the majority of this work relates to static robots, between which continuous real-time connectivity can be assumed, a number of applications have been built using mobile robots. The latter include, for example, AGVs (Cawkwell 2000), UAVs (Alighanbari et al. 2003), autonomous cars (Baber et al. 2005), and robots for disaster rescue (Hirose & Fukushima 2002). This work, however, typically assumes reliable connectivity, which might be reasonable in the confined and protected environments in which robots are typically deployed during experimentation, but would not be for large-scale applications in everyday environments. In contrast, the work in this thesis caters for the coordination of autonomous mobile entities even in the presence of communication failures.

1.3.2 Consensus-based approaches

A number of consensus-based coordination models have been proposed for mobile entities in ad hoc networks. These models mostly use a tuple-space approach, where entities coordinate by manipulating a shared collection of data objects, called tuples. LIME (Linda In Mobile Environment) (Murphy et al. 2001) caters for physical mobility of hosts and logical mobility of agents, by having a tuple space attached to each mobile entity. Entities then collaborate by transiently sharing their tuple spaces, creating a "global virtual data structure". EgoSpaces (Julien & Roman 2004) is an extension of LIME that introduces the concept of a view that allows nodes to specify from which nodes tuples must be gathered. Finally, Limone (Fok et al. 2004) is another LIME-inspired model, designed for use on small devices in unstable environments. None of this work, however, considers real-time guarantees.

The work of Nett et al. (Nett et al. 2001, Nett & Schemmer 2004) aims to ensure reliable co-operation of mobile autonomous entities. Their solution provides an event service that delivers the global state of the system to all entities every time an event is delivered. Entities can then use a local scheduling function to schedule shared resources. This work, however, is limited to infrastructure-based networks. Furthermore, reliability of this service is guaranteed only under application-specific assumptions. Similarly, some work deals with coordination of entities by providing entities with the same view of their environment (Goldberg et al. 2002, Clark 2004), but does not address the possibility of communication failures.

1.3.3 Other approaches

We have identified two main approaches to coordination that do not rely on consensus. Coordination without communication, also called reactive coordination (Gervasi & Prencipe 2004, Ijspeert et al. 2001, Schermerhorn & Scheutz 2006), is based on entities reacting to the behaviour of other entities to coordinate their actions. The approach, however, relies on entities accurately sensing the behaviour of other entities and, as sensor information is unreliable, can only offer best-effort guarantees.

GEAR (Veríssimo & Casimiro 2003) is an architecture for event-driven support of real-time entities based on the Timely Computing Base (TCB) component. This work takes the unreliability of real-time communication into account and is based on the idea that entities are informed in real-time of communication failures, and can react to them. The way in which entities should react in case of a communication failure, however, is not investigated.

1.3.4 Analysis

The majority of approaches to the coordination of autonomous mobile entities are built on the notion of consensus between entities, either on their view of their environment, or on the action they should take. Entities operating in mobile settings and under strong real-time constraints, however, might not be able to communicate with each to reach a consensus on their collective behaviour or even consult each other. Therefore, consensus-based approaches can only offer best-effort guarantees and are not suitable for applications exhibiting strong reliability and timeliness requirements.

Amongst the other approaches to coordination, the TCB approach is particularly relevant, as it takes into account the unreliability of communication and offers strong reliability under real-time constraints. The semantics and application-development support offered, however, are very limited. In contrast, our work provides a systematic process to develop applications composed of autonomous mobile entities.

1.4 Challenges

As outlined above, the limitations of direct communication and sensor data imply that entities might not be able to communicate with each other and need to act independently. So entities need to coordinate their behaviour, i.e., ensure system-wide safety constraints while taking decisions independently, using only limited information. In this section, we review the main challenges raised by this problem.

We have identified four factors that make the problem of coordinating entities particularly challenging in the presence of limited information about each other: spontaneous interaction, race conditions, reaction compatibility and ambiguity in the absence of messages. We detail each of these in turn, and illustrate them by means of the example of an unsignalised junction. For ease of explanation, we assume that the safety constraints in this example state that there can be only one vehicle in the junction at any given time.

1.4.1 Spontaneous interactions

Autonomous mobile entities in our everyday environment have typically been designed by different developers, at different times, for different purposes. This means that entities share their environment with some entities that may not even have existed at the time at which they were deployed. Therefore, entities operate amongst a potentially varying number of others entities, whose type is not known in advance. Entities might need to interact with such other entities, or at least to coordinate their behaviour with theirs. For this purpose, entities need to be able to interact spontaneously with other entities in their vicinity.

For example, autonomous vehicles might be designed to negotiate an unsignalised junction. The number or identity of the vehicles that a given vehicle will encounter during its lifetime are not known in advance. Furthermore, entities of different types can also operate in or around the junction, for example, emergency vehicles, trucks, or buses.

In addition, entities can be spread over a wide space, possibly city or nation-wide. This implies that it is impossible for an entity to build, and rely on, a complete view of the system, i.e., of all the other entities and its environment.

1.4.2 Race conditions

Because entities are autonomous and interact spontaneously, it is not possible to set an a priori order on their actions, and they might do things simultaneously. Simultaneous actions might lead to safety constraints being violated.

For example, two autonomous vehicles can arrive at a junction at the same time, both establish that there is nobody in the junction, and subsequently enter the junction at the same time, therefore violating the safety constraints.

Race conditions make it impossible to consider a flow of actions of an entity in isolation, and require that possible concurrent actions of other entities in the vicinity be taken into account in ensuring the safety constraints.

1.4.3 Reaction compatibility

When it detects that a safety constraint risks being violated, an entity might change its behaviour so that it does not happen. If two entities, however, change their behaviour in a way that is not compatible, the safety constraint might still be violated.

Consider, for example, two autonomous vehicles having detected that they have arrived in the junction vicinity at the same time, and that want to avoid entering the junction simultaneously. A possible way to adapt their behaviour so that the safety constraint is not violated would be, for example, to delay entering the junction for a duration long enough so that the other vehicle will have time to cross. If both vehicles, however, chose to adapt their behaviour and wait for the same time,

they are still likely to enter the junction at the same time.

Reaction incompatibility makes it impossible for entities to react in isolation, they need to ensure that their reaction will be compatible with that of other entities, even though they might not be able to communicate with them.

1.4.4 Ambiguity in the absence of messages

Because entities are mobile and their interactions are spontaneous, an entity does not know in advance whether there are any other entities in its surroundings. In addition, when communication is unreliable, an entity might not be able to send, or reply to, a message (via direct or indirect communication). This means that it is impossible, a priori, to distinguish between deficient communication and the absence of other entities.

This challenge arises, for example, when an autonomous vehicle arrives at an unsignalised junction. Such a vehicle needs to know whether there are any other vehicles on, or about to enter, the junction, so that it can cross it safely. For this purpose, it can send a message over the junction, requesting any entity present in the junction to reply to its message. As communication is unreliable, however, a car in the junction might not get the message, or might not be able to reply to it. Therefore, in this example, the ambiguity of the absence of messages, means that a vehicle cannot know whether there are any vehicles in the junction.

If entities have to ensure system-wide safety constraints and none of them can know for sure whether there are other entities in its vicinity, they cannot safely make progress. Therefore, ambiguity in the absence of messages makes it impossible for entities to progress safely if they do not know about the current state of communication.

1.5 Approach

It has been shown that distributed consensus is not solvable in the presence of arbitrary numbers of communication failures (Gray 1978, Lynch 1996). Furthermore, consensus protocols typically rely on knowing the identity of the participants, and then several exchanges of messages (3 for example, in the 3-phase commit protocol (Skeen & Stonebraker 1983)). These steps, however, take a significant time, and therefore the timeliness requirements of the class of application catered for imply that it might not always be possible to reach consensus even when communication is sufficient. This implies that existing consensus-based coordination methods that rely on continuous real-time connectivity cannot be applied.

Instead of relying on continuous connectivity between entities, our approach is centred on the idea that entities need to adapt their behaviour depending on currently available information. This allows entities to make progress when it is safe to do so, while making sure that their safety will never be compromised. This approach relies on entities being able to quantify the information currently

available to them, via both data communication and sensing.

An entity can evaluate its currently available information via direct communication from the messages it receives, but it also needs some feedback about how well it can currently communicate with other entities. The most-often-considered form of feedback on the state of communication is the identity of entities with which an entity can currently communicate, as in group-based communication (see, for example, Friedman 2003, Singh et al. 2006). Alternatively, the TCB model (Veríssimo & Casimiro 2002), mentioned in 1.3.3, uses currently achievable message latency as the metric to quantify the state of communication. In this work, however, we chose to use the space-elastic model (Hughes 2006) that uses the geographical proximity in which entities can currently communicate as a metric for the state of current communication. This model is particularly relevant for mobile entities, as the use of a geographical metric makes it easy to derive constraints depending on the movement of entities.

Quality of sensing can be quantified in terms of the number, type and accuracy of sensors available, as well as their current sensing range. The contributions of this thesis include a model for sensor data that, similarly to the space-elastic model, allows developers to reason about the geographical proximities in which sensor data is available, and therefore in which indirect communication is currently possible.

Comhordú, the coordination model presented in this thesis, shows how such feedback on the current state of communication can be exploited to ensure that entities can make progress when possible, while remaining safe, despite having access only to unreliable information.

1.6 Goal and contributions

As motivated by Section 1.1, the goal of this thesis is to support the development of autonomous mobile entities able to ensure system-wide safety constraints while having access only to unreliable information. This requires the ability to design fully decentralised solutions, i.e., solutions where entities take decisions independently of any central coordinating entity. Moreover, this work aims to be generic, i.e., suitable for a wide range of autonomous mobile entities.

The contributions of this thesis are fourfold:

1. a space-elastic sensor and indirect communication model, that is the equivalent for sensor and indirect communication of the space-elastic model for wireless communication;

2. a coordination model, Comhordú, that builds on both the space-elastic model and the sensor and indirect communication model to allow entities to adapt their behaviour depending on available information;

3. a systematic process that allows developers to use Comhordú to program entities by translating system-wide safety constraints into a set of requirements on the behaviour of individual entities.

If these requirements, expressed as conditions for safely performing certain actions, are met, the safety constraints will be ensured;

4. a development tool, ComhorMod, that supports the development process of autonomous mobile entities by automating the systematic steps of Comhordú.

Indirect interaction is recognised as a promising research direction, but no multiagent system model has yet been defined (Keil & Goldin 2005). Therefore, an indirect contribution of this thesis is to demonstrate how indirect communication can be used in conjunction with direct communication in the design of autonomous systems.

1.7 Scope

This thesis defines a real-time coordination model for autonomous mobile entities. This work also describes a systematic process to use the model for the design of applications composed of autonomous mobile entities. To ease this process, a tool is presented that guides entity developers through the different steps required to use the model, and automates the systematic steps. Finally, the generality of the model is evaluated through its application to several scenarios and the behaviour of the generated applications is assessed via simulations. This work, however, does not present a formal definition of the coordination model, nor a formal proof of its correctness.

The development process allows system-wide safety constraints to be translated into requirements on the behaviour of individual entities. If these requirements are met, the safety constraints will be ensured. Ensuring these requirements might require entities to use specific hardware as well as specific architectures or algorithms (e.g., CPU scheduling algorithms). How to ensure that entities fulfil these requirements, however, is outside the scope of this work.

The definition and use of the model is illustrated with a number of examples from the intelligent transportation systems domain (ITS). ITS, however, is not the subject of this thesis, and is only one of the possible application domains of this work. Nevertheless, it is worth mentioning that the model has been designed for the coordination of physical mobile entities, not software agents. The constructs defined facilitate reasoning about mobility of physical entities and would not be efficient for software agents.

Note also that while this work can be used to design entities that can support humans, the entities are assumed to be autonomous and their interactions with humans are not covered by this work. One consequence of this assumption is that the actuation of entities is assumed to be potentially safety critical, i.e., best effort is not a sufficient approach. In addition, the real-time aspect of the coordination is crucial for such entities, and this is also a defining assumption for our work.

Finally, it should be noted that the issues of trust and security (as opposed to safety) are not considered in this thesis.

1.8 Road map

The remainder of this thesis is organised as follows. Chapter 2 presents a review of the state of the art on real-time coordination of autonomous mobile entities. Chapter 3 presents the models on which Comhordú relies. These are an environment model on which our work is built, a direct communication model called the space-elastic model, and our sensor and indirect communication model. Chapter 4 presents the coordination model, Comhordú and Chapter 5 shows how it can be used to design autonomous mobile entities. ComhorMod, a tool supporting the design process of such entities, is presented in Chapter 6. The generality of Comhordú as well as the quality of the outputs generated using the design process are evaluated in Chapter 7. Finally, Chapter 8 concludes this thesis and outlines possible future work.

1.9 Summary

This chapter outlined the goals and scope of the work described in this thesis – essentially, the definition of a real-time coordination model that allows autonomous mobile entities to operate safely despite unreliable communication and limited sensor information. The chapter began by presenting the basic motivation for the work described in this thesis, i.e., the need for autonomous mobile entities to coordinate their behaviour to ensure system-wide safety constraints, despite only having access to limited information. The problem was defined in more detail by examining the limitations of both direct and environment-mediated communication. The main challenges that arise from this problem were outlined, and a brief overview of existing work demonstrated that all of these challenges have never been tackled simultaneously. Finally, the chapter concluded by detailing the approach of this work, and the goals and contributions of this thesis, as well as the areas that are outside the scope of the work.

Chapter 2

Related Work

This thesis deals with the real-time coordination of autonomous mobile entities. The terms coordination, collaboration and cooperation are used loosely to refer to the idea of people or systems working together. In our view coordination differs from collaboration or cooperation in that collaborating or cooperating entities typically have the same goal, while entities that have different goals may also need to coordinate their behaviour to ensure properties, such as fairness or, in our work, safety.

As explained in the introductory chapter, the problem that this thesis addresses is the provision of a generic model for the coordination of autonomous, mobile, physical entities. Two crucial requirements of this work are high reliability to deal with safety-critical applications, and timeliness as entities interacting with the physical environment must do so in real-time. In particular, the safety constraints must be met despite potential communication faults, and the entities must adapt their behaviour within known time-bounds.

The concept of coordination has been studied in a variety of domains, which include biology, computer science, organisation theory, operations research, economics, linguistics and psychology (Malone & Crowston 1994). In particular, coordination has been studied in several computer science domains, but this work has mostly not been integrated together. In this chapter we review work on coordination from different communities, focusing on reliability and timeliness. The model-driven engineering (MDE) community, similarly to the work described in this thesis, uses high-level models to specify a system and aims to automatically generate implementations from such models. To our knowledge, however, no work has been done in this community on the real-time coordination of entities, therefore, this area is not surveyed in this section. Section 2.1 explores related work in the multi-agent systems (MAS) community, and introduces environment-mediated communication. Work from the robotics community, both application-specific and generic, is studied in Section 2.2. Work from the intelligent transportation systems (ITS) community, typically inspired by one of the two previously-cited communities, is described in Section 2.3. Section 2.4 describes work intended to formalise generic coordination mechanisms as coordination models. Work from the real-time community that is relevant

to this thesis is investigated in Section 2.5. To conclude, Section 2.6 evaluates all the mechanisms and models presented in the previous sections and compares them. Finally, Section 2.7 summarises this chapter.

2.1 Multi-agent systems

The MAS community studies computer systems comprised of autonomous, intelligent, communicating systems (or agents). This work is also often referred to as Distributed Artificial Intelligence (DAI). MAS have been used to solve a wide range of problems in domains such as artificial intelligence, distributed systems, robotics and artificial life (Ferber 1999). MAS have introduced the notion of collective intelligence, and focus on how agents can coordinate their behaviour and collaborate to reach goals that may be difficult to achieve by an individual agent or monolithic system (Ranjit 2007). In this section, we first explain the limitations of work from the MAS community to solve the problems addressed in this thesis, and then describe in more detail two concepts from the MAS community that are particularly relevant to our work: commitments and environment-mediated communication.

2.1.1 Physical vs. software agents

While some definitions of MAS encompass the possibility of agents with some physical embodiment, most of the research focuses on software agents and aims to optimise system performance. Therefore, the problems tackled typically do not match the ones studied in this thesis, as software agents do not interact with the physical environment, and therefore often do not require timeliness or reliability guarantees. Furthermore, software agents can typically communicate reliably with each other, either via interprocess communication or wired communication, and therefore potentially have access to complete, timely, and reliable information about the system. Finally, software agents do not have to deal with robot kinematics, i.e., the limitations imposed by the need to actuate on the physical environment.

A few projects in the MAS community deal with physical entities, such as work on the decentralised control of Automatic Guided Vehicles (AGVs) (Weyns, Schelfthout & Holvoet 2005, Weyns & Holvoet 2008). This work applies concepts from a branch of MAS called situated MAS, in which agents have an explicit position in the environment (Ferber 1999), to design the software. Each AGV is controlled by an AGV agent, while transport agents are used to represent loads to be handled by the AGVs. The situated MAS approach, motivated by a need for greater flexibility in AGV behaviour and the need to cater for a dynamic AGV population, is shown to be feasible and efficient in terms of bandwidth usage. This work, however, does not consider the timeliness and reliability of communication. While these characteristics are not really problematic in confined industrial settings, they represent significant challenges in generic settings (Gaertner & Cahill 2004). The work of Weyns, Schelfthout & Holvoet is representative of the work in the MAS community, where the emphasis is on higher-level problems

and relies on assumptions about lower-level characteristics such as the reliability of communication.

As MAS have been applied to a wide range of problems, however, this body of research offers interesting inspiration for work with physical agents. Of particular interest for the work described in this thesis are the notion of commitments as well as the use of indirect communication and stigmergy.

2.1.2 Commitments

A number of projects in the MAS community have tried to formalise the underlying theories and principles that govern coordination and cooperation. In particular, it has been argued that *commitments* (pledges to undertake a specified course of action) and *conventions* (means of monitoring commitments in changing circumstances) are the foundations of coordination in multi-agent systems (Jennings 1993). Agents can make commitments both about actions and beliefs, and these commitments can either be about the past or the future. Because their circumstances might change, agents might want to revoke their commitments. Conventions describe circumstances under which an agent should reconsider its commitments and indicate the appropriate course of action to either retain, rectify or abandon the commitment. The concept of commitment has been used to structure numerous projects in the MAS community, for example, for agents characterised as having Beliefs, Desires and Intentions, called BDI agents (Fasli 2001). Similarly, the notion of a situated commitment, based on the roles of the involved agents and the local context in which they are placed, has been characterised to enable explicit collaborations between situated agents (Weyns et al. 2004) .

2.1.3 Environment-mediated communication

Situated agents typically share a common environment, and adapt their behaviour depending on the state of the environment. Therefore, when an agent changes something in its environment, this change can be detected by another agent sensing the environment. In this way, the behaviour of one agent can be influenced by the behaviour of another. This allows both *coordination without communication*, where agents coordinate their behaviour by reacting to the behaviour of other agents, and communication via the environment, also termed *environment-mediated communication* (EMC), where agents leave signs in the environment, that they themselves and other agents can sense. Note that the distinction between these two concepts is not sharp, as leaving signs might be part of agents' behaviour; this is discussed in more detail in Chapter 3.

Coordination without communication has been used to solve a large number of problems such as flocking (Gervasi & Prencipe 2004), large-scale assembly (Ijspeert et al. 2001), and territory exploration (Schermerhorn & Scheutz 2006).

EMC is the basis of stigmergy. The term stigmergy was first coined by a French entomologist studying the nest building behaviour of termites (Grassé 1959), and it is the extremely complex coordinated behaviour achieved by such insects that remains the classical example of stigmergy in the real world. In stigmergic interaction, an agent's actions leave signs in the environment, that it and

other agents sense and that determine their subsequent actions (Parunak 2003). There does not seem to be a consensus on the etymology of the word stigmergy or its precise meaning. Beckers et al. (1994) suggest that the origin of the word stigmergy is derived from the roots stigma 'goad' and ergon 'work', thus implying a sense of incitement to work by the products of work. In this view, EMC is a means used by stigmergy. However, Parunak (2003) provides an alternative derivation and suggests that the term is formed from the Greek words stigma 'sign' and ergon 'action' and so therefore captures the notion of an entity's actions leaving signs in the environment that influence the subsequent actions of other entities. In this second definition, stigmergy is the same as EMC. Stigmergy has been applied successfully to solve many MAS problems such as combinatorial optimisation (Dorigo et al. 1999), as well as in other domains of computer science such as load balancing and routing in communication networks (Schoonderwoerd et al. 1996, Caro & Dorigo 1998), peer-to-peer application design (Babaoglu et al. 2002), and coordination in robotics (Holland & Melhuish 1999).

Keil & Goldin (2005) have formalised the notion of *indirect interaction*, as being interaction via persistent, observable changes to a common environment, where recipients are any agents that observe these changes. Therefore indirect interaction encompasses both coordination without communication, and EMC. It is argued that, as message-passing cannot adequately model multi-agent interaction which includes both direct and indirect interaction, models that allow indirect interaction as well as direct interaction are more expressive than models that do not (Keil & Goldin 2005).

2.1.4 Analysis

While the problems addressed by the MAS community are close to the ones tackled by this thesis, the work in this community deals mostly with software agents and its emphasis is on high-level problems, i.e., the provision of appropriate programming models that assume that characteristics such as the reliability of communication are provided at lower system levels. Therefore, this work is not directly applicable to work on physical autonomous entities, where interactions with the environment introduce a number of other issues to be overcome to ensure reliability and timeliness requirements, such as limited accuracy of sensing and actuation.

The MAS community, however, is fairly mature, and its work is a source of inspiration for many other research domains. Two notions formalised by the MAS community are of particular interest for this thesis: commitments and EMC. Commitment is an essential concept, as capturing pledges of what agents will do allows prediction of what is going to happen. This notion inspired the concept of contracts between entities in Comhordú. Conventions describe when agents can withdraw their commitments. When dealing with safety, however, commitments should not be revoked. Instead, in Comhordú, entities can transfer their responsibility to another entity (see Chapter 4); this is equivalent to transferring a commitment.

Another significant contribution of the MAS community is the formalisation of coordination without communication and EMC, as indirect interactions. Comhordú uses both coordination without

communication and EMC.

2.2 Multi-robot systems

In the robotics community, a number of projects have investigated the coordination of autonomous robots. While the majority of this work on multi-robot systems (MRS) relates to static robots, between which continuous real-time connectivity is assumed, a number of applications have been built using mobile robots. These applications span a wide range of application domains, for example, manufacturing (Cawkwell 2000, Simmons et al. 2000), space exploration (Goldberg et al. 2002), defence (Parker 2003, Konolige et al. 2004), and search and rescue operations (Hirose & Fukushima 2002).

This section first presents work that addresses only the coordination of a specific aspect of robots behaviour, and then reviews MRS projects that provide generic coordination mechanisms.

2.2.1 Application-specific solutions

A number of projects deal with the coordination of specific aspects of robot behaviour, or for undertaking a specific task. Examples of the former include collision avoidance (Guo & Parker 2002), while examples of the latter include terrain exploration (Rekleitis et al. 2001), multi-target observation (Beard et al. 2002), and large object manipulation (Simmons et al. 2000).

Work on collision avoidance is particularly relevant to this work because of the strong reliability and timeliness requirements of this task. Approaches to motion planning can be either *deliberative* or *reactive*. In the first case, robots cope with environmental changes by adopting a strategy to reorganise the behaviour of the entire team. In the second case, each robot copes with environmental changes independently (Iocchi et al. 2001). Simple reactive motion planning strategies cannot guarantee deadlock-freedom and convergence even in simple cases (Guo & Parker 2002). Deliberative motion planning for robots in a dynamic environment with moving obstacles is a hard problem. Even for a simple case in two dimensions, the problem is NP-hard and is not solvable in polynomial time (Fujimura 1992). Because a deliberative solution to this problem is computationally intensive, a number of algorithms have been designed to solve it a priori, i.e., before the robots move (see, for example, Buckley 1989, Warren 1990). This design choice, however, requires that the characteristics of the environment be known in advance, which precludes their use in dynamic environments. Online solutions typically consider a simplified version of the problem, for example, by decomposing the problem into path planning and velocity planning: a path for each robot is first derived, then a velocity profile that avoids collisions is computed for each path. Amongst the projects that offer an online solution two categories of solutions emerge: *centralised* and *decentralised*. Centralised solutions allow completeness and global optimisation (Clark et al. 2003), but rely on all information being centralised at a single place, which is costly in terms of communication, constitutes a single point of failure, and is not scalable. In distributed solutions, robots devise a common reaction but the decision is taken

collaboratively by the robots. Distributed solutions yield less efficient solutions because only local information is available, and are more prone to deadlock.

A particularly interesting solution is presented by Clark (2004). This solution, built on a coordination platform called Dynamic Robot Networks, aims to combine the advantages of both centralised and decentralised systems, by letting robots form ad hoc networks and run a centralised coordination protocol on each network. The coordination platform, which takes into account both sensor and communication limitations, is reviewed in more details in 2.2.2.3.

While the previously mentioned bodies of work acknowledge the need to provide real-time guarantees, their proposals limit the real-time concerns to proposing fast algorithms, that enable online computation but do not give timing guarantees. A notable exception is the work of Yared et al. (2007) that acknowledges that real-time communication in ad hoc networks is challenging. To tackle the fact that communication delays in wireless networks are unbounded, they offer a time-free solution to the collision avoidance problem, based on path reservation, i.e., where the environment is divided in zones that robots can request, own and then release and where a robot must own a zone before travelling over it. The solution is time-free because a robot waits until it has received an answer from all its neighbours before entering a zone. Their solution reduces the problem scope by exploiting the locality of the collision avoidance problem, an approach that is also used in the work described in this thesis. The safety guarantees provided, however, are based on two assumptions: synchronous neighbourhood discovery, i.e., the ability for a robot to discover, within a given time bound, all other robots within one communication hop, and a fixed communication range. Both assumptions are fairly restrictive, and the second in particular is unlikely to be met in environments where obstacles can create zones in which communication is impossible.

2.2.2 Generic solutions

While a large proportion of the work in the robotics community deals with specific (aspects of) multi-robot systems, the goal of this thesis is to provide a generic solution to the coordination of autonomous mobile entities. In this section, we review the principal generic approaches to robot coordination. Because of the limitations of wireless communication and sensing outlined in Chapter 1, centralised solutions requiring all the necessary information to be available at a single place are ignored, and only decentralised solutions are detailed.

2.2.2.1 Multi-robot task allocation

ALLIANCE (Parker 1998) is a framework for coordinating MRS composed of heterogeneous behaviour-based robots. The aim of the framework is to allow robots to individually select appropriate actions throughout their mission, based on the requirements of the mission, the activities of other robots, the current environmental conditions, and the robots' own internal states, in order to maximise the reliability of the team, i.e., the likelihood that the mission will be fulfilled. The actions that robots

must undertake are assumed to be independent, so coordinating the robots is limited to coordinating which robot is performing which task. Robots have sets of behaviours, which are controlled by modules called motivational behaviours, that can cross inhibit each other, i.e., the activation of one behaviour set suppresses the activation of all other behaviour sets. Motivational behaviours depend on two parameters *impatience*, which enables a robot to handle situations when other robots fail in performing the task, and *acquiescence*, which enables a robot to handle situations, in which it, itself, fails to properly perform its task. These concepts allow adaptive, fault-tolerant task allocation.

The reliability of this framework in terms of fulfilling a mission has been formally analysed (Parker 1998). Use of the framework has been demonstrated on a variety of proof-of-concept applications including hazardous waste cleanup, a cooperative box pushing demonstration (Parker 1994), and multi-target observation (Parker 1999). While this work takes into account the unreliability of sensing, communication, and the robots themselves, by allowing a robot to change its action when it senses that it or other robots fail to perform a task, coordination in this framework is limited to selecting independent subtasks.

A number of other works are concerned with efficient and reliable task allocation in MRS (see, for example, Østergaard et al. 2001). This problem has been formalised and shown to be an instance of the well-known optimal assignment problem (OAP) (Gerkey & Mataric 2003). All this work, however, limits the problem of coordination to task allocation to fulfil a high-level mission, therefore tight coordination is not supported and timeliness of the solution is not taken into account.

2.2.2.2 A distributed layered architecture for mobile robot coordination

To enable coordination between multiple mobile robots, Goldberg et al. (2002) extend the traditional three-layered approach adopted for many single-agent autonomous systems. This three-layered approach provides event handling at different levels of abstraction through the use of behavioural, executive and planning layers. The planning layer decides how to achieve high-level goals and sends plans to the executive layer. The executive layer decomposes plans into tasks, sequences the tasks and monitors their execution. Finally, the behavioural layer interfaces to the robot's sensors and actuators, by controlling the robot and sending back sensor data and status information to the higher layers. The proposed extension allows robots to interact at each layer, hence allowing (1) the construction and sharing of plans between multiple robots, (2) the establishment and maintenance of executive-level, inter-robot synchronisation constraints and (3) the establishment of distributed behaviour-level loops. This architecture is outlined in Figure 2.1. This approach has been tested on both large-scale structure assembly (Simmons et al. 2000) and a Mars exploration scenario (Goldberg et al. 2002).

While this work allows tight coordination between robots, it assumes high-bandwidth, low-latency communication between robots to achieve good performance in interacting, multi-robot behaviours.

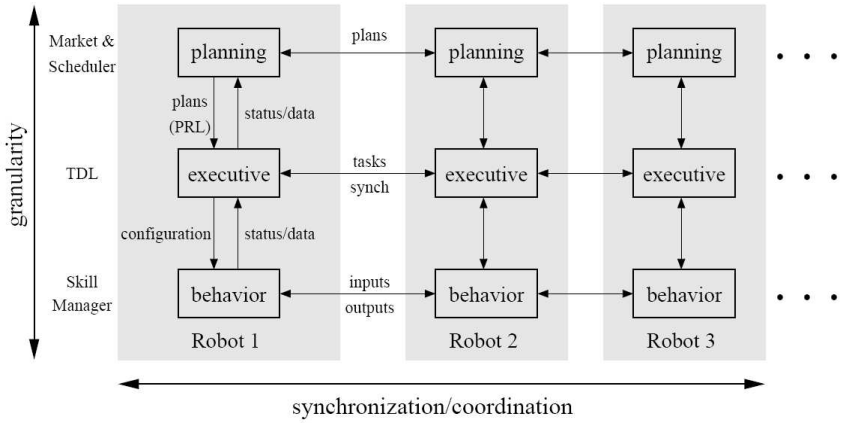


Figure 2.1: Layered multi-robot architecture (Goldberg et al. 2002).

2.2.2.3 Dynamic robot networks

Dynamic Robot Networks (Clark et al. 2003, Clark 2004) is a platform for the coordination of autonomous mobile robots that takes into account the fact that sensing and inter-robot communication are limited. It is based on the idea that robots dynamically form ad hoc networks as they move, and provides a common world view to all entities in that network, hence allowing centralised coordination on each network. This work therefore aims to maximise the efficiency of the planning by making use of available local information to optimise the behaviour of robots. The unreliability of sensing and communication is modelled as a range within which they are available. This approach is very close to the models used in this work, but no analysis of the required communication and sensing range is presented. Furthermore, the lack of a feedback mechanism to know the areas in which sensing and communication are currently available implies that these are assumed constant, an assumption unlikely to be met in the presence of obstacles in the environment.

In addition, timeliness guarantees are not evaluated. Furthermore, while this work is presented as being a general approach for the coordination of autonomous mobile entities, the only application detailed is motion planning. No detailed information is available about how this work could be used for other coordination problems.

2.2.3 Analysis

This section reviewed existing work on MRS. A significant proportion of this work assumes reliable connectivity, which might be reasonable in the confined and protected environments in which robots are typically deployed during experimentation, but would not be for large-scale applications in everyday environments. In contrast, the work in this thesis caters for the coordination of autonomous mobile entities even in the presence of communication failures.

A number of projects in the robotic community either assume that team membership is fixed and

known, and allow tight interaction between team members, or assume that the systems are unaware of each other, i.e, that each robot executes its own task without any knowledge about the other members of the team, in which case communication is only indirect and the coordination is only best effort (Simmons et al. 2000, Farinelli et al. 2004). In contrast, this work aims to give strong guarantees for open systems, i.e., systems where new active agents may dynamically join and later leave (Ciancarini 1996).

Another interesting observation is that real-time concerns are limited: coordination algorithms are often designed to be fast, but none of the generic solutions mentions the timeliness of communication, even though it is a significant challenge in wireless networks. Furthermore, while a number of projects provide generic infrastructure for the coordination of MRS, no coordination model has, to date, been explicitly designed for robotics (Farinelli et al. 2004). This is what the work described in this thesis aims to achieve.

2.3 Intelligent transportation systems

The intelligent transportation systems (ITS) community investigates how computers, information and communication technologies can be applied to improve transportation infrastructure and vehicles (Zhao 1997). Early achievements for intelligent vehicle systems were in the domain of driver assistance: automatic reverse parking (e.g., Paromtchik & Laugier 1996), adaptive cruise control (ACC) (e.g., Ioannou & Stefanovic 2005), and driver information systems (e.g., Nadeem et al. 2004, Ueki et al. 2004, Karam et al. 2006). More recently, some work has focused on fully autonomous vehicles. This requires a higher level of reliability as the system is completely responsible for the safety of the passengers. This characteristic makes this problem particularly relevant to the work described in this thesis.

2.3.1 Autonomous cars

In the last decade, the idea of driverless, autonomous, vehicles has moved from the domain of pure science fiction, to a vision that should be achievable in the not-too-distant future (Baber et al. 2005). Research in this area includes the search for adequate sensors and actuators (Aufrere et al. 2003), vehicle control algorithms (Kato et al. 2002), and assessment of the usability of such vehicles (Parents & Gallais 2002). This has led to a number of results, including a practical demonstration of driverless vehicles following a road lane, overtaking a slower vehicle, and crossing an unsignalised junction (Kolodko & Vlacic 2003, Baber et al. 2005).

A good benchmark for work in the area are the challenges organised by the Defense Advanced Research Projects Agency (DARPA), an agency of the United States Department of Defense, responsible for the development of new technology for use by the military. In 2005, four of the competing autonomous vehicles successfully completed a 132-mile (212 km) desert route for the DARPA grand

challenge[1]. This shows that a number of solutions exist for autonomous cars in static environments. Cars competing in the DARPA urban challenge, in November 2007, are expected to complete a 60-mile (96 km) urban area course while obeying all traffic regulations, negotiating other traffic and obstacles, and merging into traffic. Significant competitors in the challenge include the Stanford racing team[2], the Tartan racing team from Carnegie Mellon University[3], and the MIT team[4]. While for this challenge, cars will have to be able to avoid each other and other mobile obstacles, cars do not communicate with each other and therefore their coordination relies only on sensor information.

2.3.2 Collaborative driving

The next significant paradigm in ITS evolution is the notion of collaborative driving. By collaborating with each other, autonomous vehicles are expected to achieve improved safety and improved efficiency as well as better passenger comfort (Parents & Gallais 2002, Kolodko & Vlacic 2003). Some autonomous vehicle architectures already include the possibility for vehicles to cooperate (Kolodko & Vlacic 2003, Naranjo et al. 2006). The support offered for coordination, however, is not detailed, or is limited to exchanging position information. Furthermore, communication is assumed to be reliable. Specific aspects of collaboration have been researched in more detail, however, in particular platooning and intersection management.

Note that while there is a significant body of work on achieving collaborative driving by installing infrastructure on roads and using vehicle-to-roadside communication, with technologies such as Dedicated Short Range Communication (DSRC) (Federal Communications Commission 1999), this section focuses mainly on car-to-car approaches as these are closer to the problems addressed by this thesis.

2.3.2.1 Platooning

A significant body of work has looked at how to achieve platoons of autonomous vehicles, i.e., groups of vehicles whose actions on the road are coordinated by means of communication (Varaiya 1993). Platoons are expected to enable increased road capacity and efficiency, reduced congestion, energy consumption and pollution, and enhanced safety and comfort (Michaud et al. 2006).

Research efforts have mainly focused on low-level control (e.g., use of steering, throttle and brake for lateral and longitudinal control), sensor issues (e.g., the suitability of different types of sensors, achievable accuracy), string stability (i.e., the attenuation of spacing errors as they propagate upstream in a platoon), minimal spacing (to ensure safety while optimising traffic flow), and on demonstrating the feasibility of cooperative driving scenarios in limited and controlled conditions (Michaud et al. 2006). Group communication and coordinated manoeuvring have only received minor attention due

[1] http://www.darpa.mil/grandchallenge
[2] http://cs.stanford.edu/group/roadrunner/
[3] http://www.tartanracing.org/
[4] http://grandchallenge.mit.edu/

to the lack of reliability of existing sensing and communication devices, and the safety risks of testing such aspects with real vehicles. The application of techniques such as Team Oriented Programming (TOP) from the MAS community has been investigated to solve these problems (Hallé et al. 2004, Hallé & Chaib-draa 2005). In this approach, platoon members are assigned roles within a team hierarchy, and tasks relating to these roles are defined. This strategy is shown to be efficient both in terms of vehicle reactivity and number of messages exchanged. This work, however, has been tested only in a simulator, where neither communication and sensor unreliability nor timeliness issues have been taken into account.

Of particular interest to the work described in this thesis is work on assessing different coordination strategies between cars in a platoon, and required communication for different manoeuvres (Michaud et al. 2006). In particular, this work describes the different roles that cars in a platoon can have in a manoeuvre and identifies a number of strategies in terms of whether cars in a given role can communicate to cars in other roles, and whether these other cars can provide feedback to the initial cars.

Different combinations of sensors, architectures and control strategies have also been tested, and these efforts culminated in a number of demonstrations of cars travelling in platoons (Kato et al. 2002, Empey 2002). In these demonstrations, however, the number of cars in a platoon is limited, the settings are protected, and only simple manoeuvres are performed. Overall, while platooning of autonomous vehicles is a very active research area, these efforts have not yet yielded a generic solution to the problem that caters for a wide range of manoeuvres amongst many cars in normal road conditions where communication and sensor information are limited.

2.3.2.2 Intersection management

Another body of work in ITS deals with collaborative collision avoidance, and more particularly collaborative intersection crossing. Techniques from the MAS community have been applied to this problem, by having a "driver" agent control each autonomous car, and an intersection manager control each intersection (Dresner & Stone 2005). The driver agent of a vehicle approaching a junction requests and receives slots from the intersection manager, during which the vehicle may pass. The system is centralised, in the sense that each intersection manager coordinates the motion of all vehicles in its vicinity. This reservation-based system is implemented in a simulator where it is shown to improve both delay on vehicle journey and throughput of the intersection in comparison to traditional systems such as traffic lights or stop signs (Dresner & Stone 2004). Interactions between driver and intersection manager agents obey a protocol that has been designed so that vehicle safety is not compromised by message loss. The model, however, does not take into account the timeliness of communication and vehicles behaviour, therefore safety can be compromised if vehicles do not get a reservation when they expect or spend more time than planned in the intersection.

Another proposal for intersection management has been evaluated in an experimental testbed using

robots (Sheng et al. 2006). This work offers a fully distributed algorithm, where vehicles broadcast their intended path. If the paths of some vehicles meet, they coordinate their velocities so that they will not collide, by applying an algorithm on all their paths. Therefore, this work requires a consensus amongst all vehicles whose paths meet. While this work takes into account that the range of communication is limited, this range is assumed to be constant, which is not a realistic assumption for environments where obstacles can obstruct wireless communication. Furthermore, real-time aspects are not taken into account.

Finally, in the Cybercar project, an algorithm for collaborative intersection crossing that relies on refining partial trajectories is demonstrated on real vehicles (Bouraoui et al. 2006). This algorithm, however, relies on vehicles having access to an accurate view of their environment and its evolution over a given time period. While it is suggested that this could be achieved by vehicles and obstacles exchanging information over an ad hoc network, the timeliness of the information exchange is not taken into account in the model. Furthermore, how vehicles should react when they have detected a potential collision is not investigated.

2.3.3 Analysis

Autonomous cars are becoming a reality, and collaborative driving is a promising research area. Results in this domain, however, are still limited. Because the problems involved are very complex, simplified versions are often addressed. A significant body of work is inspired by work in the MAS community and typically offers fairly elaborate coordination means. While the models of some of this work take car kinematics into account, they typically do not take communication unreliability and timeliness issues into account.

Another significant body of work on collaborative driving draws inspiration from the robotics community. The use of robots has been advocated to evaluate solutions with real world constraints such as limited perception, imprecise actions, latency, real-time decision making, embedded computing, unanticipated events, etc (Michaud et al. 2006). Contrary to simulators, however, robots cannot model real vehicle dynamics. Furthermore, robots typically operate in controlled conditions, where their communication and sensing can be assumed to be reliable. In addition, work in this area typically offers limited coordination mechanisms.

Finally, work on the coordination of autonomous vehicles is typically very application specific, and addresses only a specific situation in which cars need to coordinate their behaviour.

2.4 Coordination models

Work in the area of coordination models deals with the investigation of generic models, semantics and middleware for coordination. In this section, we review the most significant work in this area related to the problem of real-time coordination of autonomous mobile entities.

Coordination models can be split into two classes: data-centric or message-centric, depending on whether the emphasis is on data sharing or message passing. We review each in turn, with an emphasis on models that are suited to mobile entities.

A coordination model was defined in Chapter 1 as a set of coordinable entities, a set of coordination media, and a set of coordination laws that dictate how entities coordinate themselves through the given coordination media, using a number of coordination primitives. While most of the models presented in this section fit into this definition, some of the message-centric systems (those described in Section 2.4.2.1) offer only communication means and no coordination laws. It can be argued, however, that by communicating entities can have common knowledge of their environment hence enabling coordination. For this reason, such systems are often considered as part of the coordination community and are included in this section.

2.4.1 Data-centric models

Gelernter & Carriero (1992) advocated a clear separation between the interactional and the computational aspects of software components. This consideration has been encapsulated in the design of Linda (Gelernter 1985), originally presented as a set of inter-agent communication primitives that may be added to almost any programming language. This set includes primitives for process creation, as well as for adding, deleting and testing for the presence of data in a shared dataspace. Linda therefore allows decoupled communication between processes: each process only needs to know about the data available, not about the processes producing or consuming it. This characteristic led it to be recognised as a general-purpose coordination paradigm for distributed programming (Cabri et al. 2006). This work served as a basis for the definition of the notion of coordination models, and the tuple-space abstraction, where entities coordinate by manipulating a shared collection of data objects, called *tuples*. Most of the data-centric coordination models use a tuple-space approach. In this section, we first review the data-centric models that take the mobility of entities into account and then discuss work that introduces a notion of time.

2.4.1.1 Mobility

The most influential coordination model supporting mobility is LIME (Linda In a Mobile Environment) (Murphy et al. 2001, 2006), a coordination model and middleware designed for ad hoc networks, inspired by work on the Linda model. LIME caters for physical mobility of hosts and logical mobility of agents (i.e., run-time migration of software components), by having a tuple space attached to each mobile entity. Entities then collaborate by transiently sharing their tuple spaces, creating a "global virtual data structure" (Murphy et al. 2001). This work has been extended to include information coming from the physical environment in addition to application data (Murphy & Picco 2004). LIME has been used for entertainment applications (Murphy et al. 2006), as well as support for human-led space exploration (Murphy & Picco 2004), and data replication (Murphy & Picco 2006).

To facilitate development of context-aware applications, EgoSpaces (Julien & Roman 2002, 2004), one of the many extensions of LIME, introduces the concept of a view that allows nodes to specify from which other nodes tuples are gathered. To offer higher-level coordination support, the concept of views was later extended to include reactions, which consist of actions that are automatically performed in response to specified changes in a view (Julien & Roman 2004). Use of EgoSpaces was demonstrated on an emergency vehicle warning system, a subscription music service, and a collaborative puzzle game (Julien & Roman 2006). Despite the strong reliability requirements of some of these examples, the proposed applications, however, offer only best-effort semantics and in particular do not offer any reliability or timeliness guarantees. TOTA (TUples On The Air) (Mamei & Zambonelli 2004) allows the definition of tuples that are automatically disseminated by copying them to connected nodes according to an application-specific rule. It has been shown that the TOTA middleware can be used to program stigmergic coordination (Mamei & Zambonelli 2005).

Finally, Limone (LIghtly-coordinated MObile NEtwork) (Fok et al. 2004) is another LIME-inspired model, designed for use on devices with potentially limited power and memory over ad hoc networks. As disconnection and message loss are frequent in ad hoc networks, in Limone each agent maintains strict control over its local data and defines an acquaintance policy that governs the agents with which it will interact. All distributed operations include mechanisms such as timeout to prevent deadlock due to packet loss or disconnection. Therefore, Limone supports coordination in unstable networks. This model is demonstrated on a universal remote control application that can control devices in its proximity.

All the work presented in this section offers high-level semantics and is generic, i.e., suitable for many coordination applications. None of this work, however, considers real-time guarantees. Furthermore, because the guarantees provided by tuple-spaces are quite strong and high-level, we believe that it would be particularly hard to provide them reliably for mobile entities under stringent timeliness requirements.

2.4.1.2 Real-time

Recognising a need for real-time coordination, Jacquet et al. (2000) and Jacquet & Linden (2007) have extended the Linda model with the notions of both relative and absolute time. With respect to relative time, they describe two extensions: a delay mechanism to postpone the execution of communication primitives, and explicit deadlines on the validity of tuples and on the duration of suspension of communication operations. For absolute time, they introduce a wait primitive to wait until an absolute point of time, and time intervals, to express both the validity of tuples in the data store and on the duration within which communication operations should occur. This work, however, does not cater for the mobility of agents. Furthermore, the work is limited to soft real-time, in that it does not cater for coordination reliability.

2.4.2 Message-centric

Another class of coordination models is the message-centric class. Models in this class are mostly built on the publish/subscribe or event-based communication abstraction.

2.4.2.1 Publish/subscribe architectures

Location-based Publish-Subscribe (LPS) (Eugster et al. 2005) is a publish/subscribe architecture designed specifically for collaboration in mobile ad hoc applications. The main difference between LPS and traditional publish/subscribe architectures is that event dissemination and reception is bounded in physical space: a publisher defines a publication range and a subscriber defines a subscription range. Only when the publication range of the publisher and the subscription range of the subscriber overlap is an event disseminated to the subscriber.

A similar model is provided by the Scalable Timed Events And Mobility (STEAM) (Meier & Cahill 2003, Meier, Cahill, Nedos & Clarke 2005) middleware. STEAM builds on the observation that event consumers are typically interested in events produced by entities in their vicinity. For this reason, STEAM adds the possibility to filter events based on geographical locations, using *proximities*, to traditional content-based filtering. Proximities can be of any shape and can be defined either absolutely (via GPS coordinates), or relatively around the entity (using an anchor point and a size) (Killijian et al. 2001). RT-STEAM (Meier, Hughes, Cunningham & Cahill 2005) is a real-time version of STEAM.

Newer work offers a context-aware publish/subscribe service in mobile ad hoc network (Frey & Roman 2007). This service allows context information, such as position and direction, for example, to be exploited when matching events against subscriptions.

None of these systems, however, offers explicit support for coordination, i.e., offer laws that entities can use to coordinate their behaviour.

2.4.2.2 Coordination middleware

Motivated by the observation that in some applications, coordination needs to be ensured by the exchange of multiple, related, messages, i.e. a (distributed) protocol, Schelfthout et al. (2006) describe a middleware supporting protocol-based coordination in dynamic networks. They argue that protocols often rely on some kind of identification, or at least aggregate properties of their interaction partners, that are not available in other existing coordination models where communication is typically anonymous. Furthermore, protocols might need one-to-one communication, while existing models and architectures typically support only one-to-many communication. Finally, existing coordination models or middleware do not provide any support for stateful communication, i.e., state information maintained between messages.

The proposed solution is based on two abstractions: *roles* and *views* (Schelfthout et al. 2005, Schelfthout & Holvoet 2005, Schelfthout 2006). Roles specify the behaviour of a class of interaction partner in a particular interaction, in terms of the messages that are sent and expected to be received

(as well as their timing, and their recipients or senders), and the relation between the role and the behaviour of the application component that it represents. Views are collections of data objects, that are copies or representations of data objects available on a set of nodes in the network, kept up-to-date automatically by the middleware. Because of the nature of these abstractions, this work could be seen as combining aspects of both data-centric and message-centric approaches. The supporting middleware, however, provides only best-effort guarantees in the presence of unreliable communication. In addition, no support for real-time is offered.

Finally, some work in this category caters for timeliness requirements (Limniotes et al. 2002), but addresses only the coordination of static components, and therefore does not support the possibility of connectivity loss.

2.4.3 Analysis

The coordination model community recognises the need for generic coordination models, to replace application-specific approaches, hence allowing modularity, reusability, exchangeability and extensibility of coordination mechanisms (Deugo et al. 2001). However, none of the existing work deals with both mobility and real-time concerns. As we have established in Chapter 1, real-time communication and coordination in mobile settings is very challenging. Therefore, the approach in our work offers less high-level abstractions than some of the existing data-centred approaches. Our work, however, aims to provide stronger reliability guarantees, suitable for safety-critical applications.

As discussed by Schelfthout (2006), the tuple-space abstraction is inherently coupled in space (as entities must know which tuple-space to access) and decoupled in time (entities do not need to be present at the same time). Conversely, publish/subscribe middleware is inherently decoupled in space (entities do not need to know where other entities are) and coupled in time (entities need to be present at the time where a message is sent). The application domain of this thesis calls for time coupling, as real-time requirements play an essential role for such applications. This motivates our choice of an event-based abstraction. Comhordú is based on the space-elastic model, which itself relies on RT-STEAM (this is detailed in Chapter 3).

Furthermore, similarly to the middleware supporting protocol-based coordination (Schelfthout 2006), our work aims to provide application designers with the possibility of designing protocols i.e., the exchange of several related messages, for the coordination of entities. This is modelled in Comhordú by the notions of roles and contracts. In addition to supporting the definition of such protocols, our work aims to support developers in using Comhordú for deriving these protocols automatically from a problem specification.

2.5 Mobile real-time systems

Providing real-time guarantees requires specific mechanisms at every level of a system. This difference characterises work in the real-time systems community. A number of projects in the real-time systems communities are particularly relevant to the work presented in this thesis. In particular, recognising the limitations of the deadline concept to deal with open systems, a number of projects have explored how to capture the real-time requirements of open systems. In addition to this work, two bodies of work deal with the real-time coordination of autonomous mobile entities. We review each of these in detail.

2.5.1 Specifying real-time requirements

Real-time requirements are traditionally specified at the level of the system implementation, by defining deadlines for messages and tasks (Stankovic et al. 1999, Bickford et al. 1996, Schemmer & Nett 2003). This can only be done after the system is designed, and requires that developers derive low-level real-time constraints so that the safety of the application is ensured. However it might be that the safety constraints required by the application cannot be achieved with the given system design, requiring a redesign of the application, and more generally a trial and error process. For this reason, a number of projects have aimed at specifying real-time requirements at a higher level, independently of the low-level implementation details.

2.5.1.1 RT-entities and RT-objects

An approach to a general formulation of global timing constraints, independent of a specific programming model has been presented (Kopetz & Kim 1990) and is partly adopted in a general model of real-time systems (Kopetz 1997, 2001). This model comprises RT-entities, reflecting the environment, and RT-objects, that make up the computer system. An RT-entity is a state variable of relevance for the given system, and is located either in the environment or in the computer system. Information about the state of an entity at a particular point of time is captured by an observation, composed of the name of the RT-entity, the point of time when the observation was made, and the observed value of the real-time entity. RT-objects contain the information about the environment as stored in the computer system. Timing constraints on an RT-object can be expressed as the "accuracy", denoting the temporal gap between the state of an RT-entity in the environment and the state of an associated RT-object. So the accuracy describes the consistency between the system and the environment in the temporal domain. The notion of "consistency constraint" is defined for specifying consistency conditions to be fulfilled in the value domain between different RT-objects. But no notion is defined for expressing timing constraints between RT-objects. Therefore, this model is restricted to expressing timing constraints between the environment and the system (and between replicated objects) and cannot be applied for expressing timing constraints between autonomous systems (Mock 2004*b*).

2.5.1.2 Real-Time UML

The Unified Modelling Language (UML) is a graphical language for specifying, visualising, constructing, and documenting object-oriented software systems. UML in itself does not provide any means of defining explicit temporal constraints or timing properties. Two approaches address these deficiencies: UML for Real-Time, and Real-Time UML.

UML for Real-Time (UML-RT) extends UML with constructs to facilitate the design of complex embedded real-time software systems (Lyons 1998, Selic 1998). The constructs of "capsule", "port" and "connector" provide additional support for modelling the structure of the system, while a "protocol" models communication within the system. Capsules are software components, potentially physically distributed, whose internal structure is described by sub-capsules and the connections between these. A component interacts with its surroundings, and with its sub-capsules, through a set of ports that are the only parts of a component that are visible to others. Connectors are used to model communication channels between two or more ports. A protocol defines a number of roles, and the signals sent and received by each role. Compared to standard UML, UML-RT provides some additional support for modelling the architecture of interactive systems. It does not, however, provide support for modelling timing issues (Carlson 2002).

Real-time UML (RT UML) is another extension of UML for real-time systems. The key elements of RT UML have been standardised as a UML profile for schedulability, performance, and time (Object Management Group (OMG) 2005). In this profile, timing constraints can be expressed by timing annotations, and capture timing requirements on the messages exchanged both between the system and external actors, and between objects within the system. Timing requirements, however, can only be specified at a given level if the system structure in terms of objects and messages exchanged between the objects is already specified at that level.

2.5.1.3 Precision consistency relation

Mock (2004*a*,*b*) described a framework for the coordination of autonomous systems, in which the process of choosing a programming model and communication infrastructure, with respect to given real-time requirements, is supported in a more formal way. For this purpose, a model is proposed to express the mutual timing dependencies between cooperating autonomous systems, independently of the specific system structure.

This model is based on the notion of the *precision consistency relation* (PCR), which is essentially a mathematical relation between the variables of the different objects involved, capturing the application semantics. To this notion is added the idea of *precision distance* (PD) within which the PCR must be fulfilled. This requires that for each point in time t_0, there be observations in the time interval $[t_0, t_0 + PD]$ of the object variables such that the variable values fulfil the PCR. From this notion, the suitability of programing models and communication infrastructures to implement the solution can be assessed. The framework, however, does not allow an application to be designed in more

detail and does not offer implementation-support. Furthermore the generality of the formalism for the constraints implies that it would be really hard to do so.

2.5.2 A Middleware for Cooperating Mobile Embedded Systems

Nett & Schemmer (2004) and Schemmer (2004) point out the need for reliable real-time coordination of autonomous mobile embedded systems and describe an architecture to support the coordination of mobile embedded systems. The proposed architecture encompasses a real-time CPU scheduling service based on the time-aware fault-tolerant (TAFT) scheduler (Nett et al. 1997, Becker et al. 2005). The TAFT scheduler allows users to define an exception part that is executed when the main part is about to miss it deadline. This allows working with expected-case execution times (instead of worst-case execution time as in traditional real-time approaches), and still achieves predictable timing behaviour.

The timeliness of communication is also examined in this work. The proposed solution offers timely communication to clusters of systems on wireless LANs (Mock et al. 1999). A bound on the time required for joining the WLAN, however, is not available in the general case and requires known application-specific properties such as bounded arrival rate (Nett & Schemmer 2004).

Finally, the highest middleware layer of the architecture is the cooperative application development interface (CADI). This interface offers three services: a transparent service infrastructure that allows the mobile systems to access a distributed service infrastructure transparently, i.e., without explicitly searching and contacting the nodes providing the services; a globally consistent system state that provides a consistent view of the relevant local states of all participating systems with respect to the same point of time to the application; and an homogeneous world model that provides an homogeneous view of the environment. These services are based on the timely communication layer to distribute the information and keep it consistent over all the systems. The authors argue that, provided that systems have a common view of the system state, they can coordinate their behaviour by using this state as input to decide their action locally.

The architecture, summarised in Figure 2.2, has been tested on a traffic control application where robots coordinate their trajectories to negotiate a shared space (Schemmer et al. 2001). The middleware, however, relies on reliable communication, which is provided only for infrastructure networks and under application-specific properties.

2.5.3 GEAR

Following the observation that uncertainty is not ubiquitous nor everlasting, i.e., that systems have some parts that are more predictable than others and tend to stabilise, Veríssimo et al. (2000) suggest constructing dependable real-time applications by using a (distributed) component capable of executing timely functions. This component, called the Timely Computing Base (TCB), can be used by other components to execute a set of time-related services (see Figure 2.3). More precisely, the TCB subsystem is assumed to have known upper bounds on processing delays, on the rate of drift of local clocks,

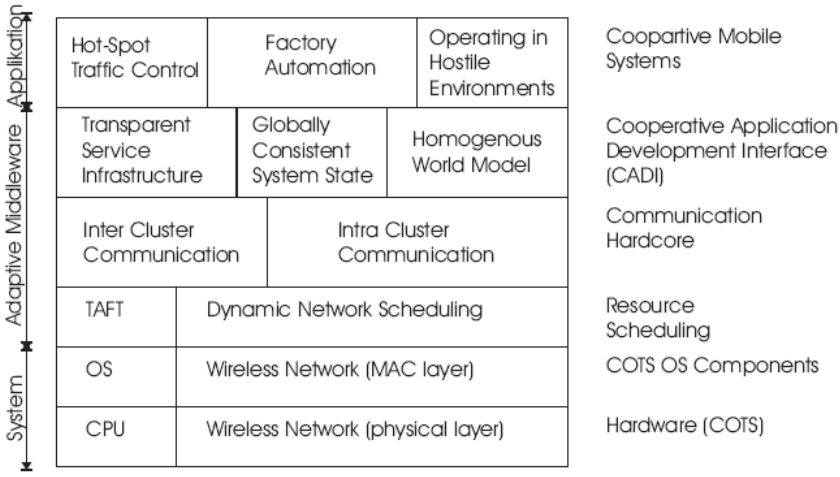


Figure 2.2: An architecture to support cooperating mobile embedded systems (Nett & Schemmer 2004).

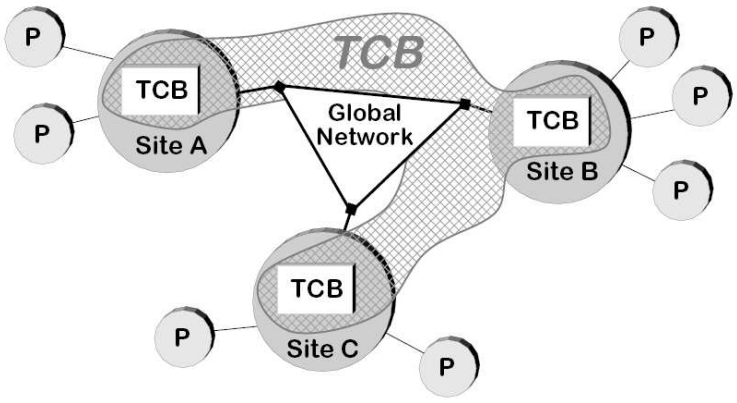


Figure 2.3: The TCB architecture (Veríssimo et al. 2000).

and on message delivery delays. These assumptions allow the TCB to provide other components with the following services: timely execution, duration measurement, and timing failure detection. Timely execution is defined with respect to livelines (time before which an execution should not start) and deadlines provided that they are achievable, i.e., that the task execution time is bounded by a duration d and that there is at least d between the liveline and the deadline. These services have been shown to be sufficient for implementing three classes of applications: fail-safe applications, which exhibit correct behaviour or else stop in a fail-safe state; time-elastic applications, where time bounds can be increased or decreased dynamically; and time-safe applications, where no incorrect behaviour results from the violation of safety properties on account of the occurrence of timings failures (Veríssimo & Casimiro 2002).

The authors have implemented the TCB on wired architectures, and have suggested a design for wireless architectures; both designs rely on separating the control network from the payload network

(of the application) using a dual network architecture (Martins et al. 2004). The TCB model, however, relies on the assumption that synchronous properties, such as known bounds on processing and message delivery delays are achievable and maintained. Given the challenging characteristics of wireless mobile networks, it is not clear that these synchrony properties can be assumed. For example, in the implementation of a wireless TCB (Martins et al. 2005), the synchronous properties of the control channel are maintained only by making assumptions about the infrastructure of the ad hoc network.

GEAR (Generic Event ARchitecture) (Veríssimo & Casimiro 2003) is an architecture that allows the seamless integration of physical and computer information flows, i.e., allows information from both the environment and other parts of the systems to be treated in the same way. Its use is demonstrated on a cooperative car scenario. This scenario is an example of the fail-safe application class: cars stop when a timing failure occurs, hence guaranteeing that safety is ensured. The solution relies on perfect timing failure detection, such as supported by the TCB. The proposed approach, however, seems to be fairly inefficient, as cars have to stop every time a timing failure occurs. It is noted that the maximum car speed can be adapted depending on the current time bound, however, while the authors point out that knowing when and how to make these adjustments is crucial, they do not answer these questions. Furthermore, the characteristics of the environment might mean that in some areas cars cannot communicate with other cars, which can lead them to being stuck in that area. Finally, the scalability of the solution has not been studied, and no guidance is provided for the design of other applications.

2.5.4 Analysis

This section reviewed efforts to specify real-time requirements for mobile systems. Comhordú aims to allow developers to program entities by supporting the translation of system-wide safety constraints into sets of requirements on the behaviours of individual entities, therefore system-wide safety constraints should be expressed independently of implementation choices. For this reason, RT UML, which allows real-time requirements for such systems to be specified but requires the system to be fully specified beforehand, is not suitable for our goal. Comhordú, however, has been influenced by the work on the precision consistency relation. Similarly to that work, Comhordú allows constraints using conditions on the variables of entities involved to be formalised, independently of implementation. The approach taken in the work on the precision consistency relation, however, is to formulate a very restricting constraint and then to specify how much it can be relaxed. This, however, implies the existence of some hard constraint(s), which should never be violated, that dictates how much the strong constraint can be relaxed. For this reason, the approach in Comhordú is to express a hard constraint directly, and then to extrapolate the conditions under which this constraint is not violated. In addition, the approach adopted by Comhordú allows the systematic generation of a skeleton of the entities' implementations, as opposed to only advising on a suitable architecture.

The second part of this section reviewed the work of Nett et all, whose goals are similar to ours.

Their approach to CPU scheduling relies on predicting timing failures and avoiding them by adapting the behaviour. This approach of predicting potential failures and adapting the behaviour so that it does not happen is at the core of Comhordú, but addresses safety faults, at a system-wide level. The approach to coordination chosen by Nett et al is to provide entities with a consistent view of the world. While they provide this consensus-based service on infrastructure networks, the reliability of this service is guaranteed only under application-specific assumptions, and the high level of this service implies that it would be very hard to implement on infrastructureless networks such as those on which the applications catered for in this thesis might run.

Finally, this section introduced the work on the TCB and the GEAR event architecture. Similarly to our work, this work is based on the idea that to provide dependable applications in unreliable environments, applications should adapt their behaviour dynamically depending on the current environmental characteristics. In these models, a notion of fail-safe states is defined, which is similar to the fail-safe modes described in this thesis. In the TCB model, entities adapt their behaviour depending on the latency with which communication is currently available. The distance dimension, however, plays a particularly important role in mobile scenarios. In particular, it is crucial for scoping interactions. For this reason, the approach used in Comhordú is to adapt the behaviour of entities depending on the varying area in which real-time communication is available within a fixed latency, as opposed to adapting the behaviour depending on the varying latency for real-time communication in a fixed area. Therefore, the work in this thesis caters for a class of space-elastic applications, as opposed to the time-elastic class catered by the work described in this section. Coordination support provided by the GEAR architecture is limited to event delivery, and the problem of ensuring system-wide constraints using these events is not addressed.

Finally, conversely to the work described in this section, the work described in this thesis aims to not only offer an execution model for entities, but also to support developers in systematically translating system-wide safety constraints into requirements on the behaviour of individual entities.

2.6 Comparison

Previous sections have reviewed related work on the real-time coordination of autonomous mobile entities. This section outlines the requirements that a solution to this challenge must meet. It then compares the systems previously presented using these criteria, and analyses their approach to coordination. Finally, the concepts presented in the previous sections that have particularly influenced our work are recalled.

2.6.1 Requirements

In this section, we outline the requirements that a system needs to fulfil to ensure the safe coordination of autonomous mobile entities. We distinguish three types of requirements: on the characteristics of

the problem tackled, on the information means supported, i.e., how an entity can get information about its environment, and on the solution characteristics.

2.6.1.1 Problem characteristics

Entities can either be *physical* or *virtual*, i.e., composed only of software. In addition, entities can evolve either in a *static* environment, whose characteristics do not evolve over time, or a *dynamic* one, where characteristics might evolve, for example, obstacles might move. This thesis caters for the coordination of physical entities operating in a dynamic environment. Therefore, a solution needs to support physical entities, and not only software ones, as well as handle a dynamic environment.

The entities in a system might interact with entities of *known identity* or *type*, as, for example, in team robotics, or conversely, interactions can be *spontaneous*. To be more generic, a solution must support interaction between an unknown number of entities, whose identities are not known in advance. Finally, a solution might be *specific* to a certain type of entity, or to a specific coordination problem, or *generic*, i.e., applicable to numerous coordination problems. Our work aims to be generic, which requires that a solution make as few as possible assumptions about the entities or the coordination problem.

2.6.1.2 Information means

Entities can get information about their environment, including the behaviour of other entities, by *direct communication*, *sensing* and EMC (or *indirect communication*). As explained in Chapter 1 (and detailed further in Chapter 3), all of these information sources are inherently unreliable. Solutions might support direct communication via *wired* communication only, or via *wireless* communication. In addition, they may or may not take the *unreliability* of each of the information means into account.

The unreliability of communication channels is typically alleviated by the use of redundant channels. Using different media for these redundant channels ensures that their error models are different and that their errors are not correlated. For this reason, we believe that, to enable reliable coordination, a solution needs to both support all of these information means so that they can complement each other, and take their unreliability into account so that system safety is not violated.

2.6.1.3 Solution characteristics

Solutions can be categorised depending on their organisation: centralised or decentralised. A centralised system has an agent (the leader) that is in charge of organising the work of other agents. In a decentralised system, agents are completely autonomous in the decision process. In a *strongly-centralised* system, decisions are taken by the same pre-defined leader, while in *weakly-centralised* systems, the leader is not chosen a priori, but is selected dynamically depending on the current situation of the team and the environment (Farinelli et al. 2004). Centralised solutions typically include

a single point of failure, do not scale well and are costly in terms of communication. For this reason, centralised solutions are not considered.

Examination of the related work described in this chapter shows that decentralised systems can be split in two categories: *consensus-based* systems that require entities to acquire a common view of a problem, and *emergent* systems that do not. In the first case, entities either have a common view of their environment and apply an algorithm to decide their actions locally, or communicate until they devise a common plan of action. In the emergent case, properties are ensured by reactivity, i.e., as defined in Section 2.2.1, entities react independently to what they perceive of their environment. The emergent system paradigm requires the least communication, and therefore takes the least time, and scales gracefully (Ijspeert et al. 2001). These characteristics make it the most appropriate paradigm for solving the problem described in Chapter 1.

Coordination often relies on an explicit predefined protocol. Solutions supporting this feature can be qualified as *strongly coordinated*, while ones that do not are *weakly coordinated* (Farinelli et al. 2004). Intuitively, strong coordination is more efficient and more reliable.

An essential characteristic of a solution to the problem tackled in this thesis is the degree of timeliness supported. Because entities are interacting with their environment, they must be coordinated in real-time. Solutions might offer *no real-time support*, *soft real-time support* where timeliness is ensured as often as possible, and *hard real-time support* where timeliness must be guaranteed. As this thesis addresses safety-critical applications, hard real-time support is required.

Finally, the last requirement on a solution is the support it offers for application development. While some solutions offer only a formalism to capture some properties or a *model*, others focus on providing a *middleware* to support coordination. Ideally, the complete process from specifying high-level requirements to *generating* the implementation and supporting the execution should be supported.

2.6.1.4 Summary

To summarise, the requirements on a system to ensure the safe coordination of autonomous mobile entities are the following (each criteria is listed, followed by a list of possible values, the required value is in bold):

- Problem characteristics
  - Entities: virtual, **physical**
  - Environment: static, **dynamic**
  - Interaction: with known entities, with known entity types, **spontaneous**
  - Applicability: application specific, **generic**
- Information means

  - Direct communication: none, wired, wireless reliable, **wireless unreliable**
  - Sensing: none, reliable, **unreliable**
  - Indirect communication: none, reliable, **unreliable**

- Solution characteristics
  - Organisation: strongly centralised, weakly centralised, consensus-based, **emergent**
  - Coordination: none, weak (no protocol), **strong** (with a coordination protocol, i.e., a set of rules that entities must follow)
  - Timeliness: non-real-time; non-real-time and soft real-time; **non real-time, soft real-time and hard real-time**
  - Support: model; model and supporting middleware, **model, supporting middleware and code generation from model**.

2.6.2 Systems comparison

In this section, we review the systems identified in the related work using the criteria defined above. Note that because most of the criteria are not explicitly mentioned in the description of the different systems, the accuracy of the following classification is limited by available information.

2.6.2.1 Problem characteristics

A number of the projects reviewed, in particular in the MAS and coordination model communities, cater only for the coordination of software agents. This work cannot be applied to the coordination of physical entities as interaction with the environment implies extra requirements for dealing with the uncertainty of sensors and actuators. Other work in these communities (Weyns, Schelfthout, Holvoet & Lefever 2005, Murphy & Picco 2006, Schelfthout et al. 2006), and work in the robotics, ITS, and real-time communities, however, deals with physical entities. Some of these projects, in particular the earlier work in the robotics community, solved the coordination problem only in the context of a known and static environment. This assumption is very restrictive and not suitable for the types of applications for which our work is catering. Most of the more recent work, however, deals with a dynamic environment.

Some work, in particular in the robotics community (e.g., Parker 1998), only caters for the coordination of a known team of robots. This, however, constrains the application and prevents evolution and addition of new entities. Other work caters only for the coordination between entities of known and fully-specified types. This is, once again, restrictive. A solution for the problem tackled in this thesis needs to allow the addition of new entities, and the coordination of an unspecified number of entities of types that might not be fully specified a priori.

Finally, work in the MAS, robotics, and ITS communities, often caters for only specific applications of coordination: for entities of a specific type, for example, AGVs (Weyns, Schelfthout & Holvoet 2005), for coordination of a specific aspect of behaviour: task allocation (Parker 1998), platooning (Hallé & Chaib-draa 2005, Michaud et al. 2006), or intersection management (Dresner & Stone 2005, Sheng et al. 2006) for example. Other work in these communities, and work in the coordination model community, however, offers generic solutions.

2.6.2.2 Information means

A number of the projects reviewed rely only on direct (wireless) communication for coordination, in particular in the ITS, coordination model and real-time communities (Dresner & Stone 2005, Schelfthout et al. 2005, Schemmer 2004). Amongst the others, only four systems take into account not only sensing, but also indirect communication (Parker 1998, Hallé & Chaib-draa 2005, Michaud et al. 2006, Veríssimo & Casimiro 2003).

As explained in Chapter 1, wireless communication is inherently unreliable: its range is limited, and obstacles also mean that this range varies over time and distance. Amongst the reviewed work, these characteristics are only taken into account by five bodies of work (Parker 1998, Clark 2004, Dresner & Stone 2005, Meier, Hughes, Cunningham & Cahill 2005). Similarly, sensing is limited in range and accuracy, and only four bodies of work take this into account (Parker 1998, Michaud et al. 2006, Sheng et al. 2006, Bouraoui et al. 2006). Finally, amongst the work that has been reviewed, only ALLIANCE (Parker 1998) supports indirect communication and takes into account sensor unreliability, and hence the unreliability of indirect communication.

2.6.2.3 Solution characteristics

Some work, in particular in the robotics community, adopts a centralised architecture. We have not reviewed those in detail, as a centralised architecture is not suitable for the coordination of entities that cannot easily communicate. Therefore, the solutions that we reviewed are decentralised. Most of these aim to obtain a consensus between all entities. The exceptions that adopt an emergent approach are coordination without communication solutions and reactive approaches for collision avoidance in MRS, as well as work on the TCB that relies on detecting timing failures and informing entities of them, hence catering for cases where consensus cannot be maintained.

Most of the coordination work cited offers weak coordination, as no protocol is defined for how entities will coordinate their behaviour, with the exception of the middleware supporting protocol-based coordination (Schelfthout et al. 2006), and various application-specific solutions. Note however, that most solutions rely on some underlying protocols to provide entities with consistent data, but these protocols do not define how entities should use the data to coordinate their behaviour.

Some systems do not take timeliness into account at all, for example, most work in the MAS community, as well as most coordination models. Most other work only caters for soft real-time

requirements: timeliness is a concern, but only a best effort guarantee is provided. Only three bodies of work cater for the provision of hard real-time requirements (Meier, Hughes, Cunningham & Cahill 2005, Schemmer 2004, Veríssimo & Casimiro 2003).

The level of coordination support in terms of application development is varied. In the MRS and ITS community, coordination support is typically low-level and does not offer high-level semantics. In the coordination community, coordination is supported via high-level constructs such as tuple spaces. In the real-time community, work is typically focused on offering a common view of the environment.

2.6.2.4 Summary

In this section, we classified the systems reviewed according to the criteria listed in the previous section. A summary of this classification is shown in Table 2.1. This table lists only the most significant systems in each community. In addition, these systems are ranked only according to the following criteria: applicability (generic or specific); information means supported and whether their unreliability is taken into account; and timeliness. While this does not capture all the criteria mentioned above, this subset is sufficient to show that none of the existing work addresses the real-time coordination of autonomous entities as specified in Chapter 1. Each criteria is rated using a number of stars, and as justified in the previous section, a system suitable for the problem that we tackle needs to have a rating of two stars for each criteria (this corresponds to generic applicability, supporting both direct and indirect communication as well as sensing with their unreliability being taken into account, and offering hard real-time support). The table shows that none of the systems achieve this ranking.

2.6.3 Analysis

As outlined above, most of the work reviewed adopts a consensus-based approach. This section first assesses the suitability of such an approach for solving scenarios that have stringent timeliness and reliability requirements, such as those catered for by this thesis. We then examine the suitability of the approaches adopted by other systems.

2.6.3.1 Consensus-based systems

In consensus-based systems, entities must reach an agreement either on their view of their environment, or on the action that they should take. In the first case, which is adopted by the majority of the work reviewed in this chapter, the common view of the environment can be used as input to an algorithm for each entity to decide locally what actions it can and should undertake. As entities use the same input data, the decisions taken locally can be made consistently (i.e., ensure that the resulting actions will be safe). The common view of the environment is typically delivered by a middleware, hence hiding the steps necessary to reach the consensus. In the second case, either a leader entity takes the decision and communicates it to other entities, as in centralised systems, or all entities communicate with one another until they reach an agreement on the action to take.

System	Applicability	Information means: Direct communication	Information means: Sensor data	Information means: Indirect communication	Timeliness
Multi-Agent systems (MAS)					
Decentralised control of automatic guided vehicles (AGVs) (Weyns, Schelfthout & Holvoet 2005)	★	★	★	-	★
Multi-Robot systems (MRS)					
ALLIANCE (Parker 1998) and successors	★	★★	★★	★★	★
A distributed layered architecture for mobile robot coordination (Goldberg et al. 2002)	★★	★	★	-	★
Dynamic robot networks (Clark 2004)	★★	★★	★	-	★
Intelligent transportation systems					
Collaborative Driving system using teamwork (Hallé & Chaib-draa 2005)	★	★	★	★	-
Coordinated maneuvering of automated vehicles in platoons (Michaud et al. 2006)	★	★	★★	★	★
Multi-agent traffic management (Dresner & Stone 2005)	★	★★	-	-	-
Experimental testbed for cooperative driving (Sheng et al. 2006)	★	★	★★	-	★
Cybercar cooperation for safe intersections (Bouraoui et al. 2006)	★	★	★★	-	★
Coordination models					
Data-centric (LIME, EgoSpaces,...) (Murphy et al. 2006)	★★	★	★	-	-
RT-STEAM (Meier, Hughes, Cunningham & Cahill 2005)	★★	★★	-	-	★★
Middleware supporting protocol-based coordination (Schelfthout et al. 2006)	★★	★	-	-	-
Real-time systems					
A middleware for cooperating mobile embedded systems (Schemmer 2004)	★★	★	★	-	★★
GEAR (Veríssimo & Casimiro 2003)	★★	★★	★	★	★★

Legend: Applicability: ★ specific; ★★ generic.
Information means: - not supported; ★ supported, assumed reliable; ★★ supported, assumed unreliable.
Timeliness: - none; ★ soft real-time; ★★ hard real-time.

Table 2.1: Comparison summary.

In applications that exhibit timeliness requirements, consensus has to be reached within a bounded time. To reach a consensus, entities need to communicate with each other, either via direct or indirect communication or through some form of shared data structure to which they all have access. In mobile settings, however, entities might not be able to communicate with each other over some periods of time. Therefore, consensus-based approaches can only offer best-effort guarantees and are not suitable for applications exhibiting strong reliability and timeliness requirements in mobile settings.

2.6.3.2 Other systems

Amongst the work described in this chapter, we have identified two main approaches to coordination that do not rely on consensus. Coordination without communication, also called reactive coordination (Gervasi & Prencipe 2004, Ijspeert et al. 2001, Schermerhorn & Scheutz 2006), is based on entities reacting to the behaviour of other entities to coordinate their actions. The approach, however, relies on entities accurately sensing the behaviour of other entities and, as sensor information is unreliable, can only offer best-effort guarantees.

The second approach is the one adopted by the TCB (Veríssimo & Casimiro 2003), where the unreliability of real-time communication is taken into account and catered for by informing entities in real-time of communication failures, so that they can react to them. The way in which entities should react in case of a communication failure, however, is not investigated. In addition, the TCB guarantees that timing failure detection is bounded only with respect to the time at which the event was delivered or the action finished (Veríssimo & Casimiro 2002). In particular, this means that the delay between the time of the timing error and the entity notification is not bounded, hence making it impossible for entities to make provision for possible timing errors. Therefore, the TCB approach is also not suitable for the provision of safety constraints with real-time requirements in mobile settings.

2.6.3.3 Conclusion

The consensus-based approach adopted by most related work is not suitable for the provision of applications exhibiting strong reliability and timeliness requirements in mobile settings. For this reason, Comhordú adopts an alternative approach to coordination, which allows safety to be guaranteed even in the absence of a consensus between entities. This approach uses coordination without communication, but also supports direct and indirect communication by exploiting real-time feedback on currently available information provided by novel real-time sensing and communication models. These models are similar to the TCB, but offer a bound on fault notification, hence allowing safety to be guaranteed even when available information is limited.

2.6.4 Other influential concepts

Comhordú was designed so that it can be applied systematically, making it suitable for tool-support and automation. Inspired by work on the precision consistency relation (Mock 2004*b*), Comhordú

encompasses a high-level formalism to express system-wide safety constraints by relations on the states and actions of entities.

These constraints are distributed using the notion of roles, as used in some work in the MRS, ITS, and coordination model communities (Hallé et al. 2003, Michaud et al. 2006, Schelfthout 2006), as well as contracts, that are similar to the notions of commitments and conventions defined in the MAS community (Jennings 1993). To enforce these contracts, entities can use not only direct communication and sensing data, but also indirect communication as formalised by the MAS community. The combination of the contract and communication means used constitutes a protocol similar to those defined in the middleware supporting protocol-based coordination (Schelfthout et al. 2006).

Distance plays a fundamental role in our approach by allowing interactions to be scoped and therefore the complexity of the problem reduced. This approach was also adopted for collision avoidance (Yared et al. 2007) and Dynamic Robot Networks (Clark 2004) in the MRS community, as well as platooning (Michaud et al. 2006) in the ITS community. The approach in our work is to provide reliability guarantees by adapting the behaviour depending on the currently available information. This approach has been used in the real-time community (Nett & Schemmer 2004). The real-time communication and sensing models adopted are space-elastic, a contrast to the time-elastic communication model of the TCB (Veríssimo & Casimiro 2002).

2.7 Summary

This chapter has first reviewed work related to real-time coordination of autonomous mobile entities in different communities: MAS, MRS, ITS, coordination models, and real-time systems. A number of criteria have been defined to rate these systems, and these criteria have allowed us to classify the work mentioned. This demonstrated that none of the existing work is suitable for addressing the challenges addressed in this thesis. Finally, we have shown how the design of Comhordú was influenced by some of the work presented, and introduced its main characteristics. The following chapter details the models on which Comhordú is built, while Comhordú itself is presented in Chapter 4.

Chapter 3

Problem Modelling: Communication and Sensor Models

In this chapter, we describe the hypotheses on which Comhordú builds. These hypotheses are captured in a model of the environment (presented in Section 3.1), a data or direct communication model (outlined in Section 3.2), and a sensing and environment-mediated or indirect communication model (detailed in Section 3.3). The chapter also presents a comparison of the direct and the indirect communications models (in Section 3.4). The last section of this chapter presents the fault model for our work.

3.1 Environment model

This section details a model of the environment that is used in the remainder of the thesis. In particular, the communication models outlined in Section 3.2 and 3.3 are presented in terms of the environment model.

3.1.1 Elements and entities

The environment is modelled as a collection of *elements*. These can have different shape and size, and their attributes, for example position and speed, can evolve over time or not. We distinguish between *entities*, which are used to model the system, i.e., what is within the sphere of control of application developers, and *passive* elements that describe the surrounding of the system. Entities can have any of the following abilities: send and/or receive messages, sense and/or actuate. While passive elements are assumed not to have these possibilities. The goal of this work is to derive requirements on the behaviour of entities so that they be able to coordinate their behaviour to evolve safely in their environment, i.e., that each entity evolve safely amongst other entities and passive elements. Note that, as entities are autonomous, the system is fully decentralised. Furthermore, it might be that

some entities are already implemented and deployed, and therefore their behaviour cannot be altered; requirements that takes this behaviour into account can be derived.

3.1.2 Indirect communication

An entity can change its environment via actuators, and this change might be detected by another entity sensing the environment. Therefore, entities can use sensing to communicate through the environment. This form of indirect communication is sometimes termed environment-mediated communication (EMC). This principle has been characterised in the study of stigmergy, in which an agent's actions leave signs in the environment, signs that it and other agents sense and that can be used to determine their subsequent actions (Parunak 2003) (see Chapter 2). Other forms of environment-mediated communication include human environment-mediated communication, where humans use mobile computing devices and the physical environment to communicate with each other, for example by preparing an electronic note on a PDA and leaving it on a door where a colleague's can collect it later with his own PDA (Gellersen et al. 1999). In the following, we refer to environment-mediated communication between entities simply as *indirect communication*, as opposed to direct communication, which is based on message passing.

A *signal* is a change of the environment performed by an entity to communicate with other entities. For example, a siren is a signal used by emergency vehicles to warn cars of their arrival. It might be noted, however, that the distinction between a signal and other entity behaviour is not always clear: in some cases, it is not certain whether the sole goal of an action performed by an entity is to warn other entities or if it is part of its behaviour. For example, an autonomous car approaching a junction can be sensed by other entities that can then deduce that it is likely to cross, but it might not be clear whether it is doing so on purpose to be detected, or merely to get ready to cross the junction.

Indirect communication, however, is more powerful than sensing, as it enables communication of intent, i.e., information about the future behaviour of the entity. For example, a pedestrian traffic light can inform cars that it will let pedestrians cross soon by turning to amber. Sensing can also be used to infer intent, but the information may not be reliable. Furthermore, sensing cannot be used to predict non-continuous variables, nor variables whose variation rate is not bounded, unless these are correlated with another (continuous, and with a bounded variation rate) variable. This is discussed in more details in Chapter 5.

3.1.3 Element classification

The means that an entity can use to get information about elements in its environment depend on the elements' types. Sensors can be used to detect the presence of elements of any type and some aspects of their states, provided they are in range. In addition, entities can use sensing to detect a signal from another entity via indirect communication. Table 3.1 summarises the different types of elements in

	Elements	
	Entities	Passive elements
Characteristics		
Can receive messages and signals	✓	×
Means for coordination		
Detection through sensor	✓	✓
Indirect communication	✓	×
Direct communication	✓	×

Table 3.1: Different types of elements of the environment, their characteristics and means for getting information about them.

the environment, their characteristics and the means that can be used to obtain information about them.

3.2 Direct communication model

As discussed in Chapter 2, for entities having only limited information about other entities and their environment to make progress, while ensuring system-wide safety constraints, requires that they have information about the current state of communication. In particular, if the safety constraints imply real-time requirements, entities need to have feedback within a bounded time when communication is degraded. While the feedback on the state of current communication could use any of several metrics, such as membership like in group communication systems, or currently achievable latency like in the TCB, this work assumes feedback in the form of a geographical proximity on which communication is available. The reasons of this choice are twofold: firstly, it is to our knowledge the only model for wireless, and possibly ad hoc, networks that offers feedback within a bounded time from the time at which the quality of communication is degraded, and secondly, as discussed in the previous chapter, the choice of the distance metric is particularly suitable for applications composed of mobile entities.

This work assumes the space-elastic communication model, a model for real-time communication in wireless networks, including ad hoc networks (Hughes 2006). In this model, real-time communication is guaranteed within an adaptable geographical proximity of the sending entity. The rationale for this model is first described, then the terminology used and the model specifications are detailed. Its assumptions and implementation are then discussed.

3.2.1 Rationale

The space-elastic model exploits the rationale observed by Hartenstein et al. (2001), i.e., the relevance of context to a particular geographical area, in guaranteeing real-time constraints only within specific proximity bounds. In (ad hoc) wireless networks, varying link quality and network topology mean that it is impossible to guarantee communication with predefined Quality of Service (QoS) requirements in a fixed area. Therefore, the approach adopted by the space-elastic model is to guarantee periodic

communication with such predefined QoS requirements only within a dynamic (varying over time) proximity. Message senders are notified of changes in the communication coverage in real-time, hence allowing them to adapt their behaviour to current communication conditions.

This model is motivated by a class of applications that rely on communication with predefined QoS. To implement such applications in (ad hoc) wireless networks they must be able to rely on real-time communication, or adapt their behaviour to the quality of communication. If we consider the example of a pedestrian traffic light that uses wireless messages to warn cars when it is red, the normal mode of operation of a traffic light might be to turn to red after a pedestrian has pressed a button. But, if timely communication cannot be achieved within a wide-enough area, cars may not be informed of the state of the traffic light early enough to allow them to brake before arriving at it, and the safety of pedestrians might be at risk. Therefore, it is crucial for the traffic light to be informed about how far it can communicate. When the area in which the QoS requirements for communication are guaranteed is smaller than a threshold, the traffic light must change its behaviour and remain green independently of whether or not a pedestrian pressed the light.

3.2.2 Specifications

This section describes the terminology and parameters used by the space-elastic model.

3.2.2.1 Terminology

The terms used for the direct communication model are the following: a *sending entity* sends a *message* to a *receiving entity*.

3.2.2.2 Parameters definitions

Within this model, a sending entity announces its requirements on communication in terms of QoS and a proximity, called the *desired coverage*. The QoS is defined by the messages' period, *period*, and the latency, *msgLatency*, within which the messages must be delivered. The desired coverage is a proximity that can be of any shape and can be defined either absolutely (via GPS coordinates), or relatively around the entity (using an anchor point and a size) (Killijian et al. 2001).

Depending on the topology of the network (i.e., the distribution of the nodes and the quality of the wireless links), it might not be possible over some period of time to deliver a message in time to all interested entities within the desired coverage. Therefore, the size of the area in which timely delivery of messages is provided, called the *actual coverage*, changes over time. In the worst case, no communication is possible; this corresponds to an actual coverage of null. The sender is notified of changes to the actual coverage within a bounded time, $adaptNotif$. Therefore, an entity knows within $msgLatency + adaptNotif$ after sending a message about the area in which it has been successfully delivered, and can adapt its behaviour accordingly. Variations of the actual coverage within the desired coverage are shown in Figure 3.1.

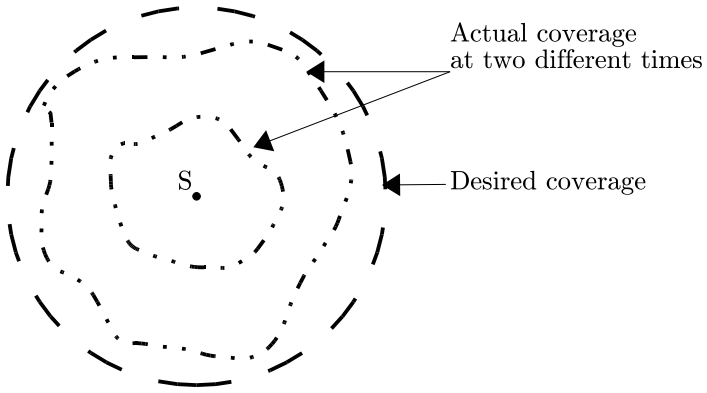


Figure 3.1: Different coverages of the space elastic model.

An entity is said to be present within the actual coverage once it is able to receive messages after arriving in the communication coverage. This takes an implementation-dependent time, *present*, which might be necessary to include the entity in the real-time route for example.

To summarise, the parameters of the model are the following:

present the time required for setting up the communication from the time an entity enters the coverage,

period the period with which messages are sent,

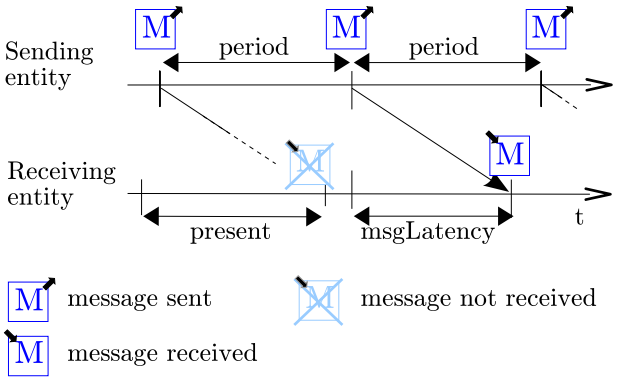
msgLatency the latency for a message to be delivered,

adaptNotif the time required for the message sender to be notified of an adaptation of the communication coverage.

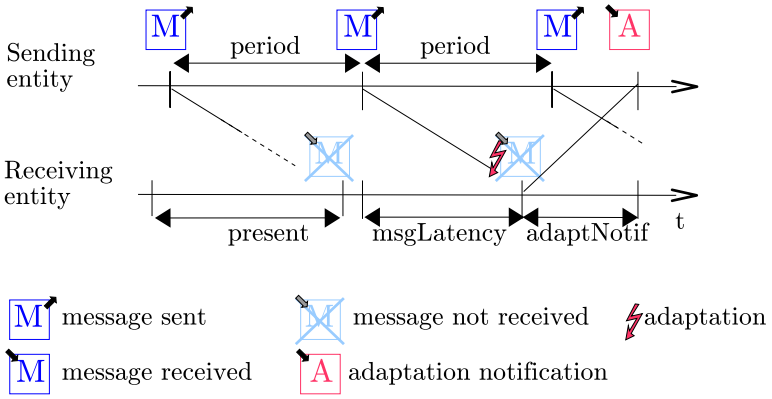
These parameters and their relationships are shown in Figure 3.2. In Figure 3.2(a), it can be seen, for example, that it takes up to $present + period$ from the time an entity enters the coverage until it receives a message, as it might become present in the coverage just after the delivery time of a message. In addition, once an entity is present in the coverage, it might take up to $period + msgLatency$ for the receiving entity to be notified of a planned change of the state of the sending entity. Similarly, Figure 3.2(b) shows that a sending entity is notified of the zone where a message was delivered at most $msgLatency + adaptNotif$ after sending it. In addition, a receiving entity might be in the coverage for up to $present + period + adaptNotif$ before a sending entity is notified that it has not received any message.

3.2.3 Guarantees

Given the specification described above, the real-time communication guarantees of the space-elastic model are:



(a) When no adaptation occurs



(b) When an adaptation occurs

Figure 3.2: Direct communication time lines.

- **to message senders**: to be able to communicate within *msgLatency* in the actual coverage, and to be notified within *adaptNotif* if this coverage changes,

- **to message receivers**: to receive every message of types in which they have expressed interest, if, at the delivery time of each message, they are present within the actual coverage of the message sender.

3.2.4 Assumptions

This model assumes that applications are space-aware, i.e., that they can specify and interpret bounds in space, reliably; for example, autonomous vehicles might be fitted with GPS. It also assumes that an area is either covered or not, not allowing for transient messages losses, or black spots.

3.2.5 Implementation

The feasibility of the space-elastic model to provide low-jitter real-time communication and time-bounded adaptation notification has been demonstrated in real-world settings (Hughes 2006). This evaluation is based on an implementation of the space-elastic model, in the form of an event-based middleware called RT-STEAM (Meier, Hughes, Cunningham & Cahill 2005). RT-STEAM is a real-time version of STEAM (Scalable Timed Events And Mobility) (Meier & Cahill 2003), which uses the SEAR (Space-Elastic Adaptive Routing) real-time routing and resource reservation protocol, over the TBMAC (Time-Bounded Mac Access Control) protocol (Cunningham & Cahill 2002).

3.2.6 Conclusions

This model allows applications to reason about available real-time communication guarantees. In our example, the behaviour of the traffic light needs to change depending on whether real-time communication within a given latency is provided in an area wide enough to allow incoming cars to stop in front of it. The space-elastic model, however, does not determine the value of this critical coverage. Furthermore, the behaviour of the traffic light needs to be constrained to ensure that it does not turn to red unless it is safe to do so, and turns back to green as soon as necessary. More generally, ensuring safety constraints requires that entities adapt their behaviour to the state of communication. This might include, for example, changing the transmission power, or the message period.

Comhordú models the communication of entities using the space-elastic model, and derives requirements on their behaviour to ensure that some safety constraints will not be violated. These requirements include when and how entities must adapt their behaviour, and in particular the size of the critical coverage(s).

3.3 Sensor and indirect communication model

This section presents a model for sensing and indirect communication. The rationale for the need for, and characteristics of, such a model is first detailed. Then the specifications of the model are presented. The following sections present the guarantees that the model provides and the assumptions on which it builds. Finally, the implementation of such a model is discussed.

3.3.1 Rationale

As well as direct communication (i.e., sending and receiving messages), entities can use sensing to obtain information about their environment and the behaviour of other entities. Sensor data is inherently of limited accuracy, and can only give information in a limited range. The accuracy and range of a sensor might vary over time, depending on the conditions; it may vary, for example, depending on the luminosity or temperature (Brooks & Iyengar 1998).

Therefore, as in the case of direct communication, entities cannot rely on continuous sensor data over a fixed range. Furthermore, sensor data is typically relevant only in a geographic area. For this reason, we have designed a sensor model with the same characteristic features as the space-elastic model: information is relevant to entities in a particular area, and entities receive real-time feedback about the current state of sensing. Whether an entity can sense a characteristic of an element depends on the distance between the entity and the element, the characteristics of the surroundings of the entity, and potentially the intensity of the signal as well as the sensitivity of the sensor used; therefore these will be the QoS parameters used in this model.

3.3.2 Specifications

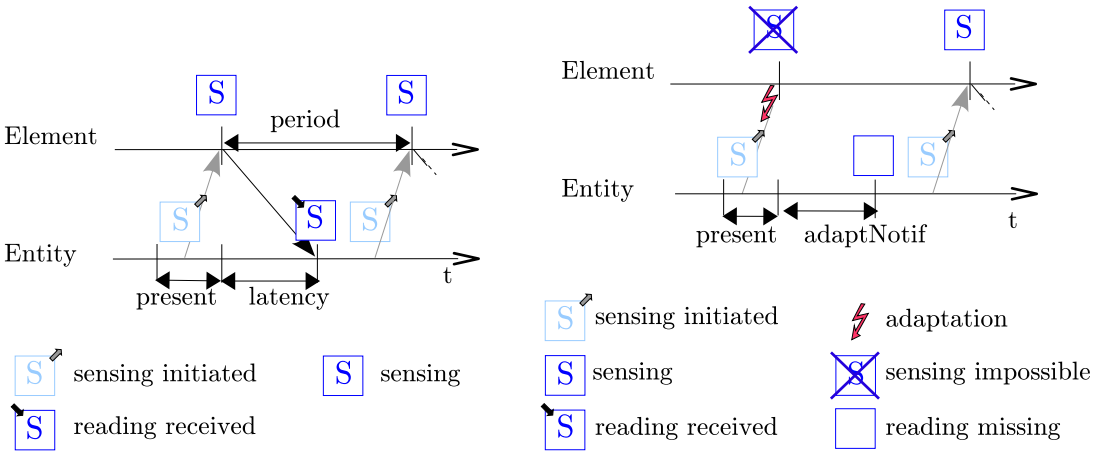
This section first presents the terminology used by this model, and then details the parameters it defines.

3.3.2.1 Terminology

We use the following terms: an entity *senses* an element of the environment, and receives a *sensor reading*. In the case where two entities communicate through the environment, a *signalling entity* emits a *signal* through the environment, which can then be sensed by a *sensing entity*.

3.3.2.2 Parameter definition

In our model, sensing is assumed to occur periodically, and to produce readings that are delivered to the sensing entity. Firstly, a sensing entity specifies a proximity over which it desires to sense, called the *desired coverage*, as well as the latency with which it wants the information, and the desired sensitivity. At any time, the sensing entity is provided with information about a sub-area of the desired coverage, called the *actual coverage*. The information is provided in the form of a map of



(a) When no adaptation occurs.

(b) When an adaptation occurs.

Figure 3.3: Indirect communication time lines.

the coverage annotated with up-to-date sensor data. Sensing entities are notified within a bounded time, $adaptNotif$, of any change to the actual coverage. Signalling entities emit a signal through the environment that is assumed to be either continuous (e.g., a siren) or persistent (e.g, by leaving a mark on the ground), so that a sensing entity can sense a signal of a signalling entity at any time provided it is in its coverage.

The timing parameters of our sensor model are the following:

present time required before sensing the environment from the time an entity enters the coverage,

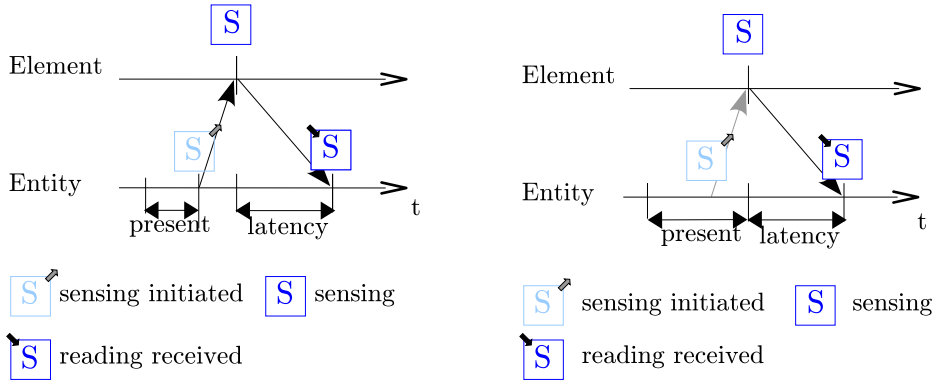
period the sensing period,

latency the latency for sensor information (between the time an element is actually sensed and the time a reading is available at the application level),

adaptNotif the time required for the sensing entity to be notified of an adaptation of the sensing coverage.

These parameters are shown in Figure 3.3. Figure 3.3(a) shows, in particular, that it takes up to $present + period + latency$ from the time an element enters the coverage until an entity senses it, as it might become present in the coverage just after a sensing time. In addition, once an entity is present in the coverage, it might take up to $period + latency$ for the receiving entity to be notified of a planned change of the state of the element. Similarly, Figure 3.2(b) shows that a sensing entity is notified of the zone for which it has sensing data at most $adaptNotif$ after sensing. In addition, an element might be in the coverage for up to $present + period + adaptNotif$ before a sensing entity is notified that it cannot sense it.

The parameter *present* includes time for setting up sensing (this might include, for example, the time required to deploy the sensors, initialise or calibrate them), and also the time to initiate the first



(a) Complete sensing process, and possible definition of the parameter *present*.

(b) Chosen definition of the parameter *present*.

Figure 3.4: Simplification of the model of the sensing process.

sensing. It might be noted that, while *present* could have been defined as only the time required to set up sensing (see Figure 3.4 a), in this model it also integrates the initiation of the first sensing (see Figure 3.4 b). Therefore, as *latency* is defined as the time between the actual sensing and the delivery of the sensor reading to the application, it will take up to *present* to initiate sensing, then *latency* to receive the sensor reading, and then sensor readings will be delivered every *period*.

3.3.3 Guarantees

The model guarantees:

- **to sensing entities**: to be able to sense within *latency* in the actual coverage, and to be notified within $adaptNotif$ if this coverage changes,
- **to signalling entities**: that their signal will be received by all entities in whose actual coverage they are present.

3.3.4 Assumptions

This model assumes that it is possible for an entity to know for certain the area on which it has information, and the accuracy of this information as well as being notified in real-time of any changes.

3.3.5 Implementation

Sensor data can be provided either by single (physical) sensors or by some kind of virtual sensors that generate data obtained by the fusion of readings from a group of physical sensors. We use the term sensor to refer to either of these, as the abstraction is the same: it delivers data to the application in response to a stimulus detected by some real-world hardware device (this definition is inspired by the definition of a sensor in the sentient object model (Fitzpatrick et al. 2002)). This sensor model is a

higher-level abstraction of sensor readings. Sensor readings can come from one or several sensors, of the same or different types. Readings can be fused, thus creating a virtual sensor, of higher accuracy or coverage (Brooks & Iyengar 1998). Each reading can be tagged with a timestamp, and the combined reading can also be given a time.

Typically, sensing or sensor fusion yields sensor readings with an associated probability. These readings correspond either to a specific geographic point, or to a zone, depending on the sensor type. In our model, sensor data is represented as available or not available. This can be achieved by using a threshold probability value, over which data is considered reliable enough to be used

Sensor readings are mapped over space. So at any time, an entity has a real-time image (Kopetz 1997) of his surroundings. This image is populated by the values of the readings where available, and marked as no reading available otherwise. This view, however, requires that it be possible to know where sensing data is available. We assume that this can be approximated using information about sensor characteristics, current environmental conditions, and possibly sensor data itself (e.g., if an ultrasound sensor detects an obstacle, it can be inferred that it has no information about the environment behind this obstacle, or if the readings of a sensor are aberrant or erratic, it can be assumed that the sensor is malfunctioning). The reading map needs to be updated automatically, so that old information is updated or removed. This will typically be achieved by applying a decay to data as it becomes older, but the specific mechanism is outside the scope of this work.

3.3.6 Conclusions

This model allows applications to reason about available sensing information. If we consider the example of a pedestrian traffic light using green-amber-red lights to warn cars when it will let pedestrian cross, then cars need to know how far they will be able to detect a traffic light to ensure that they will always have time to stop in front of it if necessary. Cars can adapt their speed depending on how far they can sense, which maps the real-life situation of a human driver adapting it behaviour to its visibility.

3.4 Comparison of the communication models

This section compares the two communication models.

The two models that we have defined use the same parameters, but three main differences can be noted:

- In the direct communication model, the entity that sends a message (the sending entity) receives feedback about where this message has been delivered. In the sensor model, however, it is the entity receiving the information (the sensing entity) that receives the feedback.
- In the direct communication model, an entity can receive messages as soon as it becomes present

	Direct communication	Indirect communication
Knowledge of communication timing (when is a message delivered)	Sending entity	Sensing entity
Knowledge of communication coverage	Sending entity	Sensing entity
First communication with an entity arriving	$present + period$	$present{+}period{+}latency$
Maximum time elapsed without information before feedback when arriving next to each other	$present + period + adaptNotif$	$present + period + adaptNotif$
Time to detect a change, once entities have discovered each other	$period + msgLatency$	$period + latency$
Freshness of data	$msgLatency$	$latency$

Table 3.2: Comparison of the two communication models.

in the proximity of a sending entity. In the sensor model, however, an entity will not receive sensor data until at least *latency* after having become present.

- Also, in the direct communication model, sending is periodic and receiving permanent (i.e., a receiving entity is assumed to be able to receive messages at any time), while in the indirect communication model, signalling is assumed to be permanent (because the signal is either continuous or persistent) and sensing periodic.

Table 3.2 details the comparison between the two models.

3.5 Fault model

In this section, we explicit how the models mentioned above relate to classical fault models for distributed systems. We first address timing aspects, and then the failure model.

Several different assumptions can be made about the timing of events in a system. At one extreme processes can be assumed to be completely synchronous, performing communication and computation in perfect synchrony. At the other extreme, they can be completely asynchronous, taking steps at arbitrary speeds and in arbitrary relative orders (Lynch 1996). A commonly accepted definition of synchronism is characterised by bounded and known processing speed, load patterns, delivery delays, and differences among local clocks (Verissimo & Almeida 1995). The implementation of the space-elastic model relies on these assumptions of synchrony, and therefore our work does too. In particular, both models rely on the availability of global time, i.e., processes having access to clocks whose differences are bounded, such as provided in (Mock et al. 2000) for example.

Processes and communication may fail in many different ways: a process might stop (crash failure), might fail to respond (omission failure), might respond outside the specified time interval (timing failure), might give an incorrect response (response failure) or might exhibit more severe failures,

where it behaves arbitrarily (Byzantine failures) (Cristian 1991). Communication failures can include message loss (omission failure) or duplication (response failure) (Lynch 1996). The space-elastic model assumes only crash, omission and timing failures. It masks these failures by changing the size of the actual coverage and notifying the sender. Our work assumes the space-elastic model, but does not tolerate any failures of the space-elastic model, nor any process failures (unless it can be enforced that processes switch to a fail-safe mode (see Section 4.3.2) before failing).

3.6 Summary

This chapter first presented a model of the environment, then a model for real-time (direct) communication and finally a model for (real-time) sensing and indirect communication. The next section of this chapter presented a comparison of the two communication models. Finally the last section presented the fault model for our work. In the next chapter, we detail how Comhordú facilitates building on these models to guarantee system-wide safety requirements.

Chapter 4

Comhordú - A Real-Time Coordination Model for Autonomous Mobile Entities

Ensuring the safe coordination of autonomous mobile entities using wireless ad hoc networks and sensor information is challenging, especially because of the unpredictability of information available. For this reason, the approach taken in this work is that autonomous mobile entities should take into account the possibility that, over some periods of time, communication or sensor coverage can be degraded, and must ensure that their behaviour remains safe even in these conditions. This chapter describes Comhordú, a real-time coordination model for autonomous mobile entities using this approach. This model builds on the sensor and communication models presented in Chapter 3.

The first section of this chapter presents our approach to the coordination problem and relates it to classical distributed system problems. Section 4.2 describes a formalism in which to express high-level, implementation-independent, system-wide safety constraints. In Section 4.3, the notions of compatibility and responsibility are introduced as the basis for distributing the enforcement of these safety constraints over entities, and three coordination primitives that entities can use to ensure the safety constraints are defined. Section 4.4 defines the notions of contracts and zones, which can be used to translate the system-wide safety requirements into requirements on an entity's behaviour and the chapter is summarised in Section 4.5.

4.1 Approach

As discussed in Chapter 2, traditional approaches to the coordination of autonomous entities rely on achieving consensus amongst these entities. As mentioned in Chapter 1, however, distributed consensus has been proven not to be solvable in the presence of an arbitrary number of communication

failures (Lynch 1996). As mobile entities communicate over a wireless network where communication is unreliable, this means that their coordination cannot be achieved via consensus.

Our approach relies on the observation that often, entities do not need to agree on their view of the world or their actions to ensure the safety constraints, but that instead some entities can take responsibility for ensuring them independently. For example, an entity could delay taking an action that might violate the safety constraints. Unless the coordination problem is trivial, ensuring the safety constraints must hinder the progress of such entities, for example, by delaying a desired action. The coordination problem then becomes a problem of how to ensure that, at any time, in every group of entities whose states might violate a safety constraint, at least one entity ensures that it does not (we call this entity a responsible entity, and this property the responsibility condition in the following).

This approach only caters for a class of applications for which some entities can, independently of other entities, take some actions to ensure that the safety constraints will not be violated. While this criteria restricts the domain of application, we found that many applications fit into this category. For example, for many mobile entities, it might be sufficient to stop to ensure that safety constraints are not violated. Similarly, it is sufficient for a pedestrian traffic light to remain green to ensure that no cars will go through a red light (though during that time, pedestrians will not be able to cross the road).

Ensuring the responsibility condition is similar to the classical distributed systems leader election problem (Garcia-Molina 1982), except that instead of aiming to have at most one leader, the problem is to have at least one responsible entity. To cater for the possibility of having several responsible entities, the way in which responsible entities guarantee that the safety constraints are not violated must be conservative, i.e., several entities must be able to take this action simultaneously. Consider, for example, that a car stops before entering a four way junction (c.f. Chapter 1) to let vehicles from another direction pass through the junction. If the vehicles approaching from all directions adopt the same strategy, the safety constraint that only cars coming from one direction should cross the junction simultaneously will still be ensured (though no car will progress through the junction). Similarly, if two emergency vehicles send messages to warn vehicles to get out of their way, they might both receive messages from one another, and get out of each other's way, which will be safe. Different protocols that satisfy the responsibility condition will lead to different trade-offs in terms of achievable progress depending on the state of communication.

In addition, we assume that, initially, the responsibility property holds. For responsible entities to make progress, they need to take actions that could potentially violate the safety constraints. Therefore, they must be informed when sensing and/or communication is sufficient to ensure that the safety can be guaranteed while they make progress. This is achieved by using the sensing and communication models defined in Chapter 3.

4.2 Specifying safety constraints

This section presents the motivations for the safety constraint formalism, its main concepts, its syntax and semantics, discusses what it means for a safety constraint to be ensured, and finally shows how a safety constraint can be decomposed into simpler safety constraints.

4.2.1 Motivations

Safety constraints for autonomous mobile entities are composed of constraints on the state of both entities and their environment, which is modelled as a collection of elements (see Chapter 3). One of the particularly relevant parameters for mobile elements is the relative positions of elements, and in particular the distance between them. Knowledge of their relative positions allows the observation that mobile entities most often need to coordinate their behaviour when they are in the same vicinity, the definition of which is application-specific (Killijian et al. 2001), to be exploited. For example, an emergency vehicle needs only to coordinate its behaviour with cars on the same stretch of road. In this section, a formalism to express these notions and their interactions is introduced.

4.2.2 Concepts

A number of concepts are used to model the problem and formalise safety constraints: scenarios, modes, states and state compatibility, as well as goals and priority lists.

4.2.2.1 Scenarios

A *scenario* encompasses a set of *element types* $E_1, E_2, .., E_n$, an ordered *priority list*, and a *safety constraint*. For example, when considering the scenario of an emergency-vehicle warning system, the element types might represent cars and emergency vehicles. The priorities for this scenario would be first for emergency vehicles to be able to drive as fast as possible to their destinations, and secondly, for cars to be able to progress as fast as allowable towards their destinations. A possible safety constraint for this scenario is that no emergency vehicle should collide with any car. In this definition, a scenario has a single safety constraint; if the interaction of the entities of a scenario must fulfil several safety constraints, either several scenarios can be defined, or these safety constraints can be linked using a logical conjunction.

4.2.2.2 Modes

The behaviour of an element depends on its type, and is composed of a set of modes of operation, termed simply *modes*, that describe the actions it can take, and the transition rules between these modes. Modes should be defined so that an element is always in one of its modes, i.e., transitions between modes are assumed to be instantaneous. For example, given the maximum speed of an emergency vehicle $v_{\max}$, and an increasing set of speeds $\{v_i\}_{i\in[0,p]}$ with $v_0 =$

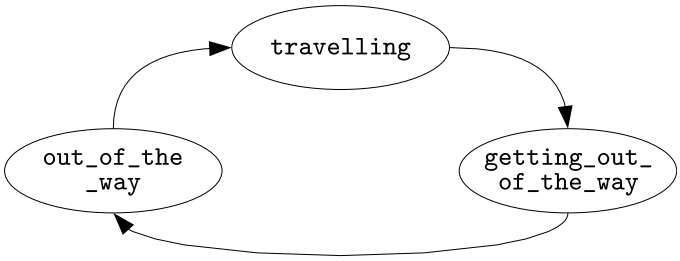


Figure 4.1: Possible mode diagram for an entity of type car.

0 and $v_p = v_{\max}$, the modes of emergency vehicles can be defined as: `stopped`, {`going_at_v`$_i$, `accelerating_to_v`$_i$, `braking_to_v`$_{i-1}$}$_{i\in[1,p]}$. Similarly, the behaviour of a car can be modelled with the modes `travelling`, `getting_out_of_the_way`, and `out_of_the_way`. The set of modes of element type E_i is denoted as M_i. Modes and the transitions between them can be represented in a mode diagram, which is a state diagram where the nodes are the modes of the scenario, and the edges are the transitions. Some transitions can be labelled with conditions on state variables, which trigger the transition. Figure 4.1 shows a possible mode diagram for an entity of type car.

The behaviours of elements can be specified in different ways. Modes represent possible actions (actuation, sensing, sending messages or signals, processing) of an element. The modes of an element type should be chosen to reflect its action in relation to the safety constraint of the scenario of which they are a part: actions that can lead to the violation of the safety constraints should be separated from actions that are safe. Furthermore, amongst the actions that can lead to the violation of the safety constraints, actions that might lead to different parts of the safety constraints to be violated should be separated. This implies that the specification in terms of modes of an element depends on the scenario for which it is specified. For example, the behaviour of a car in an emergency-vehicle warning system scenario could be described as outlined above as `travelling`, `getting_out_of_the_way`, and `out_of_the_way`, while the behaviour of the same car for a traffic light scenario might be described in terms of whether the car is aware of a nearby light and whether it is stopping in front of the light, so the modes could be `cruising`, `obeying_traffic_light`, `stopping`, `stopped`. The behaviours of elements taking part in different scenarios can be described as a composition of the behaviour description for each scenario, this process is detailed in Section 5.3.

4.2.2.3 States and state compatibility

The situation of an element at a given time is described by its *state,* which is made up of a number of *state variables*: the element's mode, position and, depending on its type, some application-specific information. State variables can vary over time and/or over space. The set of possible states of elements of type E_i is denoted as S_i. In the example, the states of both emergency vehicles and cars can be defined as of the combination of their mode, location, current speed, and direction. For an

element type E_i, the function mode $\mathcal{M}_i : \mathrm{S}_i \mapsto \mathrm{M}_i$ returns the mode of a given state.

The states $(s_{e_1}, s_{e_2}, ..., s_{e_m})$ of a finite set of elements $\{e_1, e_2, ..., e_m\}$ are said to be *compatible* if the safety constraints are not violated when the elements are simultaneously in these states. This relation is denoted as $\mathcal{C}_\mathrm{s}(s_{e_1}, s_{e_2}, ..., s_{e_m})$. For example, the states of an emergency vehicle and a car are compatible if they are far enough away. Also, the state of a car that is off the road is compatible with the state of any emergency vehicle.

4.2.2.4 Relationship between states and modes: mode invariants

The (current) mode of an element describes the action(s) that this element is currently undertaking, while an element's state describes its current situation, including its position. Modes can therefore be used to predict state evolution. For example, the state variable "position" of a car will remain constant as long as it is in the mode `stopped`. While an element is in a given mode, only some of its state variables might change, and these changes might be constrained. This is captured using *mode invariants*: predicates on state variables and their possible variations that remain true while an element is in a given mode. An example of an invariant on the variations of a state variable is: the state variable "position" will not vary by more than 5 m/s, hence capturing the maximum speed in that mode. Mode invariants capture semantic information about modes.

4.2.2.5 Goals and priority list

Entities have a *goal,* which is formalised as a condition on its state variables that the entity is aiming to achieve. All entities of a given type have the same goal, which can be parametrised. For example, the goal of all entities of type car might be to arrive at their destination, but each car has a specific destination. The goal of an entity can change over time, for example, when an initial goal is reached, but any entity has only one goal at a given time.

The goals of some entities might be more important than the goals of others. This is captured by the *priority list* of a scenario, which is an ordered list of modes, guarded by conditions on state variables, that capture the relative priorities of actions of entities in that scenario. The priority list therefore expresses the relative priority of achieving the goals of entities of different types. Modes can be omitted from the priority list, if none of them has a higher priority that all the others.

If, for a mode $m \in \mathrm{M}_i$, and a condition c on the state variables of a scenario, "if c then m" is used to denote the predicate "if c is fulfilled, then all elements of type E_i should be in mode m", the priority list of a scenario can be described as a tuple

$$(\text{if } c_1 \text{ then } m_1, \text{if } c_2 \text{ then } m_2, ..., \text{if } c_q \text{ then } m_q)$$

of predicates of decreasing priority, where $m_1, m_2, ..., m_q$ are modes of elements of this scenario and $c_1, c_2, ..., c_q$ are conditions on the state variables of the elements of the scenario. So for example, the

priority list of the emergency-vehicle warning scenario can be expressed as:

$$(\text{if } \textbf{true} \text{ then } \texttt{going\_at\_v}_{\text{p}},\ \text{if } \textbf{true} \text{ then } \texttt{going\_at\_v}_{\text{p-1}}, ...,\ \text{if } \textbf{true} \text{ then } \texttt{going\_at\_v}_{1},\ \text{if } \textbf{true} \text{ then } \texttt{travelling})\,,$$

which captures that the priorities of the scenario are first for the emergency vehicle to travel as fast as possible, and second, for the car to travel (note that the modes `stopped` of emergency vehicles and `out_of_the_way` of cars do not appear in the priority list because neither of them has a higher priority than the other). The **true** conditions can be omitted, and therefore the goal of the example can be expressed as:

$$(\texttt{going\_at\_v}_{\text{p}}, \texttt{going\_at\_v}_{\text{p-1}}, ..., \texttt{going\_at\_v}_{1}, \texttt{travelling})\,.$$

4.2.2.6 Incompatibilities

An incompatibility is a condition on the state variables of elements, which can be used to capture a state that must not occur. Four types of incompatibilities have been identified as having a particular interest for autonomous mobile entities:

SOV: a condition on a state variable of an element (composed of the State variable, a relational Operator, and a Value),

SOS: a condition on the relative values of two state variables of two elements (composed of a State variable, a relational Operator and another State variable),

distance: a condition on the distance between two positions,

cardinality: a condition on the cardinality of a set of entities that satisfy some condition on their state variables.

Examples for each of these incompatibility types are presented in Section 4.2.3. These incompatibilities can be combined using conjunctive and disjunctive logical operators. This allows a wide range of conditions on the states of entities in a scenario to be expressed. In particular, a condition on all entities of a given type can be expressed using a combination of incompatibilities of types SOV, SOS, and distance, while a condition on n entities of a given type can be expressed using a cardinality incompatibility that relies on a condition on their state variables.

4.2.2.7 Summary

This work aims to derive the requirements on entities' behaviours so that they can progress towards their goal, while ensuring system-wide safety constraints. The approach taken is that entities, while attempting to fulfil their goal, adapt their behaviour depending on the information currently available to them to ensure that the safety constraint is not violated. This idea is based on the rationale that

whether the safety constraint might be violated depends on the actions of entities (as well as their initial state, and the state of the environment). So, given a safety constraint, the set of allowable behaviours for an entity is restricted by the information currently available to it: it can only be in modes in which it knows that the safety constraints will not be violated given the information that it has. The priority list allows each entity to chose optimal behaviour from its set of allowable behaviours, so that it is nearing the goal as fast as possible.

4.2.3 Syntax

In addition to the behaviour of entities, the safety constraints also need to be modelled. Safety constraints may refer to the following elements:

- the state of any element of a given type, e.g., s_{car} (when the safety constraint needs to refer to several elements of the same type, these are distinguished by a number placed after the type name, e.g., $s_{\text{car 2}}$)
- the variables of a state, which can be expressed by the state name, a period and the variable name, e.g., s_{car}.mode
- a number of relational operators, which can be applied to the variables: $<$, $\leq$, $>$, $\geq$, $=$, $\neq$
- a number of basic incompatibilities, which can be of four types:
  1. SOV, e.g., s_{car}.speed ≤ 3
  2. SOS, e.g., s_{car}.speed $\leq s_{\text{ev}}$.speed
  3. distance, denoted distance(.,.), e.g., distance(s_{car}.position, s_{ev}.position)
  4. cardinality, denoted |{E, C}|<operator><value>, where E is an entity type, and C a condition on the state variables of entities of type E, e.g.,

$$|\{e_{\text{car}}, s_{\text{car}}.\text{mode} = \texttt{out\_of\_the\_way}\}| > 1$$

- two logical operators, which can link incompatibilities: $\wedge$, $\vee$.

The EBNF (ISO 1996) description of the language is presented in Figure 4.2.

4.2.4 Expressing the safety constraints

This formalism can be used to express the safety constraints of a scenario as a set of incompatibilities between states, including constraints on the relative distance of elements. For example, the safety constraint that cars and emergency vehicles should not collide can be stated as: for any state s_{car} of any entity of type "car" and any state s_{ev} of any entity of type "emergency vehicle",

$$\mathcal{C}_{\text{s}}(s_{\text{car}}, s_{\text{ev}}) \text{ iff } \neg\big((\text{distance}(s_{\text{car}}.\text{position}, s_{\text{ev}}.\text{position}) < d) \wedge (s_{\text{ev}}.\text{mode} \neq \texttt{stopped}) \wedge (s_{\text{car}}.\text{mode} \neq \texttt{out\_of\_the\_way})\big)\,.$$

```
incompatibility = ( incompatibility, "∧", incompatibility )
                | ( incompatibility, "∨", incompatibility )
                | ( element-type, ".", state-variable, rel-operator, value )
                | ( element-type, ".", state-variable, rel-operator,
                                              element-type, ".", state-variable )
                | ( "distance(", position, ",", position, ")", rel-operator, value )
                | ( "|", entity-type ".", state-variable, rel-operator, value "|",
                                                        rel-operator, value )
rel-operator = "<" | "≤" | ">" | "≥" | "=" | "≠"
```

Figure 4.2: EBNF description of the safety constraint formalism.

This expresses the fact that the safety constraints will not be violated if a car and an emergency vehicle are far enough away, or the emergency vehicle is stopped, or the car is out of the way of the emergency vehicle.

While being high-level and implementation-independent, this formalism captures all the salient details of the safety constraints. Furthermore, because it uses high-level abstractions, it is easy to use to express safety constraints. Note that this formalism does not allow the expression of any safety constraints (for example, safety constraints referring to the relative values of three variables, such as the height of entity e_1 plus the height of entity e_2 is smaller than the height of entity e_3 cannot be expressed using this formalism). The formalism does, however, capture safety constraints from a wide range of scenarios, and is sufficient to express the safety constraints of all the examples from the ITS domain that we studied.

4.2.5 Solvability

As already mentioned, the goal of this work is to derive some requirements on entities' behaviours so that they can progress towards their goal, while ensuring the system-wide safety constraint. Such a set of requirements, however, might not exist. A scenario is said to be *solvable* if there exists a set of requirements on the behaviours of entities such that the system-wide safety constraint is always ensured. A set of such requirements is called a *solution* of the scenario.

A solution, however, might not allow entities to make progress towards their goal. Whether a solution allows entities to make progress depends on two factors: (1) whether the conditions for an entity to transition to those modes that allow it to make progress will ever be met, and (2) whether deadlocks in the behaviour of entities can potentially happen, i.e., whether it might happen that while adapting their behaviour to progress towards their goal, a group of entities reach a situation in which none of them can make any more progress unless some of the others do. The first factor depends on the solution implementation, and in particular, on the characteristics of the technology used (wireless transmitter, actuators and sensors). The second factor depends on the complete behaviour of entities, i.e., not only the safety specification. None of these factors can be assessed at the safety design stage, as only the safety behaviour of entities is known at this stage. Therefore, this work focuses

on systematically producing a solution that will ensure that the safety constraint is not violated, and programmers must then assess whether these solution will allow entities to make progress with available technologies.

Solutions, however, can be evaluated within the model in terms of the hardware they require, the progress of which entities they favour, and the a priori feasibility of their requirements. Some scenarios have several solutions, and an evaluation makes it possible for model users to compare solutions, and chose the optimal solution for a specific application. This process is detailed in Chapter 5.

4.2.6 Decomposition of the safety constraints

A safety constraint is a condition on the state variables of elements that should never become true. This logical condition is composed of conjunctions and disjunctions of basic incompatibilities, which are simple conditions on the state variables of one or two elements. Boolean logic states that any logical condition can be decomposed in a disjunction of conjunctions, called the disjunctive normal form (Hazewinkel 1994). This allows us to decompose a safety constraint into a number of safety constraints composed only of conjunctions of basic incompatibilities. Therefore, we assume in the following that the safety constraint of the scenario is a conjunction of basic incompatibilities. Section 5.3 will show how the solutions of a scenario composed of several such safety constraints can be derived.

4.3 Safety constraint distribution

High-level system-wide safety constraints, while being simple and quite intuitive to state in this form, are not easily exploitable as such. In general, it is non-trivial to deduce the necessary and sufficient requirements on individual entities behaviour from such safety constraints, or even to check that some specification of the entities' behaviours ensures that these safety constraints will not be violated. To ease this process, this section introduces the concepts of responsibility, mode compatibility and three coordination primitives that can be used to derive requirements on entity behaviours.

4.3.1 Responsibility

For every possible incompatibility between the states of some elements, i.e., possible violation of the safety constraint, at least one of these entities needs to ensure that it does not occur. We say that this entity is *responsible* for the incompatibility.

The *role* of an entity is defined with respect to an interaction, as in object-oriented software engineering (Schelfthout & Holvoet 2005). The possible roles of entities of a given type are defined by model users. By default, entities only have a single (default) role, but if a safety constraint refers to several entities of the same type, they are distinguished by their role. So, for example, if the safety constraint in a scenario with autonomous cars states that two cars must not collide, i.e., for any two

cars $\mathrm{car}_{\mathrm{following}}$and $\mathrm{car}_{\mathrm{leading}}$:

$$\mathrm{s}_{\mathrm{car}_{\mathrm{following}}} \mathcal{C}_{\mathrm{s}} \mathrm{s}_{\mathrm{car}_{\mathrm{leading}}} \text{ iff } \neg\big(\mathrm{distance}(\mathrm{s}_{\mathrm{car}_{\mathrm{following}}}.\mathrm{position}, \mathrm{s}_{\mathrm{car}_{\mathrm{leading}}}.\mathrm{position}) < d\big)\,,$$

then cars might be distinguished depending on their role: 'following' or 'leading', which depend on their relative positions. Responsibility can be attributed to entities of a certain type or to entities in a certain role. For example, emergency vehicles might be responsible to ensure that they do not collide with cars or cars might be responsible for ensuring that no other car collides into them from behind, so cars in the leading car role are responsible for possible state incompatibilities with cars behind them. Responsibility might be attributed a priori or in real-time, and might be transferred. However, at any time, at least one entity must be responsible for each possible incompatibility. The problem of dynamic responsibility attribution relies on roles, and is not trivial, as consistent role selection is a hard problem, as illustrated, for example, in the context of multirobot teams (Weigel et al. 2002). As mentioned in Section 4.1, however, the problem of responsibility attribution is simpler than many problems of role attribution, as it is sufficient to have at least one entity in the role that is responsible, but is acceptable to have more than one. Programmers must specify an entity type and role that constitutes an initial partition of the entities so that there is an entity responsible for each combination of elements whose state might violate the safety constraint.

This notion of responsibility is the first step in the translation of system-wide safety constraints: it allows the duty of ensuring the safety constraint to be distributed over entities. Being responsible for an incompatibility implies requirements on the entity's behaviour: it should ensure at any time that the incompatibility does not happen. This requires that an entity be able to foresee when an incompatibility might happen. This can be deduced from the modes of the different elements. For this purpose, we define the notion of mode compatibility.

4.3.2 Mode compatibility

A set of modes $(m_{e_1}, m_{e_2}, ..., m_{e_n}) \in \mathrm{M}_{e_1} \times ... \times \mathrm{M}_{e_n}$ is *compatible* if, when some elements are simultaneously in these modes, their states are compatible. If we define, for $m \in \mathrm{M}_i$, $\mathrm{S}_{i,m}$ as the set of states of the element e_i, in which it is in mode m, i.e., $\mathrm{S}_{i,m} := \{s \in \mathrm{S}_i : \mathcal{M}_i(s) = m\}$, mode compatibility can be defined as:

$$\mathcal{C}_{\mathrm{m}}(m_1, m_2, ..., m_n) \text{ iff } \forall (s_{e_1}, s_{e_2}, ..., s_{e_n}) \in \mathrm{S}_{1,m_1} \times ... \times \mathrm{S}_{n,m_n}, \mathcal{C}_{\mathrm{s}}(s_{e_1}, s_{e_2}, ..., s_{e_n})\,.$$

For example, the modes `out_of_the_way` of a car and `going_at_vi` of an emergency vehicle are compatible because when they are in these modes, their states are always compatible.

While the notion of state incompatibility captures whether the safety constraints are being violated at a given time, mode compatibility enables us to make predictions that no incompatibility will happen (when elements are in these modes). Note that if the modes of a set of elements are not compatible, it does not imply that the safety constraints will be violated. For example, the modes `travelling`

Primitives	Meaning
Adapt	Perform an action other than the one planned
Delay	Perform a planned action later than initially planned
Transfer	Send a message to other entities, either via direct communication or via the environment

Table 4.1: The three coordination primitives and their meaning.

of a car and `going_at_Vi` of an emergency vehicles are not compatible, as entities might collide into each other when they are in these modes, but if they are far enough apart, the safety constraints will not be violated (and so their states at the time are compatible).

A mode m of an entity e is said to be a *fail-safe mode* if it is compatible with all the modes of all the other elements. This is noted $\mathrm{FSM}(m, e)$. It is sufficient for an entity to remain in a fail-safe mode to ensure that the incompatibility for which it is responsible will not happen. For example, the mode `stopped` is a fail-safe mode for emergency vehicles. Note that only entities whose actions might change the state variables mentioned in the safety constraint can have fail-safe modes (as the actions of other entities can never ensure that the safety constraint will not be violated).

We assume that both direct and indirect communication between entities can at times be completely impossible (see Chapter 3). Therefore, for a safety constraint never to be violated, responsible entities need to have fail-safe modes to which they can revert when communication is deficient. In particular, this implies that only entities which can influence the state variables mentioned in the safety constraint, can be made responsible for an incompatibility.

4.3.3 Coordination primitives

For a responsible entity to ensure that no state incompatibility will happen, it is sufficient to ensure that, at all times, its mode is compatible with the modes of all surrounding elements. An entity does not know, a priori, what modes the elements in its vicinity are or will be, and not even if there are any elements in its vicinity. The behaviour of entities is modelled in terms of actions they undertake (modes), and they can either continue the same action (remain in the mode, so delay changing to another mode), change action (i.e., change mode), or send messages (a special kind of action that we distinguish as it does not directly contribute to the goal). Three primitives have been defined to capture these possibilities: adapting its behaviour (i.e., change mode), delaying its action (i.e., delay changing mode), or transferring its responsibility (by sending messages to other entities if possible).

These primitives are detailed below and summarised in Table 4.1.

4.3.3.1 Adapting its behaviour

A responsible entity can have information about the modes in which other elements can be both a priori (by previous knowledge) and in real-time, via messages or sensor information. Using this information, a responsible entity can adapt its behaviour, i.e., enter a mode other than the one planned, to always

be in a mode in which the safety constraints will not be violated. It is sufficient for an entity to remain in a fail-safe mode to ensure that the incompatibility for which it is responsible will not happen. This primitive will be referred to as "Adapt".

4.3.3.2 Delaying actions

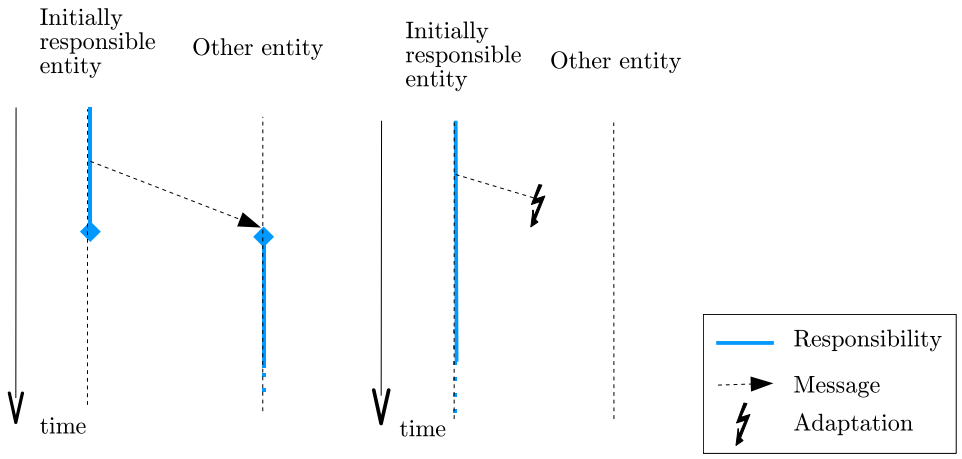
The second primitive consists of delaying an action that can trigger an incompatibility (i.e., delaying switching to a mode in which an incompatibility might occur). An entity can delay its action until it gets information that it is safe to undertake it, or until it has transferred its responsibility, as explained below. This coordination primitive will be referred to as "Delay".

4.3.3.3 Transfer

Another means for responsible entities to ensure that the incompatibilities for which they are responsible do not occur, is to warn other entities that the incompatibility might occur so that other entities can then change their behaviour to prevent the incompatibility. Because, as defined in Chapter 3, passive elements cannot receive messages, this coordination primitive can only be used for the coordination of entities, and not the coordination of entities and passive elements of their environment.

The warning can be given either by direct or indirect communication. Messages and signals might contain information about the responsible entity's state, mode, and intention. This can then be used by entities receiving a warning to optimise their reaction, i.e., to avoid the incompatibility while making as much progress as possible towards their goal. For direct communication, messages need to be sent periodically over a proximity that is big enough so that entities approaching receive a message early enough to be able to react to its contents if necessary. For indirect communication, signals need to be available for as long as the incompatibility can happen, and be perceptible in an area that is wide enough to ensure that entities will have time to react.

An entity sending a warning (either by direct or indirect communication) does not know whether any entity actually received the warning (c.f. Chapter 3). Therefore, entities that receive the warning become responsible to ensure that no incompatibility arises with the entity that sent it and other elements in its vicinity. This can be seen as a *transfer of responsibility*. This transfer is however only partial, as the responsible entity remains responsible for the incompatibility in relation to other entities. As shown in Figure 4.3(a), the transfer of responsibility is effective only if and when the message (or signal) is received. If the message is lost, the responsible entity remains responsible, as shown in Figure 4.3(b). Therefore this primitive relies on the communication models described in Chapter 3, and in particular, on the real-time feedback on the state of communication that they provide, so that entities can know whether a transfer has been successful. This primitive will be referred to as "Transfer".



(a) The message was received, therefore the responsibility is transferred.

(b) The message was lost due to an adaptation. Therefore the responsibility was not transferred.

Figure 4.3: Transfer primitive, and its effect on responsibility.

4.4 Translating safety constraints

In this section, we introduce the notions of contracts and zones, which build on responsibility, mode compatibility and the coordination primitives to translate the safety constraint into requirements on the behaviour of entities.

4.4.1 Contracts between elements

A responsible entity can use a combination of the three coordination primitives mentioned above to ensure that the incompatibility for which it is responsible does not occur. This must be decided a priori, and can be seen as an implicit contract between the responsible entity and other elements. This notion of contract is similar to the one used in software engineering and design by contract (Meyer 1992). Contracts between the elements regulate both the sending of signals and messages and the reception of them. Responsible entities need to have a contract with elements of every type mentioned in the safety constraint.

A contract needs to specify how and when entities should warn each other of possible incompatibilities. Therefore, contract parameters are:

- how early should an entity warn another entity that a possible incompatibility can happen,
- how intense should a signal or message be (e.g., loudness of a siren),
- how often messages are sent or sensing is done.

We have identified three types of contracts: (1) contracts without transfer, (2) contracts (with transfer) without feedback, and (3) contracts (with transfer) with feedback. The contract types differ in which

coordination primitives are used by responsible entities and other elements that are party to the contract. These contract types are described below, in order of increasing complexity: every contract described builds on the contracts previously presented and adds further possibilities. Each contract type is illustrated using the emergency-vehicle warning system example, in the case where emergency vehicles are responsible. In the contract description, the term "responsible entity" refers to the entity that is initially responsible (it might however, transfer its responsibility at a later stage). Moreover, the times mentioned are times of delivery (as opposed to times of sending of messages).

4.4.1.1 Contract without transfer

In a contract without transfer, the responsible entity does not transfer its responsibility, and must always ensure, by adapting its behaviour if necessary, that the safety constraints are not violated. Other elements do not need to know about the contract, or even of the existence of the responsible entity. This contract can be used between an entity and elements of any type (passive elements and entities). In this case, the coordination primitives that can be used are only Adapt and Delay, and they are used only by the responsible entity.

If such a contract is used in the emergency-vehicle warning example, emergency vehicles would drive relying only on sensor information to detect the presence of cars and foresee their behaviour. Therefore, the average speed of emergency vehicle would be slow because they are constrained by the normal behaviour of other traffic.

4.4.1.2 Contract without feedback

In a contract without feedback, the responsible entity must warn other entities when the safety constraints are liable to be violated. Other entities then become responsible to ensure that the incompatibility does not happen. This contract can only be used between entities, as it requires that participants be able to receive messages. Note that the responsible entity can also adapt its behaviour to ensure that an incompatibility does not happen. It must do so, in particular, when it cannot ensure that all entities will be warned early enough in advance.

The terms of the contract are the following: the responsible entity must ensure that other entities will receive a message of intensity i_{warning} at least a preagreed t_{warning} before an incompatibility can happen (or that no incompatibility can happen), and other entities must ensure that, at any time, they can avoid an incompatibility provided they are warned about it, at intensity i_{warning}, t_{warning} in advance. The time line for such contract is outlined in Figure 4.4. Entities must also agree on what communication means they will use: direct communication, indirect communication, or both. The requirements on entities' behaviours depend on the communication mean(s) used. In all cases, however, the responsible entity can use any of the three coordination primitives, while other entities can use only Adapt and Delay.

If this contract is used in the emergency vehicle example, emergency vehicles would warn cars of

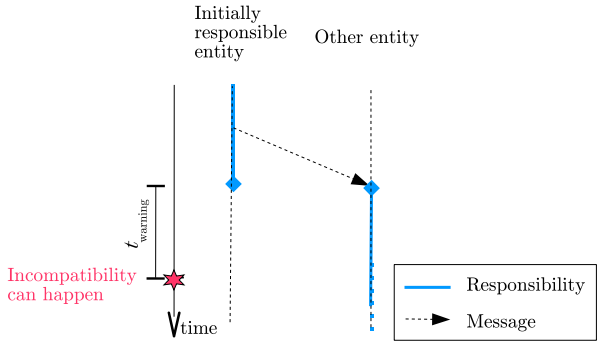


Figure 4.4: Time line for a contract (with transfer) without feedback.

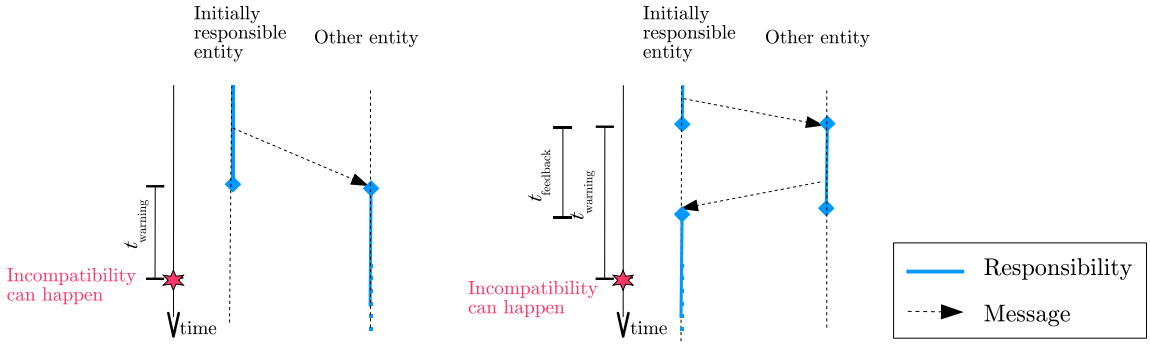
their arrival by sending them either a signal (e.g., by using a siren), or a message. Upon receiving such a warning, cars must get out of the way before the emergency vehicle arrives to ensure that the safety constraint will not be violated.

4.4.1.3 Contract with feedback

In a contract with feedback, the responsible entity must also warn other entities when an incompatibility can happen. In this case, however, entities that are warned that an incompatibility can happen can provide feedback to the responsible entity, by sending a message or a signal, when they cannot adapt their behaviour to avoid violating the safety constraint. This contract can only be between entities.

The terms of the contract include two preagreed time durations t_{warning} and t_{feedback}, and two intensities, i_{warning} and i_{feedback}. The responsible entity must warn other entities at least t_{warning} in advance when the safety constraints are liable to be violated, and also be able to react to their feedback within $t_{\text{warning}} - t_{\text{feedback}}$, and ensure that the incompatibility will not happen. Other entities must be able at any time either to react within t_{warning} to a transfer from a responsible entity, or give feedback within t_{feedback} to this entity when they are unable to safely remain responsible. Both possibilities are pictured in Figure 4.5 (a) and (b) respectively. Note that this contract might include the exchange of further messages, but after the initial exchange the entities have discovered each other's presence, and if necessary, the delay to exchange more messages can be included in the definition of t_{warning}. In this case, both responsible and other entities can use any of the three primitives. Note that contracts with feedback require the exchange of two messages when feedback is used, hence increasing the time cost, compared to a contract without feedback. Therefore, these contracts will be used in scenarios where it is expected that feedback should not be used often.

If this contract type is used in the emergency vehicle warning system, every emergency vehicle would warn cars of its arrival (by either signalling or sending messages), and cars receiving such warnings would try to get out its way, and if this is not possible, would send feedback to the responsible entity.



(a) when the other entity reacts to the transfer (b) when the other entity sends a feedback

Figure 4.5: Time lines for a contract (with transfer) with feedback.

4.4.1.4 Contract types summary

Contracts are used to characterise the protocol under which entities will interact: they define when an entity should change its behaviour to adapt it to these of another entity, when an entity should send a message to another entity, and when that other entity should itself send a message. The terms of the contracts for both responsible and other entities for each of the contract types are detailed in Table 4.2. These contracts need to be decided a priori, as is the semantic attached to responsibility transfer (e.g., what a siren means, or what are the meaning of message parameters).

More contract types could be defined, for example to distinguish between cases where a responsible entity uses both the Adapt and Delay primitives from cases where it uses only one of them. These three contract types, however, capture all the different possibilities in terms of interactions between entities (i.e., responsibility transfer), and it is the interactions between entities that influence the requirements on entities behaviour for ensuring system-wide safety constraints, so these three contract types are sufficient to design solutions.

The contract without feedback is actually a sub-case of the contract with feedback, where a consumer would never use the possibility of providing feedback. Opting for a contract without feedback will typically allow to chose a smaller contract parameter t_{warning}, as their is no need to schedule time for feedback. Contracts that entities have with elements must be decided a-priori. The use of the three primitives by both responsible entities (R) and other elements (O) in the three contracts is described in Table 4.3.

Note that safety constraints will be guaranteed only if the contract is respected. If it is believed that some entities might disregard safety constraints and not obey a contract, legislation might be introduced to enforce this. For example, it could be imposed that every autonomous car commercialised obeys a number of contracts necessary to ensure the safety of all road users.

Type of contract	Requirements on the responsible entity	Requirements on other elements	Type of elements the contract can be with
Without transfer	Adapt its behaviour or delay its actions to ensure that the incompatibility for which it is responsible will not happen. Can use both sensor information and messages received.		Elements
Without feedback	Warn other entities at least t_{warning} in advance, when the incompatibility for which it is responsible is liable to occur.	Be able to receive a warning from a responsible entity, and react to it within t_{warning}.	Entities
With feedback	Warn other entities at least t_{warning} in advance when the incompatibility for which it is responsible is liable to occur. Adapt to the feedback from another entity within $t_{\text{warning}} - t_{\text{feedback}}$.	Be able at any time to adapt within t_{warning} to a warning from the responsible entity, or to communicate within t_{feedback} to the responsible entity.	Entities

Table 4.2: Requirements imposed by the three types of contracts.

Contract	Adapt	Delay	Transfer
Without transfer	R	R	-
Without feedback	R, O	R, O	R
With feedback	R, O	R, O	R, O

Legend: R: responsible entity; O: other element.

Table 4.3: Use of the primitives by the contracts.

4.4.1.5 Transfer means

In the contracts with transfer, entities can use signalling, message passing, or both, to transfer their responsibility. The means to be used must be agreed upon by the entities; it is also part of their contract. This implies that there are actually three types of contracts with transfer without feedback. These are denoted Tx where x is either "d" for direct, "i" for indirect, or "di" for direct and indirect depending on the communication means used. The contracts with transfer without feedback are therefore

- using direct communication (Td),
- using indirect communication (Ti),
- using both direct and indirect communication (Tdi).

There are also 9 types of contracts with transfer with feedback, denoted TxFy, where x and y stand for the communication means used for the transfer and feedback respectively:

- using direct communication for both transfer and feedback (TdFd),
- using direct communication for transfer, and indirect communication for feedback (TdFi),
- using direct communication for transfer, and either direct or indirect communication for feedback (TdFdi),
- using indirect communication for transfer, direct communication for feedback (TiFd),
- using indirect communication for both transfer and feedback (TiFi),
- using indirect communication for transfer, and either direct or indirect communication for feedback (TiFdi),
- using both direct and indirect communication for transfer, and indirect communication for feedback (TdiFi),
- using both direct and indirect communication for transfer, and direct communication for feedback (TdiFd),
- using both direct and indirect communication for both transfer and feedback (TdiFdi).

In a contract Tdi, a transfer can be made via either direct or indirect communication. However, because other entities do not know which of the two transfer means will be used, they have to be able to react to either. In addition, as these other entities will be notified of any changes in the status of indirect communication, they will have to adapt their behaviour to such changes, as responsible entities might have used indirect communication. Therefore, a contract Tdi corresponds to a contract Ti where responsible entities rely on indirect communication, and direct communication can be used

	Responsible entity		Other entities	
	Direct communication	Indirect communication	Direct communication	Indirect communication
Contract without transfer Contract without transfer				
Contracts with transfer without feedback				
Td	✓			
Ti		✓		
Tdi	✓	✓		
Contracts with feedback				
TdFd	✓		✓	
TdFi	✓			✓
TdFdi	✓		✓	✓
TiFd		✓	✓	
TiFi		✓		✓
TiFdi		✓	✓	✓
TdiFd	✓	✓	✓	
TdiFi	✓	✓		✓
TdiFdi	✓	✓	✓	✓

Table 4.4: Different types of contracts and their use of communication means. A ✓ means that, in this contract, entities of the type described by the column use the communication means.

for optimisation. For example, an emergency vehicle warning system could be build upon a contract Tdi where emergency vehicles are responsible, and use sirens to warn cars of their arrival, but can also use direct communication if it allows them to go faster. Similarly, a contract TdiFy is equivalent to a contract TiFy, where direct communication can be used as an optimisation for the transfer, and TxFdi is equivalent to a contract TxFi where direct communication can be used as an optimisation for the feedback. The 13 types of contracts are summarised in Table 4.4.

4.4.2 Zones

The contracts are related to geographical zones: the safety and the consistency zone, as well as the critical coverage

4.4.2.1 Safety zone

By the definition of state compatibility, the states of all elements of a scenario must be compatible at all times in order to ensure the safety constraint. The safety constraint, however, actually imposes requirements only on specific states, typically when two or more elements are "close" according to some application-specific definition. For this reason, we define the *safety zone SZ* of an entity, as the

set of positions of elements where their states are liable to be incompatible with that of the entity. There is a safety zone per responsible entity/other entity type combination. The safety zone of the responsible entity e_{R} with regards to elements of type e_{O} is denoted $SZ(e_R, e_O)$.

Safety zones can be relative to an entity, or absolute and therefore the same for all entities (of the responsible entity type and other entity type combination). For example, the safety zone of a pedestrian traffic light is the pedestrian crossing, the safety zone of a car in a collision scenario is typically a zone around its "centre" that defines its body, and is relative to a car. In an unsignalised junction, however, the safety zone of a car is actually the area of the junction, and is therefore absolute, and common to all cars. We refer to the safety zone in either of these cases as "the entity's safety zone".

Note that the safety zone does not depend on the modes of the responsible entity or other entities. The safety zone of a responsible entity and another entity type combination can be expressed explicitly in the safety constraint by using the distance(.,.) incompatibility (see 4.2.3) or otherwise can be implicit, in which case programmers need to estimate its value. For example, the safety constraint in a collision avoidance scenario can be that cars should not be closer than a distance d, in which case the safety zone can be deduced directly from the safety constraint: it is a circle of size d around a car. The safety constraint of a pedestrian traffic light scenario might be that cars should not pass by the traffic light when it is red. In this example, the safety zone cannot be directly deducted from the safety constraint, but model users can assess it, as it is the area around the traffic light that should be free when it is red (i.e., the pedestrian crossing).

4.4.2.2 Consistency zone

If a responsible entity foresees that another entity could be in a state that is not compatible with its own state when that other entity enters its safety zone, the responsible entity can choose to transfer its responsibility, by sending a message or emitting a signal. In this case, it must do so early enough, so that the incoming entity will know about the possible incompatibility early enough to have time to adapt its behaviour (either by not entering the safety zone, or by changing its mode) to prevent the incompatibility. The time this will take depends on the mode that the responsible entity is in. The zone in which incoming entities must know of the responsible entity to be able to adapt to it is called the *consistency zone* of the mode m_{R} that the responsible entity is in, and noted $CZ(m_{\mathrm{R}})$. If m_{R} is a fail-safe mode, $CZ(m_{\mathrm{R}}) = 0$, as incoming entities never need an accurate view of the state of the responsible entity.

4.4.2.3 Critical coverage

If the responsible entity chooses to transfer its responsibility, to ensure that all incoming entities know the state of the responsible entity when entering $CZ(m_{\mathrm{R}})$, communication must be guaranteed in a zone $CC(m_{\mathrm{R}})$ around $CZ(m_{\mathrm{R}})$. This is called the *critical coverage* associated with the mode m_{R} of

the responsible entity. In the case of direct communication, the critical coverage is the coverage within which timely communication is required. In the case of indirect communication, the critical coverage is the coverage within which sensor information must be available. On failure of communication (i.e., when the critical coverage of its current mode is not covered), a responsible entity needs to adapt its behaviour, by entering a mode whose critical coverage is covered. If m_R is a fail-safe mode, $CC(m_{\mathrm{R}}) = 0$, as a responsible entity does not need to communicate when it is in a fail-safe mode.

4.4.2.4 Example

In the emergency-vehicle scenario, for example, the safety zone of emergency vehicles are the vehicles themselves. This can be approximated as a zone of diameter 3 m around their centre, that encompasses the entire vehicle. No car should ever enter an emergency vehicle's safety zone to ensure that they do not collide into it.

If emergency vehicles are responsible, every emergency vehicle needs to ensure that any car entering its consistency zone has an accurate view of its state, i.e., knows that it is arriving. This ensures that cars will have time to get out of the way after being warned of the arrival of an emergency vehicle.

In the case where emergency vehicles use a contract with transfer, every emergency vehicle warns cars of its arrival (transfer of responsibility) by either direct or indirect communication. To ensure that cars have an accurate view of its state before entering the consistency zone, an emergency vehicle needs to be able to send messages, via direct or indirect communication, in the critical coverage. If real-time communication is not guaranteed over the critical coverage, cars might not receive a message before entering the consistency zone, and therefore might not have time to react to avoid entering the safety zone, hence potentially colliding with the emergency vehicle. Therefore, entities notified of the communication coverage must adapt their behaviour when the communication is not guaranteed over the critical coverage. If direct communication is used, emergency vehicles will be notified of coverage changes (see Chapter 3) and should slow down when communication is not guaranteed over the critical coverage. If indirect communication is used, cars will be notified of sensing coverage changes, and should get out of the road when timely sensing is not guaranteed over the critical coverage (as they might not find out about an arriving emergency vehicle early enough to have time to get out of its way).

The different zones and their definitions are summarised in Figure 4.6. The next chapter investigates how the size of the different zones can be deduced from the contract parameters, hence making it possible to derive requirements on the behaviour of entities.

4.5 Summary

In this chapter, we have defined Comhordú, a real-time coordination model for autonomous mobile entities that builds on the environment, direct communication, and sensing and indirect communication

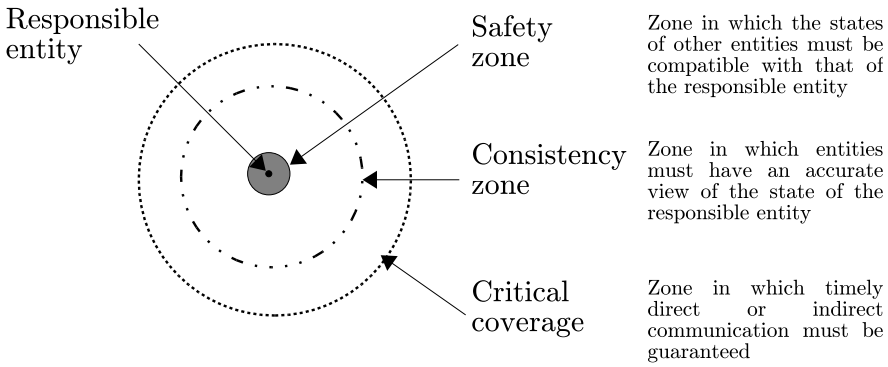


Figure 4.6: Definitions of the different zones within the critical coverage.

models defined in the previous chapter. Comhordú uses the notions of modes and states to formalise a scenario and specify a safety constraint. Such a safety constraint can then be distributed amongst entities using the notions of responsibility, mode compatibility and some coordination primitives. This distribution translates into contracts between entities, which can be used to derive requirements on the behaviour of entities. In the next chapter, we show how to design a solution using Comhordú, and how to derive the set of solutions for a given scenario. In addition, we also show how scenarios can be combined.

Chapter 5

Using Comhordú to Derive Requirements on Entity Behaviour

Using the Comhordú model defined in the previous chapter, programmers can specify safety constraints and possible interactions amongst entities. This chapter shows how these specifications can be used to systematically derive requirements on the behaviour of entities so that they ensure the safety constraints. In particular, we show how developers can choose the appropriate contract type and responsible entity in Section 5.1. We then present the approach used to distribute the safety constraints, and detail the requirements on entities that can be derived for each contract type in Section 5.2. In addition, we show how the requirements on entity behaviours can be derived for combinations of safety constraints, and how scenarios can be combined in Section 5.3. Finally, Section 5.4 summarises and concludes this chapter.

5.1 Designing a solution

To ensure system-wide safety constraints, entities must coordinate their behaviour. This requires them to interact, and their interactions are captured by contracts. The contracts that an entity must fulfil impose requirements on its behaviour. As defined in Chapter 4, a solution of a scenario is a set of requirements on the behaviours of entities that ensure that the system-wide safety constraint of this scenario is always respected. The entities that need to interact are the ones mentioned in the safety constraints. If several entities of the same event type are mentioned in the safety constraint, the way they interact can differ depending on their role. A solution is specified entirely by a combination of a responsible entity, and a tuple of contract types, with one contract type for each entity type and role mentioned in the safety constraint. (Note that the initially responsible entity is the same for all of these contracts.) To simplify the explanations, we will refer to such a combination simply as "a combination" in the following. As explained in Chapter 4, passive elements cannot be made

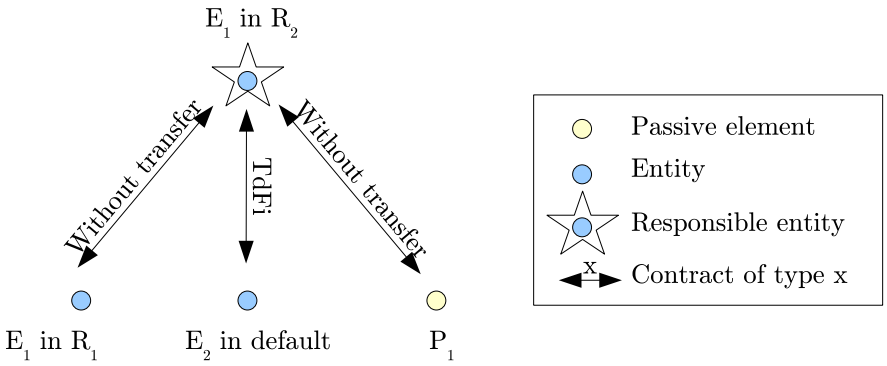


Figure 5.1: Responsible entity and contract types combination example.

responsible and the only contract type that can be used with them is a contract without feedback.

To illustrate the definition of combinations, consider the example of a scenario with a safety constraint mentioning entities of type E_1 with two roles R_1 and R_2, entities of type E_2, and a passive element of type P_1. In this scenario, a combination is a responsible entity type and role (either E_1 in R_1, or E_1 in R_2, or E_2), and a tuple of two contract types, between responsible entities and entities of the other two entity types and roles. (The contract between the responsible entity type and role and the passive element is a contract without transfer and does not need to be specified.) An example of a combination is: $\langle E_1$ in R_2, contract without transfer with E_1 in R_1, contract with feedback TdFi with $E_2\rangle$, as illustrated in Figure 5.1. The set of all combinations can be obtained by varying the responsible entity, and all the contract types. In the example scenario, there is a choice of 3 possible responsible entities, and two contracts can be picked independently of a choice of 13 (see Table 4.4), so there are $3 \cdot 13 \cdot 13 = 507$ combinations.

In the following, we first detail the heuristics used to derive the set of solutions, that is, to prune the set of combinations of those that cannot ensure that the safety constraint will not be violated. In the second part, we detail how the remaining solutions can be evaluated and compared.

5.1.1 Deriving the set of solutions

This section explains how the set of combinations of a scenario can be pruned of the combinations that contain contracts that entities cannot obey. A similar problem has been studied in the MAS community, in the form of the correctness of commitment protocols (Yolum 2005). Commitments, however, can be revoked. Adapting the result of (Yolum 2005) to contracts (i.e., removing the possibility for either entities bound by a contract to revoke it), leads to the result that for a contract to be a solution, at every step in its execution, entities that have a commitment (i.e., are responsible) must be able to either fulfill it or transfer it. This intuitive result is used in the following. First, two cases where the behaviour of entities is constrained are identified, and then, for each of these cases, the constraints on entities are identified. Finally, a process that allows the assessment of the contracts that entities cannot fulfil is presented.

	Communication	
	Sufficient	Not sufficient
State		
Compatible	No need to adapt	Sender if direct communication is used, receiver if indirect.
Not compatible	Entities that are responsible	Sender if direct communication is used, receiver if indirect.

Table 5.1: Summary of which entities have their behaviour constrained when the communication is sufficient, or not and when the behaviour of entities is compatible or not.

5.1.1.1 Two cases where entities' behaviour is constrained

To ensure that a safety constraint will be respected, responsible entities need to ensure that their state remains compatible with that of other entities, by either adapting their behaviour or transferring their responsibility if possible. Consider the example of an emergency-vehicle warning system, where the safety constraint is that emergency vehicles should not crash into ordinary vehicles. If emergency vehicles are responsible and use a contract with transfer, emergency vehicles can either adapt their behaviour, i.e., remain stopped or they need to ensure that they warn other vehicles before they arrive, by sending them messages, hence transferring responsibility.

When communication is degraded, however, responsibility cannot be transferred. For this reason, when a contract with transfer is used, the behaviour of entities notified of communication degradations is constrained because they have to ensure that the safety constraint is respected. Depending on the communication means used, either entities that transfer their responsibility or others are notified of communication degradations and have to adapt their behaviour. If emergency vehicles, for example, use a contract with transfer via direct communication, every emergency vehicle will be notified when the area in which its messages are delivered decreases, and will need to adapt its behaviour by slowing down in these cases. If, however, emergency vehicles use a contract with transfer via indirect communication, each ordinary vehicle will be warned when its indirect communication coverage decreases, and will have to ensure that the safety constraints are respected by adapting its behaviour (i.e., getting off the road).

To summarise, there are two possible cases where the behaviour of entities is constrained: when they are responsible, and their state might become incompatible with that of other entities, and when they are notified that communication is degraded. Note that these two cases are not exclusive: an emergency vehicle might be notified that the area in which it sent messages is not wide enough and therefore slow down, while, at the same time an ordinary vehicle might have received one of these messages and hence has become responsible and will get out of the way. As both entities will react by entering a fail-safe mode, however, the safety constraints will still be ensured. This discussion is summarised in Table 5.1.

5.1.1.2 Obeying the constraint for states to remain compatible

The behaviour of responsible entities is constrained because they need to ensure that their state remains compatible with that of other entities. In addition, their behaviour is constrained by the coordination primitives allowed by the contracts to which they obey. In particular, if the only coordination primitives that an entity can use are Adapt and Delay, then entities need to have a fail-safe mode that they can be in when the safety constraints may be violated.

This is the case of responsible entities in a contract without transfer, of other entities in a contract with transfer without feedback, and of responsible entities in a contract with feedback (see Table 4.3 on page 71). This is summarised in the second column of Table 5.2.

5.1.1.3 Obeying the constraint to ensure safety when communication is degraded

In case of degraded communication, entities that might transfer their responsibility via direct communication need to adapt their behaviour. This is the case of responsible entities in the contracts Td, TdFd, TdFi, TdFdi, and other entities in the contracts TdFd, TiFd, and TdiFd. Similarly, in case of degraded indirect communication, entities that can receive a transfer of responsibility are notified and need to adapt. This is the case of other entities in the contracts Ti, Tdi, TiFd, TiFi, TiFdi, TdiFd, TdiFi, TdiFdi (remember that contracts using both communication means rely on indirect communication, and use direct communication as an optimisation, as discussed in 4.4.1.5), and of responsible entities in the contracts TdFi, TdFdi, TiFi, TiFdi, TdiFi, TdiFdi. These results are summarised in the third column of Table 5.2.

5.1.1.4 Using the constraints to derive the solutions

Because entities can be surrounded by many other entities, and can receive several transfers at the same time, to be able to cater for the worst case, entities who need to adapt their behaviour need to be able to switch to a fail-safe mode within the contract parameter t_{warning}, so in particular, this requires them to have a fail-safe mode in the first place. Therefore, contracts impose that some entities have fail-safe modes. Whether responsible entities and other entities need to have fail-safe modes depends on the contract type used; this is summarised in the fourth and fifth columns of Table 5.2 respectively.

The fact that some contract types impose that some entities have fail-safe modes can be used to detect which combinations contain contracts that cannot be obeyed. The combinations that use contracts that would require fail-safe modes for entities that do not have any are therefore not solutions of the scenario, and can be pruned of the set of all possible combinations. If the combination set is empty at this stage, then the scenario is not solvable. Otherwise, it is solvable, and all remaining combinations are solutions of the scenario, because the safety constraint will be guaranteed, as entities have fail-safe modes in which they can be when their state might become incompatible with that of other entities, and they will be required to be able to transition back to one of these fail-safe modes within t_{warning} whenever they are in a non fail-safe mode.

	Entity type that has to adapt		**Need a fail-safe mode**	
	When behaviour not compatible	When not enough information	Responsible	Other
Contract without transfer	Responsible entity	Responsible entity	✓	
Contracts with transfer				
Td	Other entity	Responsible entity	✓	✓
Ti	Other entity	Other entity		✓
Tdi	Other entity	Other entity		✓
Contracts with feedback				
TdFd	Responsible entity	Both	✓	✓
TdFi	Responsible entity	Responsible entity	✓	
TdFdi	Responsible entity	Responsible entity	✓	
TiFd	Responsible entity	Other entity	✓	✓
TiFi	Responsible entity	Both	✓	✓
TiFdi	Responsible entity	Both	✓	✓
TdiFd	Responsible entity	Other entity	✓	✓
TdiFi	Responsible entity	Both	✓	✓
TdiFdi	Responsible entity	Both	✓	✓

Table 5.2: Entities that need to adapt and have fail-safe modes for the different contract types. A ✓ means that entities of the type described by the column need to have a fail-safe mode to which they can transition within t_{warning}.

5.1.1.5 Example

Consider the example of a traffic light at a road crossing, where the safety constraint is that no car should pass if the light in its direction is red. The behaviours of traffic lights can be modelled with the modes `green_in_direction_1` (where the traffic light is red in direction 2), `green_in_direction_2` (where the light is red in direction 1), as well as `switching_to_green_in_direction1` and `switching_to_green_in_direction2`, that correspond to the colour changes. The behaviour of cars can be modelled with the modes `stopped`, `travelling` and `stopping`. The mode `stopped` of cars is a fail-safe mode, as it is sufficient for cars to remain stopped to ensure that the safety constraint is not violated. The traffic light, however, has no fail-safe modes as the safety constraints can be violated when it is in either of the modes.

A combination for this scenario corresponds to a responsible entity type, and one contract type. As this scenario contains two entities types and that there are 13 possible contract types (see Table 4.4), the set of combinations for this scenario contains 26 combinations. As traffic lights do not have a fail-safe mode, however, they can only obey the contracts that do not require a fail-safe mode. So, the only combinations remaining are:

- ⟨car, contract without transfer with traffic lights⟩,
- ⟨car, contract TdFi with traffic lights⟩,
- ⟨car, contract TdFdi with traffic lights⟩,
- ⟨traffic light, contract Ti with cars⟩,
- ⟨traffic light, contract Tdi with cars⟩,

these are the solutions for this scenario.

5.1.2 Evaluating the set of solutions

Using the heuristics detailed in the previous section, the set of possible solutions is derived. This is the set of all combinations that ensure that the safety constraints will not be violated. However, these solutions do not guarantee that entities will ever make any progress towards their goal. In this section, we show how to identify solutions in which entities will never be able to make progress, and discuss the criteria to evaluate and compare other solutions.

Note that we are interested in scenarios where entities need to enter non fail-safe modes in order to reach their goal. Otherwise, an entity can always remain in one of its fail-safe modes, and so the solution to guaranteeing the safety constraint is trivial.

5.1.2.1 Influence of the predictability of the state variables used in the safety constraint

A safety constraint is a condition on the state variables of elements. To predict whether a safety constraint will be violated, an entity needs to be able to predict the evolution of at least some of the state variables used in the safety constraint over time and also over distance if any of the elements is mobile. As explained in 4.2.6, we can assume that the safety constraint is a conjunction of basic incompatibilities.

For entities to have a fail-safe mode, they need to be able to ensure that the safety constraint will not be violated, and therefore they need to have control over, i.e., be able to influence the value of, at least one of the state variables mentioned in the safety constraint. However, as pointed above, for the scenario to be non-trivial, entities must need to switch to a non fail-safe mode to reach their goal. To ensure safety while being in a non fail-safe mode, responsible entities need to be able to make predictions about the evolutions of at least some of the state variables mentioned in the safety constraint that it cannot control.

Being able to make predictions about the evolution of a state variable requires that the entity either controls the variable, has a priori information, or has a contract with the entity that controls it to ensure that it will issue a warning before changing its value (and not change unless it knows the warning was received if the warning is via direct communication). If we consider the example of a pedestrian traffic light, where the safety constraint is that cars should not drive through a red light, if the light can turn from green to red at any time, cars cannot safely move (and therefore, cannot safely make any progress towards their destination). If however, cars know the timing of the traffic light sequence, they can deduce the future state of the light from previous observations. Alternatively, cars can also progress safely if they have a contract with traffic lights in which they will be warned in advance of the light turning to red (typically, by the traffic light turning to amber).

This criterion therefore states that for an entity to make progress, it needs to have a priori or real-time information sufficient for it to know that over some time, a basic incompatibility will not be violated. In particular, as entities do not communicate in a contract without transfer, this means that an entity cannot make progress using such a contract, unless it can predict at least some of the variables mentioned in the safety constraints that it does not control. This allows some of the solutions to be discarded, because they will not allow entities to make progress. For example, in the pedestrian traffic light scenario introduced earlier, the combination: ⟨car responsible, contract without transfer with traffic lights⟩ is a solution of the scenario (because stopped cars will not violate the safety constraint, and therefore they can ensure the safety constraint), but, unless cars have a priori information about the light, e.g., that the light will be green every even minute, this combination will not allow cars to make progress toward their goal, and can therefore be discarded.

5.1.2.2 Influence of the priority list

The choice of contract also influences which entities will be able to make progress, both when there is enough and when there is not enough information.

In a contract without transfer, if the communication is good, responsible entities can make progress only when they know that the behaviour of other entities is, and will remain, compatible with theirs, therefore their progress is constrained. If the communication is degraded, the progress of responsible entities is very constrained as they may not know anything about the behaviour of other entities and must therefore remain in a fail-safe mode. The progress of other entities in a contract without transfer, however, is unconstrained, as they never have to adapt their behaviour.

In a contract with transfer, when the communication is good, other entities should adapt their behaviour when they receive a transfer, therefore their behaviour is constrained. The behaviour of responsible entities is intermittently constrained, as they need to warn other entities of incompatibilities, and therefore delay some mode changes until other entities have been warned. If the transfer is via direct communication (contract Td), when communication is degraded, the progress of responsible entities is very constrained as they may not know anything about the behaviour of other entities and must therefore remain in a fail-safe mode. The progress of other entities in this case is unconstrained, as they will never receive a transfer, and will never need to adapt. If the transfer is via indirect communication (as in contracts Ti and Tdi, since in the Tdi contract, direct communication is used only as an optimisation, as explained in 4.4.1.5), other entities must adapt their behaviour when communication is degraded, so their behaviour is very constrained, while the behaviour of responsible entities is still intermittently constrained, as they behave in the same way as if there is good communication.

Achievable results for the contract without transfer and contracts with transfer without feedback are depicted in Table 5.3.

For contracts with feedback, because it is expected that feedback will be used only a small proportion of the times that responsibility is transferred (see Section 5.2.4), the achievable progress is the same as for a contract without feedback using the same communication means for transfer, except that the progress of responsible entities is slightly degraded because they have to react to possible feedback.

The priority list can be used to derive the relative importance of the progress of different entity types when communication is sufficient. This criteria can be used to rank the different solutions. The desired behaviour might be different when communication is deficient, however, and therefore developers must also specify what the priority is in this case, and decide what weight to give to each criterion.

5.1.2.3 Other criteria

In addition to the progress criteria (both when communication is sufficient, and when it is not), other parameters can be used to evaluate the different solutions: the required equipment, as well as the

	Entity type that has to adapt		Progress	
	When behaviour not compatible	When not enough information	When enough information	When not enough information
Contract without transfer	Responsible entity	Responsible entity	R: ★★ O: ★★★★	R: ★ O: ★★★★
Contracts with transfer				
Td	Other entity	Responsible entity	R: ★★★ O: ★	R: ★ O: ★★★★
Ti	Other entity	Other entity	R: ★★★ O: ★★	R: ★★★ O: ★
Tdi	Other entity	Other entity	R: ★★★ O: ★★	R: ★★★ O: ★

Table 5.3: Results achieved by contracts without transfer and with transfer without feedback. R stands for responsible entity, and O for other entity. The following rating is used to qualify the entities' progress: ★ very constrained, ★★ constrained, ★★★ intermittently constrained, ★★★★ unconstrained.

likelihood that requirements for entering non fail-safe modes, and therefore making progress, will be met. While the latter criteria depends on the implementation, it can be noted that, all other things being equal, the requirements for contracts without transfer are more likely to be met than requirements for contracts with transfer, which are themselves more likely to be met than requirements for contracts with transfer with feedback. This is due to the fact that the more communication that is required, the longer in advance entities must be warned. This observation can be used to evaluate the a priori feasibility of the requirements.

5.1.3 Conclusion

We have shown in this section how some solutions can be dismissed and others be evaluated. The specific ranking of solutions depends on the weight that model users give to the different criteria: priority lists for when communication is good and for when it is degraded, required equipment, and likelihood of progress. The general process of finding the set of solutions and ranking them is summarised in Figure 5.2.

5.2 Deriving the requirements

In this section, we first present the general approach taken to ensure that the safety constraint is not violated, in Section 5.2.1. Requirements on entities' behaviour depend on the type of contract used; we cater for each contract type successively in Section 5.2.2, Section 5.2.3 and Section 5.2.4.

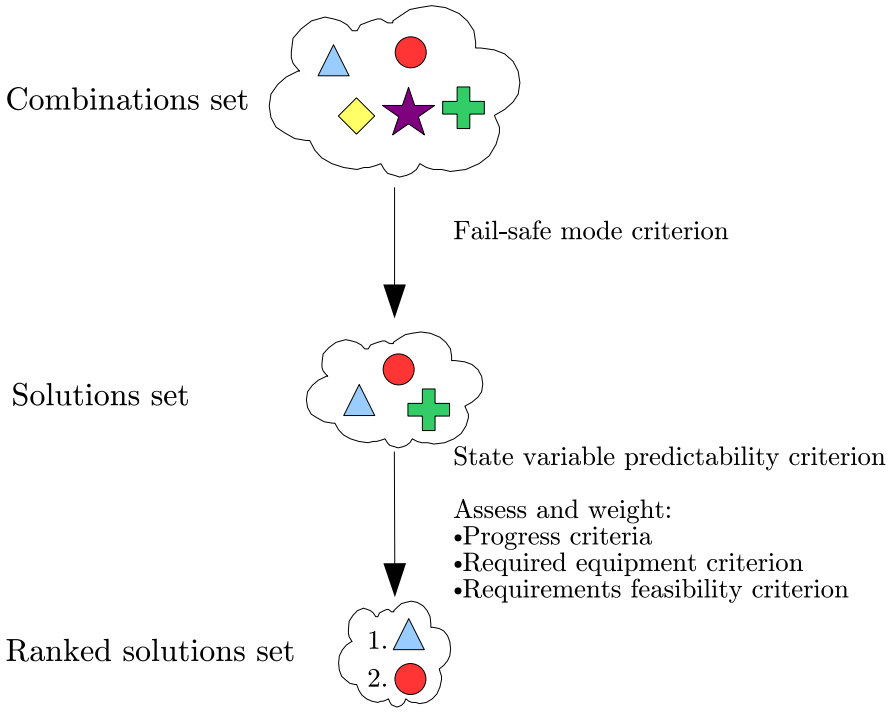


Figure 5.2: Summary of the solution design process.

5.2.1 General approach

A safety constraint is violated when the states of some elements are not compatible. By definition of the safety zones, and provided that there is a partition of responsible entities, so that at least one of them is responsible for every possible incompatibility that can arise, this will only happen when some element is in the safety zone of a responsible entity. Figure 5.3 shows an entity e_{O} entering the safety zone $SZ(e_{\mathrm{R}}, e_{\mathrm{O}})$ of a responsible entity of type e_{R}, while another entity e'_{O} is already in the safety zone with e_{R}. The entity e_{R} must avoid that the states of itself, e_{O} and e'_{O} violate the safety constraint.

The safety constraint can be violated only if at least one element e_{O} is in the safety zone $SZ(e_{\mathrm{R}}, e_{\mathrm{O}})$, while in a mode that is not compatible with the modes of other elements that are in the safety zones of e_{R}. This can be caused by either e_{O} entering the safety zone of e_{R} while being in a mode that is not compatible with the mode of e_{R}, or some entity in the safety zones of e_{R} changing mode while e_{O} is in the safety zone of e_{R}. (Note that when we say that e_{O} enters the safety zone of e_{R}, it does not imply that e_{O} moves, merely that it moves relatively to e_{R}, for example because e_{R} moves.)

A fault tree illustrating this reasoning is depicted in Figure 5.4. This shows that it is sufficient to ensure that the event B1 will not occur to ensure that the safety constraint will not be violated. This illustrates that instead of avoiding the states of elements becoming incompatible, the approach taken here is to ensure that the modes of elements are always compatible when any of them is in the safety zone of the other. Reasoning about mode compatibility instead of state compatibility allows to

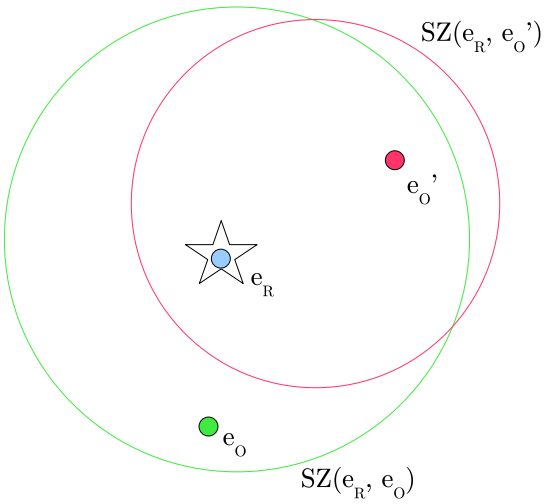


Figure 5.3: A responsible entity and some of its safety zones.

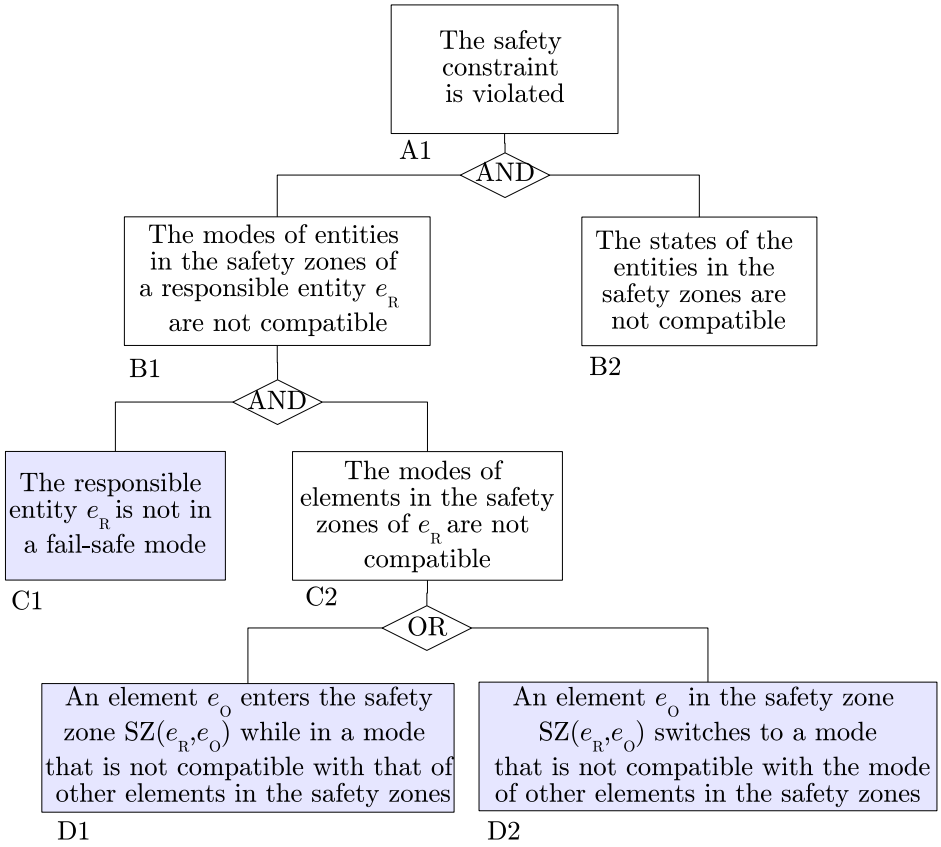


Figure 5.4: Fault-tree for a violation of the safety constraint.

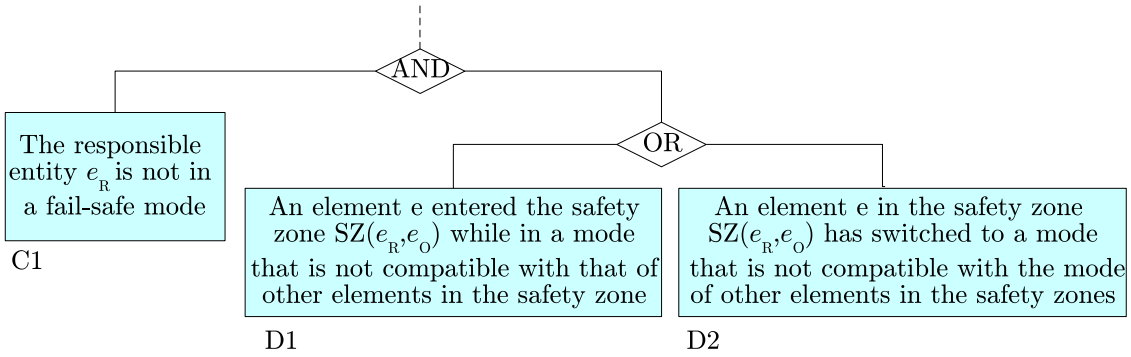


Figure 5.5: Fault-tree for a violation of the safety constraint for a contract without transfer.

anticipate when a safety constraint might be violated, and to prevent it.

The event B1 will not occur unless C1 and either D1 or D2 occur. The following subsections show how it can be ensured that C1 and either D1 or D2 will not occur, for each of the different contract types.

5.2.2 Contract without transfer

A contract without transfer can be applied between two entity types (or entities of same type in different roles) or an entity type and an element type. In the case of a contract without transfer, only the responsible entity will act to avoid incompatibilities. It has to ensure that it adapts its behaviour so that no other entity enters its safety zone or that when they do, the modes of all entities in its safety zones are compatible. This can be achieved by using both a priori known information and information obtained in real-time, by messages and sensors.

A responsible entity e_R using a contract without transfer needs to ensure at any time that the safety constraint will not be violated. As explained in the previous section, and summarised on Figure 5.5, , it is sufficient for e_R to be in a fail-safe mode for this purpose. If e_R is not in a fail-safe mode, however, it needs to ensure that neither event D1 nor event D2 occur. To prevent D1 from occurring, e_R needs to be able to be made aware of incoming elements early enough to be able to react to them. In the worst case, the reaction to an incoming entity is to revert to a fail-safe mode. Therefore, e_R needs to know about any elements in a zone of size

$$CZ(m_R) = SZ + R\_reaction(m_R) \cdot v_{max}(m_R)\,, \tag{5.1}$$

where m_R is the mode of the responsible entity e_R, $R\_reaction(m_R)$ is the maximum time necessary for e_R in mode m_R to revert to a fail-safe mode, and $v_{max}(m_R)$ the maximum (relative) speed at which elements might approach e_R when it is in mode m_R. According to our sensor and indirect communication model (described in Chapter 3), it might take up to *present* for e_R to be able to start sensing, and then up to *period* to actually sense, and *latency* to receive the data, so this requires that

e_{R} be able to sense an incoming element on a zone

$$CC(m_{\mathrm{R}}) = CZ(m_{\mathrm{R}}) + (present + period + latency) \cdot v_{\max}(m_{\mathrm{R}})\,. \tag{5.2}$$

Furthermore, to ensure that D2 does not occur, the requirements on the behaviour of e_{R} are that, when it senses an element e_{O} arriving, it needs to ensure that it will be in a mode that is compatible with that of e_{O} before e_{O} enters its safety zone. Lets call $\tau(e_{\mathrm{O}})$ the set of modes that e_{O} can be in while in the safety zone of e_{R}. In the case where e_{R} knows only of the presence of e_{O}, e_{O} can be in any mode, so $\tau(e_{\mathrm{O}}) = M_{\mathrm{O}}$, where $\mathrm{M_O}$ is the set of modes of element e_{O}; this means that e_{R} needs to revert to a fail-safe mode. If, however, e_{R} has information about the modes that e_{O} is in and might transition to, then $\tau(e_{\mathrm{O}})$ is a subset of $\mathrm{M_O}$. In this case, e_{R} must ensure that it is in a mode that is compatible with the modes of all the elements that are within its safety zone, as well as all those to which they can transition while being there. This can be expressed as e_R must ensure, at any time, that its mode m_{R}verifies $m_{\mathrm{R}}\mathcal{C}_{\mathrm{m}}\{\tau(e_{\mathrm{O}}),\ e_{\mathrm{O}}\ \text{in}\, CZ(e_{\mathrm{R}}, e_{\mathrm{O}})\}$.

A responsible entity can use both a priori and real-time information to derive the content of the set $\tau(e_{\mathrm{O}})$. In particular, if its knows that the modes of entities of the type of e_{O}vary according to an a priori-known pattern this can be exploited to deduce what modes e_{O} can be in while it is in its safety zone. Similarly, mode invariants, and in particular invariants that express that the variations of some state variables are bounded, allow to make predictions about the future modes that entities can be in.

While non responsible elements do not have to do anything to prevent an incompatibility, there are still requirements on their behaviour in the form of a bounded moving speed. Given these parameters, the responsible entity can derive the amount of information required to be in a given mode, i.e, the zone it needs to be able to sense.

5.2.3 Contracts without feedback

In the case of a contract with transfer without feedback, the responsible entity needs to warn other entities, at intensity i_{warning}, at least t_{warning} in advance when an incompatibility could happen, i.e., the safety constraint will be violated. For this purpose, it can use either direct or indirect communication. In either case, it needs to ensure that the message will be received in an area that is wide enough so that entities will be warned t_{warning} before the incompatibility can happen (or ensure that no incompatibility can happen).

An incompatibility can happen when an entity enters the safety zone of a responsible entity, or when, once an entity is in the safety zone of a responsible entity, either of the entities in the safety zones of the responsible entity changes its mode. Therefore, as illustrated in Figure 5.6, to ensure that the safety constraints will not be violated, the following constraints must be met:

C1 Entering a non-fail-safe mode: when a responsible entity intends to enter a non fail-safe mode, it must warn entities, at intensity i_{warning}, at least t_{warning} in advance, because an incompati-

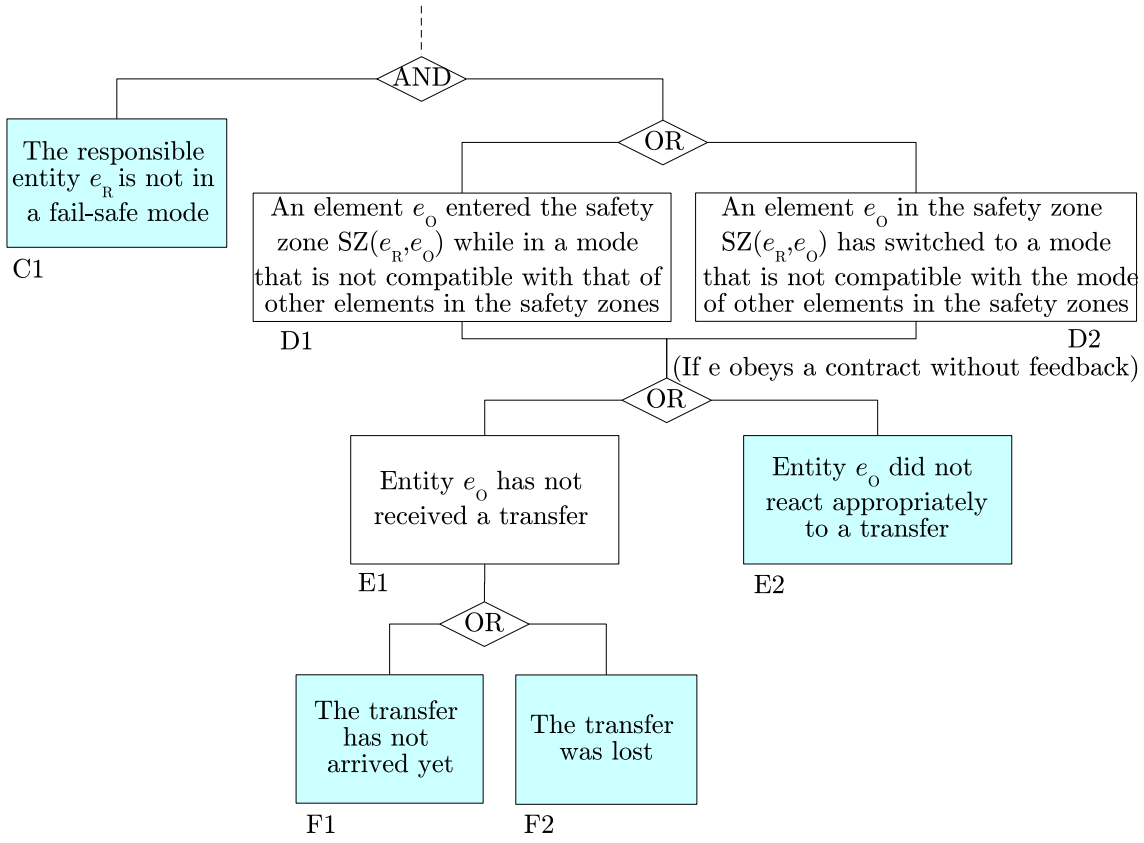


Figure 5.6: Fault-tree for a violation of the safety constraint for a contract without feedback.

Constraint	Name	Entity affected
C1	Entering a non-fail-safe mode	Responsible entity
F1	Being in a non-fail-safe mode	Responsible entity
F2	Communication degradation	Entities receiving feedback about communication
E2	Reacting to a transfer	Other entity

Table 5.4: Constraints for a contract without feedback.

bility can happen when it enters that mode. This implies warning entities that will enter the consistency zone, but also the ones within the safety zones.

F1 Being in a non-fail-safe mode: when the responsible entity is not in a fail-safe mode, it must warn other entities, at intensity i_{warning}, at least t_{warning} before they enter the safety zones,

F2 Communication degradation: when the entities cannot communicate, at least one of them will be notified (the entity sending a message in the case of direct communication, the entity sensing in case of indirect communication). An entity receiving feedback that communication is not sufficient must ensure that its mode is compatible with those of entities that might enter its safety zones.

E2 Reacting to a transfer: other entities need to be able to react to a transfer, at intensity i_{warning}, from a responsible entity about a possible incompatibility, within t_{warning}, so that the incompatibility cannot happen.

The constraints are summarised in Table 5.5. In the following we identify the requirements implied by these constraints depending on the communication means used for the transfer of responsibility: direct communication, indirect communication, or both.

5.2.3.1 Transfer via direct communication

Constraint F1: Being in a non-fail-safe mode To ensure F1, incoming entities of some type e_{O} must know of the responsible entity early enough to have time to adapt their behaviour. This implies that they must know of the responsible entity e_R before entering its consistency zone of size:

$$CZ(m_R) \geq SZ + t_{\text{warning}} \cdot v_{\max}(m_{\text{R}})\,, \tag{5.3}$$

where m_{R} is the mode of the responsible entity, SZ the safety zone of the responsible entity e_{R} and incoming entities of type e_{O}, t_{warning} is the parameter defined in the contract between entities of types e_{R} and e_{O}, and $v_{\max}(m_{\text{R}})$ the maximum (relative) speed at which entities of type e_{O} might approach an entity of type e_{R} that is in mode m_{R}.

If the responsible entity uses direct communication, it needs to send messages at intensity i_{warning}, in a zone $CC(m_{\text{R}})$ around itself wide enough to allow any incoming entity to receive them before entering the consistency zone. If an entity e_{O} enters the critical coverage of a responsible entity e_{O} at

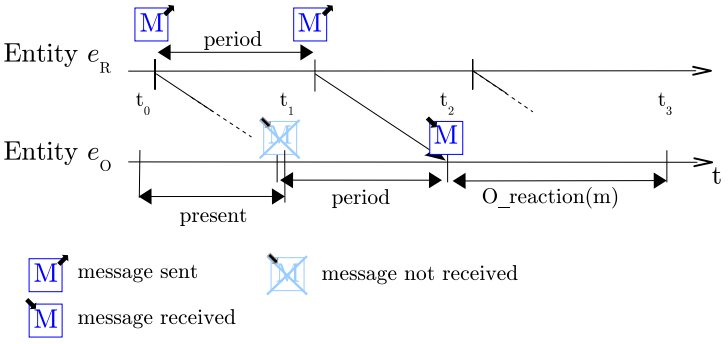


Figure 5.7: Time line for reaction to a message reception.

time t_0, it will become present at time $t_1 = t_0 + present$. In the worst case, the entity e_O will have to wait for the time *period* before it receives a message from e_R. Only then (at time $t_2 = t_1 + period$) it will have a consistent view of e_R. Entity e_O should still be outside of the consistency zone at t_2, so that it has time to react to the message before entering the safety zone. As illustrated on Figure 5.7, this requires:

$$CC(m_R) \geq (present + period) \cdot v_{max}(m_R) + CZ(m_R), \tag{5.4}$$

where *present* is the time required for an entity to become present, once in the coverage, and *period* is the period at which messages are sent.

Constraint F2: Communication degradation Constraint F2 states that, when the communication is degraded, the entity being notified must ensure that its mode is compatible with the ones of entities that might enter its safety zone. In the case of direct communication, it is the responsible entity who will receive feedback. This implies that a responsible entity must have time to revert to a fail-safe mode if one of its messages is not delivered in the critical coverage.

If an adaptation occurs before delivery of a message, this message might not be delivered to the entire critical coverage. It will take up to $adaptNotif$ for entity e_R to be notified of the adaptation. In the case where the coverage is not big enough, the responsible entity must have time to switch to another mode m'_R whose critical coverage $CC(m'_R)$ is covered before the incoming entity enters $CC(m'_R)$. So to cater for the worst case which would be an adaptation occurring just at the message delivery time (t_2 as defined above, see Figure 5.8), the following is required:

$$CC(m_R) \geq \big(present + period + adaptNotif + R\_reaction(m_R)\big) \cdot v_{max}(m_R) + CC(m'_R)\,, \tag{5.5}$$

where $adaptNotif$ is the maximum bound on the time for the producer to be notified of an adaptation of the critical coverage, and m'_R is the mode, among all modes to which the responsible entity might switch from m_R when $CC(m_R)$ is not covered, whose critical coverage is the largest (to ensure that it can switch to either of them).

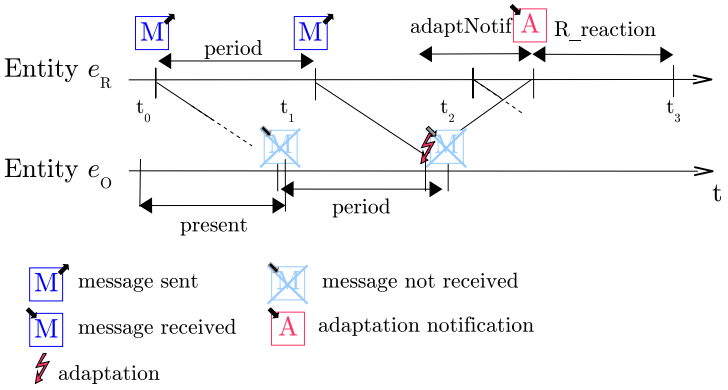


Figure 5.8: Time line for reaction to an adaptation.

Constraint C1: Entering a non-fail-safe mode When a responsible entity enters a non fail-safe mode, it needs to warn other entities at least a predefined time Δ in advance. This delay must ensure that all incoming entities will have been informed of the planned mode switch (this takes $msgLatency$), and after that, that entities still have t_{warning} to react. This implies:

$$\Delta \geq msgLatency + t_{\text{warning}} \,. \tag{5.6}$$

The responsible entity must also ensure that after its message has been delivered, it will have time to be notified of the proximity on which it was delivered (this duration is bounded by $adaptNotif$), to cancel its mode switch if the delivery zone is not big enough. This requires:

$$\Delta \geq msgLatency + adaptNotif \,. \tag{5.7}$$

So, the value of Δ can be derived:

$$\Delta = msgLatency + \max(t_{\text{warning}}, adaptNotif). \tag{5.8}$$

Constraint E2: Reacting to a transfer To ensure that an entity will have time to react to a warning within t_{warning}, it needs to ensure that it can, within t_{warning}, either switch to a mode that is compatible with the modes of all elements in the safety zone of the entity that has sent the message (using both a priori information and information potentially included in the transfer), or avoid entering the safety zone, or leave it if it was inside it. So there must be an upper bound $O\_reaction(m_{\text{R}})$ to the time required to perform either of these actions, and we need $t_{\text{warning}} \geq O\_reaction(m_{\text{R}})$.

Requirements summary From Equations 5.4 and 5.5, we can deduce:

$$CC(m_{\text{R}}) \geq (present + period) \cdot v_{\max}(m_{\text{R}}) + \max\Big(CZ(m_{\text{R}}), \\ \big(adaptNotif + R\_reaction(m_{\text{R}})\big) \cdot v_{\max}(m_{\text{R}}) + CC(m'_{\text{R}})\Big) \,. \tag{5.9}$$

Alternatively, using the formula for the size of the consistency zone CZ from Equation 5.3, this can be expressed as:

$$CC(m_{\mathrm{R}}) \geq (present + period) \cdot v_{\max}(m_{\mathrm{R}}) + \max\Big(SZ + t_{\mathrm{warning}} \cdot v_{\max}(m_{\mathrm{R}}), \\ \big(adaptNotif + R\_reaction(m_{\mathrm{R}})\big) \cdot v_{\max}(m_{\mathrm{R}}) + CC(m'_{\mathrm{R}})\Big). \quad (5.10)$$

The minimum requirements on entities obeying a contract with transfer via direct communication can therefore be summarised as follows:
The responsible entity must:

- send messages at intensity i_{warning} within a critical coverage of size

$$CC(m_{\mathrm{R}}) = (present + period) \cdot v_{\max}(m_{\mathrm{R}}) + \max\Big(SZ + t_{\mathrm{warning}} \cdot v_{\max}(m_{\mathrm{R}}), \\ \big(adaptNotif + R\_reaction(m_{\mathrm{R}})\big) \cdot v_{\max}(m_{\mathrm{R}}) + CC(m'_{\mathrm{R}})\Big) \quad (5.11)$$

  when it intends to enter or is in a non fail-safe mode m_{R};

- warn entities at least

$$\Delta = msgLatency + \max(t_{\mathrm{warning}}, adaptNotif) \quad (5.12)$$

  in advance before entering a non fail-safe mode;

- enter a fail-safe mode within $R\_reaction(m_{\mathrm{R}})$ after having been notified of an adaptation.

Other entities must:

- upon reception of a warning message at intensity i_{warning}, enter a mode that is compatible with the modes of all elements in the safety zone of the entity that has sent the message, or ensure not to enter its safety zone or leave it, within $O\_reaction(m_{\mathrm{R}})$, with $O\_reaction(m_{\mathrm{R}}) \leq t_{\mathrm{warning}}$.

5.2.3.2 Transfer via indirect communication

Constraint F1: Being in a non-fail-safe mode To ensure constraint C1 when it is in mode m_{R}, a responsible entity must continuously signal at an intensity $i_{\mathrm{signalling}}\ (m_{\mathrm{R}})$, such that over the zone $CZ(m_{\mathrm{R}}) = SZ + t_{\mathrm{warning}} \cdot v_{\max}(m_{\mathrm{R}})$, the intensity of the signal is at least i_{warning} (where $v_{\max}(m_{\mathrm{R}})$ is the maximum relative speed at which the responsible entity can approach other entities when it is in mode m_{R}).

Constraint E2 : Reacting to a transfer To ensure that it will have time to react to a warning before an incompatibility can happen, a sensing entity must ensure that its sensing coverage is big enough. Consider the case where an entity e_{R} enters the sensing coverage of an entity e_{O} at time t_0. In the worst case, sensing will not be initiated until after e_{R} has spent a period within the sensing

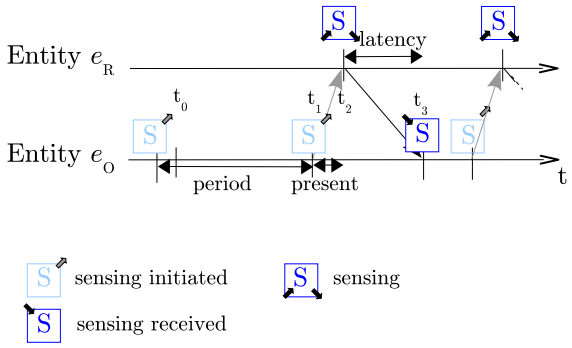


Figure 5.9: Time line for reaction to a message reception in the worst case.

range i.e., at $t_1 = t_0 + period$. Then, it will take *present* before the entity e_R is actually sensed at $t_2 = t_1 + present$. Eventually, e_O will receive the sensed data at $t_3 = t_2 + latency$. As illustrated on Figure 5.9 , this requires that the sensing entity e_O in a mode m_O can sense a signal of intensity i_{warning} in a zone of size $CC(m_O)$ that fulfils the condition:

$$CC(m_O) \geq (present + period + latency) \cdot v_{\max}(m_O), \tag{5.13}$$

where *present* is the time required for an entity to become present, once in the coverage. As it knows that it will be warned at least t_{warning} in advance, entity e_1 must ensure that within this warning time, it will have time to react. Therefore e_O must be able to change its mode so that it is compatible, or leave or not enter the safety zone within $t_{warning}$.

Constraint F2: Communication degradation In the case of indirect communication, sensing entities are notified of the degradation of communication conditions. This implies that when the sensing coverage required for a mode is not covered, they need to have time to revert to a mode whose sensing coverage is covered before any responsible entity might enter that coverage. It will take up to $adaptNotif$ for entity e_O to be notified of the adaptation. In the case where the sensing range is not big enough, e_O must have time to switch to another mode m'_O whose sensing range $CC(m'_O)$ is covered before the incoming entity enters it. So to cater for the worst case which would be an adaptation occurring just at the sensing time (t_2 as defined above, see Figure 5.10), the following is required:

$$CC(m_O) \geq \big(present + period + adaptNotif + O\_reaction(m_O)\big) \cdot v_{\max}(m_O) + CC(m'_O)\,, \tag{5.14}$$

where $adaptNotif$ is the maximum bound on the time for e_O to be notified of an adaptation of the sensing coverage, m'_O is the mode, among all modes to which the sensing entity might switch to from m_O when $CC(m_O)$ is not covered, whose critical coverage is the largest; and $O\_reaction(m_O)$ the time required by entity e_O to switch from m_O to m'_O.

Constraint C1: Entering a non-fail-safe mode When a responsible entity enters a non fail-safe mode, it needs to warn other entities at least a predefined Δ in advance. This delay must ensure that

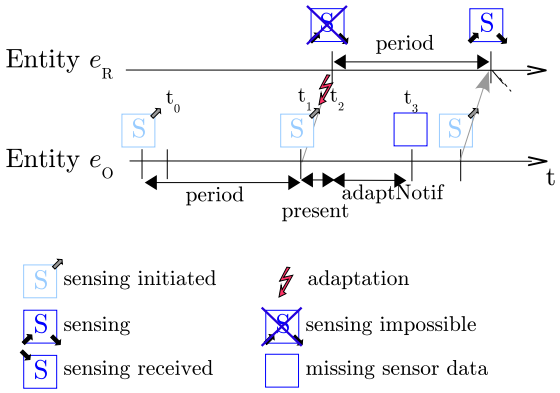


Figure 5.10: Time line for reaction to an adaptation in the worse case.

all incoming entities have been informed of the planned mode switch (this takes *latency* for the signal to be propagated, and then *period* for the entity to sense it), and after that, that entities still have t_{warning} to react. This implies:

$$\Delta = latency + period + t_{\text{warning}} \ . \tag{5.15}$$

An entity sensing the signal can suffer an adaptation of its sensing range In the worst case, the adaptation can happen just at the sensing time. So the sensing entity needs to have time to be notified of the adaptation and react to it within t_{warning}.

Requirements summary So, from Equations 5.13 and 5.14, we can deduce a lower bound on the critical sensing coverage :

$$CC(m_O) \geq (present + period) \cdot v_{\max}(m_O) + \max\Big(latency \cdot v_{\max}(m_O), \\ \big(adaptNotif + O\_reaction(m_O)\big) \cdot v_{\max}(m_O) + CC(m'_O)\Big) \ . \tag{5.16}$$

The minimum requirements on entities obeying a contract with transfer using indirect communication are therefore as follows:
The responsible entity must:

- continuously signal at an intensity $i_{\text{signalling}}$ (m_R), such that over the zone

$$CZ(m_R) = SZ + t_{\text{warning}} \ \cdot v_{\max}(m_R)\,, \tag{5.17}$$

  the intensity of the signal is at least i_{warning} when it intends to enter or is in a non-fail-safe mode.

- warn entities at least

$$\Delta = latency + period + t_{\text{warning}}\,, \tag{5.18}$$

  in advance before entering a non fail-safe mode.

Other entities must:

- at any time be in a mode m_{O} whose sensing coverage

$$CC(m_{\mathrm{O}}) = (present + period) \cdot v_{\max}(m_{\mathrm{O}}) + \max\Big(latency \cdot v_{\max}(m_{\mathrm{O}}), \\ \big(adaptNotif + R\_reaction(m_{\mathrm{O}})\big) \cdot v_{\max}(m_{\mathrm{O}}) + CC(m'_{\mathrm{O}})\Big), \quad (5.19)$$

  is covered,

- ensure that at any time, they can switch to a mode that is compatible with those of all elements in the safety zone of the responsible entity or avoid entering, or leave, the safety zone within $t_{warning}$.

5.2.3.3 Transfer via either direct or indirect communication

A responsible entity in a contract with transfer without feedback can use both direct and indirect communication. At any time, it can use either or both means, to ensure that other entities will be warned of the possible incompatibility. As other entities do not know in general whether the responsible entity uses indirect communication, they must at any time be able to sense a signal and react to it. For other entities to know the sensing coverage (c.f. Equation 5.21), they need to know the maximum speed $v_{\max}(m_{\mathrm{O}})$ at which a responsible entity can approach them by default. The responsible entity, however, might approach faster when the communication coverage is sufficient to ensure that other entities are warned.

Therefore sensors can be used to supplement direct communication: when communication is not sufficient, the responsible entity can revert to a mode for which sensing is sufficient. The requirements for this contract can thus be deduced from the requirements for the contracts without transfer via direct and indirect communication derived above: a responsible entity must ensure that either the (direct communication) coverage covers the critical coverage of the mode it is in, or that it signals over the consistency zone of this mode; and other entities must be ensure that they are in a mode whose sensing coverage is covered, and be able to react, at any time, to a warning received by either direct or indirect communication.

5.2.4 Contracts with feedback

In a contract with feedback, entities that receive a transfer can choose to transfer the responsibility back to the responsible entity by sending feedback. Therefore, the safety constraint will not be violated unless the responsible entity is not in a fail-safe mode, and a transfer has not been received, or an entity has not reacted appropriately to a transfer that it has received. As illustrated on the fault tree depicted in Figure 5.11, this requires that:

C1 Entering a non-fail-safe mode: when a responsible entity intends to enter a non fail-safe mode, it must warn entities at intensity i_{warning} at least t_{warning} in advance, because an incompatibility can

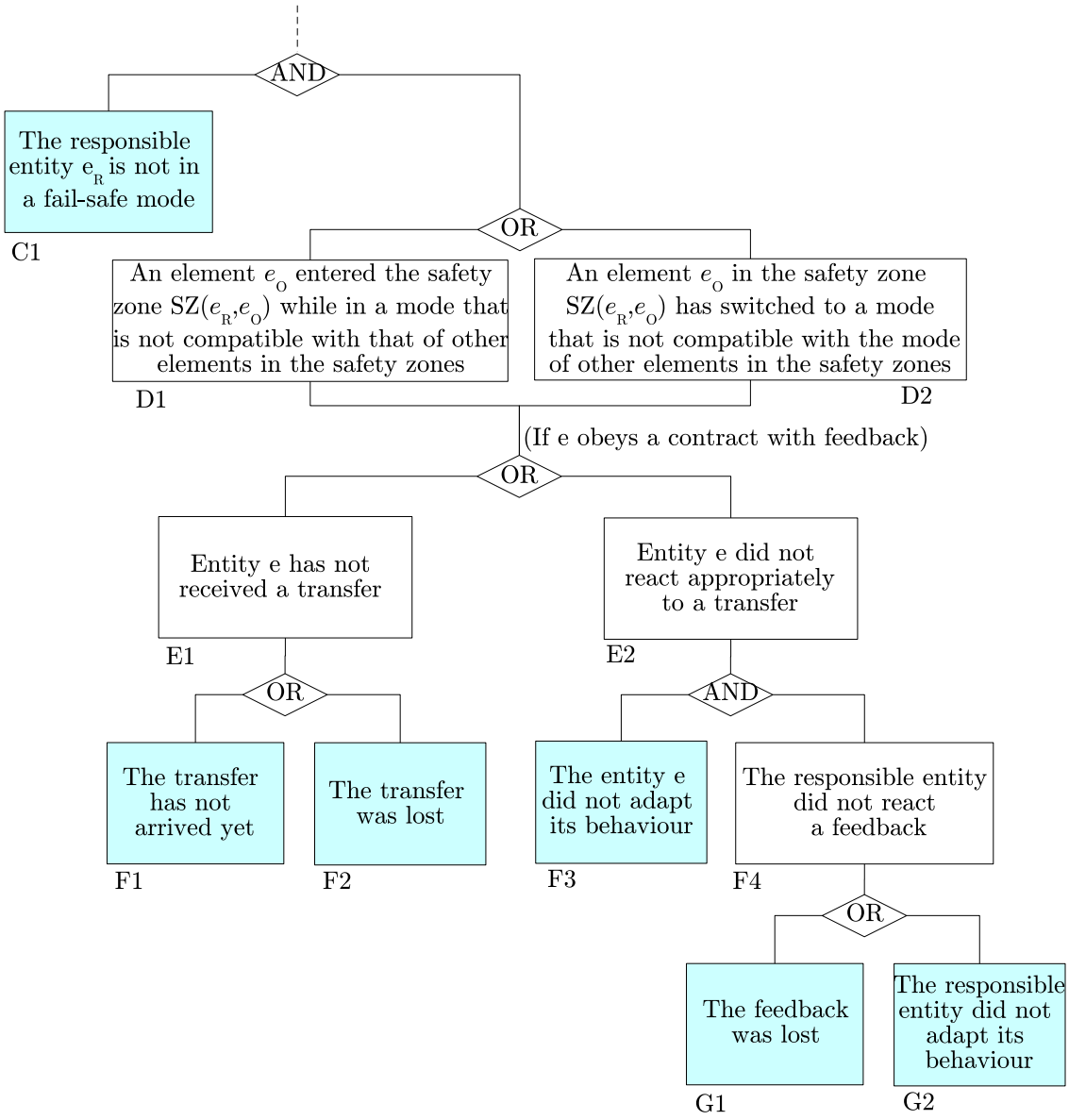


Figure 5.11: Fault-tree for a violation of the safety constraint for a contract with feedback.

Constraint	Name	Entity affected
C1	Entering a non-fail-safe mode	Responsible entity
F1	Being in a non-fail-safe mode	Responsible entity
F2	Communication degradation	Entities receiving feedback about the means of communication used for transfer
F3	Reacting to a transfer	Other entity
G1	Communication degradation	Entities receiving feedback about the means of communication used for feedback
G2	Reacting to a feedback	Responsible entity

Table 5.5: Constraints for a contract with feedback.

happen when it enters that mode. This implies warning entities that will enter the consistency zone, but also entities already within the safety zone.

F1 Being in a non-fail-safe mode: when the responsible entity is not in a fail-safe mode, it must warn other entities t_{warning} before they enter the safety zone,

F2 Communication degradation: when the communication means used for transfer degrades, at least one of the entities in the contract will be notified (the entity sending message in the case of direct communication, the entity sensing in case of indirect communication). An entity being notified that communication is not sufficient anymore must ensure that its mode is compatible with those of entities that might enter its safety zones.

F3 Reacting to a transfer: when an entity receives a warning from a responsible entity about a possible incompatibility, it should adapt its behaviour within t_{warning} so that the incompatibility cannot happen, or send feedback within $t_{\text{feedback}} - latency$.

G1 Communication degradation: same as F2, but for feedback instead of transfer.

G2 Reacting to feedback: when a responsible entity receives feedback, it needs to ensure that the safety constraint will not be violated by changing mode within $t_{\text{warning}} - t_{\text{feedback}}$.

The constraints are summarised in Table 5.5. Note that constraints C1, F2 and F3 are the same as for contracts with transfer. Communication degradation constraints (F2 and G1) can impose constraints on both responsible and other entities because of the two potential exchange of messages (for transfer and feedback). Therefore, the requirements for entities in a contract TxFy (contract with transfer via communication means 'x' and feedback via communication means 'y', see Chapter 4) are the same as for a contract Tx (contract with transfer via communication means 'x', no feedback), except that:

- Other entities have the choice upon reception of a transfer to either switch to a mode that is compatible with those of all elements within the safety zones of the entity sending the message or avoid entering the safety zone, or leave the safety zone if they were inside it, within t_{warning}, or send a message within t_{feedback}(constraint F3).

- The communication must be sufficient for the feedback (constraint G1). This implies that if the feedback is via direct communication, other entities must ensure that their messages are delivered on a zone of size

$$CC_F(m_{\mathrm{O}}) = (present + period) \cdot v_{\max}(m_{\mathrm{O}}) + \max\Big(SZ + t_{\mathrm{warning}} \cdot v_{\max}(m_{\mathrm{O}}), \\ \big(adaptNotif + O\_reaction(m_{\mathrm{O}})\big) \cdot v_{\max}(m_{\mathrm{O}}) + CC(m'_{\mathrm{O}})\Big)\ , \quad (5.20)$$

  which corresponds to Equation 5.11 with $m_{\mathrm{R}} = m_{\mathrm{O}}$ and $t_{\mathrm{warning}} = t_{\mathrm{feedback}}$. If the critical coverage for the feedback is not covered, other entities must enter a fail-safe mode within $O\_reaction(m_{\mathrm{O}})$.
  If the feedback is via indirect communication, responsible entities must ensure that they will be able to sense on a coverage of:

$$CC_F(m_{\mathrm{O}}) = (present + period) \cdot v_{\max}(m_{\mathrm{O}}) + \max\Big(latency \cdot v_{\max}(m_{\mathrm{O}}), \\ \big(adaptNotif + R\_reaction(m_{\mathrm{O}})\big) \cdot v_{\max}(m_{\mathrm{O}}) + CC(m'_{\mathrm{O}})\Big)\ . \quad (5.21)$$

  If the critical coverage for the feedback is not covered, the responsible entity must enter a fail-safe mode.

- The responsible entity must be able to react to feedback, i.e., must at any time when it has sent messages, be able to, upon reception of feedback, ensure that no incompatibility will happen, by, within $t_{\mathrm{warning}} - t_{\mathrm{feedback}}$, either switching to a mode that is compatible with the modes of all elements within its safety zone and all entities that have sent feedback, or change its trajectory so that no entity that has given feedback will enter its safety zone (constraint G2).

5.3 Combining different scenarios

The previous sections of this chapter have shown how requirements on the behaviour of entities of a scenario can be derived from a single safety constraint. In this section, we first discuss how the requirements on entity behaviour can be derived for a combination of safety constraints, and we then show how scenarios can be combined.

5.3.1 Deriving requirements for a combination of safety constraints

The safety constraints of a scenario can be expressed as a condition on the states of the elements in this scenario. Different constraints can be linked by disjunction, as described in 4.2.3. The set of requirements on the behaviour of entities imposed by a combination of safety constraints is the union of the requirements on the behaviour of entities imposed by each safety constraint. In other words, the set of conditions that must be fulfilled for switching to (respectively remaining in) a mode to ensure

two safety constraints is the union of the sets of conditions for switching to (respectively remaining in) this mode for each of the safety constraints.

Because requirements on the behaviour of entities might be contradictory, the requirements imposed by the combination of two solutions might not be implementable. In such cases, we say that the solutions are not compatible. For example, the requirements on a car might be to stop when approaching a red traffic light and get out of the way of emergency vehicles, which are incompatible if the emergency vehicle is approaching a red traffic light. Other solutions, however, might be compatible. For example, the solution of the emergency vehicle safety constraint where entities use a contract with feedback might be compatible with the solution of the traffic light safety constraint.

Assessing whether two solutions are compatible is a challenging problem because whether requirements are contradictory depends on the details of the entity behaviour and cannot easily be evaluated. For example, it is not trivial to systematically deduce that stopping and getting out of the way are not compatible, unless the semantics of each of the modes are defined and in particular the speed invariant (the speed is null in the stopped mode, and needs to be non null for getting out of the way). This work does not investigate how to predict whether two solutions are compatible. Note that, however, if the requirements are on different unrelated actuators, or if the modes for which the requirements are contradictory cannot be reached (for example because an emergency vehicle will never transfer its behaviour next to an intersection), then the solutions are compatible.

5.3.2 Deriving requirements for a combination of scenarios

Different scenarios not only have different safety constraints, but also have different elements, or potentially the same elements described using a different set of modes, because, as explained in Section 4.2, modes should be chosen with respect to a given safety constraint. The requirements on the behaviour of the elements that appear in only one of the scenarios remain the same for a combined scenario as for the scenario in which it appears. For entities defined in two scenarios, if their modes differ in both solutions, their mode diagrams need to be combined. The modes and the transitions of the combined diagram are the cross product of the modes and transitions, respectively, of the two mode diagrams. More formally, if we denote mode diagram by (M, δ) where M is the mode set and $\delta : \mathrm{M} \times \mathrm{M} \mapsto \{0; 1\}$ the transition function, then the cross diagram of the mode diagrams (M_1, δ_1) and (M_2, δ_2) is the mode diagram $(\mathrm{M}_\mathrm{C}, \delta_\mathrm{C})$ where

$$\mathrm{M}_\mathrm{C} = \left\{ m_1 \times m_2, (m_1, m_2) \in \mathrm{M}_1 \times \mathrm{M}_2 \right\}$$

and

$$\forall (m_a \times m_b, m_c \times m_d) \in \mathrm{M}_\mathrm{C} \times \mathrm{M}_\mathrm{C}, \;\; \delta_\mathrm{C}(m_a \times m_b, m_c \times m_d) = \delta_1(m_a, m_c) \vee \delta_2(m_b, m_d)\,.$$

Consider for example, the combination of two scenarios: “pedestrian traffic light” and “overtaking”. Cars are involved in both scenarios, and their modes will typically have been defined dif-

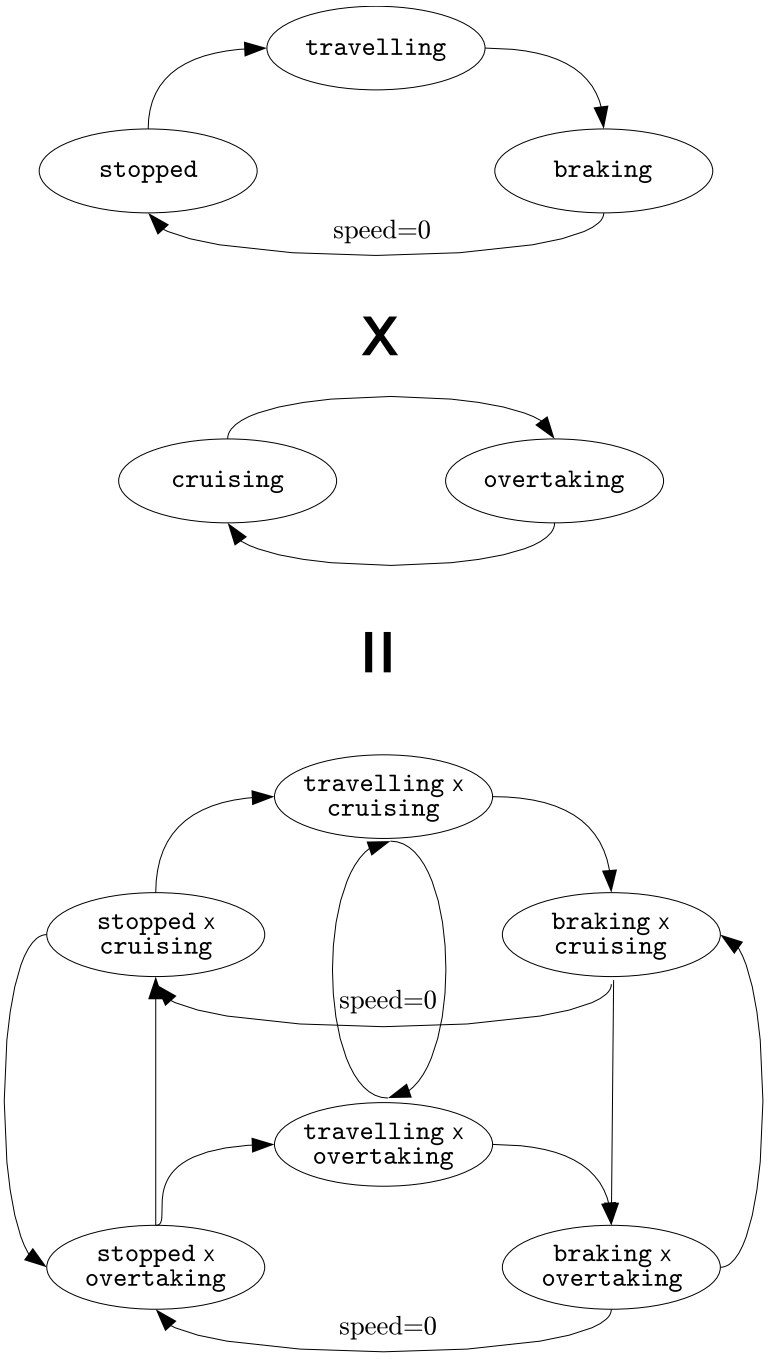


Figure 5.12: Example of mode diagram combination.

ferently in the two scenarios: `travelling`, `stopping`, and `stopped` for the traffic light scenario, and `cruising` and `overtaking` for the overtaking scenario. The modes of cars in the combined scenario are: `travelling-and-cruising`, `travelling-and-overtaking`, `stopping-and-cruising`, `stopping-and-overtaking`, `stopped-and-cruising` and `stopped-and-overtaking`. This cross-product process is illustrated in Figure 5.12.

Practically, however, semantic enables a number of the cross modes to be discarded. For example, if it is known to model users that cars do not overtake when approaching a junction, the mode `stopped-and-overtaking,` which would express that while a car was overtaking it has stopped to obey a traffic light, can be discarded because it will never be reached.

The state variables of an element defined in a combined scenario is the union of the set of state variables in each scenario. So, for example, if the state variables of entities of type car are its speed, direction and position in the traffic light scenario; and its speed, position, and its indicator status in the overtaking scenario, then the state variables for the combined scenario are its speed, direction, position and indicator status.

The set of invariants of a mode product of two modes is the conjunction of the set of invariants of each mode. So, for example, if the mode `travelling` has an invariant that the speed is non-null, and if the mode `overtaking` has an invariant expressing that the indicator is on, the mode `travelling-and-overtaking` has two invariants: that the speed is non null and that the indicator is on. Incompatible invariants enable the detection of unreachable product modes. For example, if the mode `stopped` has an invariant that the speed is null and the mode `overtaking` has an invariant that the speed is non-null, then it can be automatically detected that the mode `stopped-and-overtaking` can be discarded because it will never be reached.

The safety constraint of the combined scenario is the conjunction of the safety constraints (or the disjunction of the incompatibilities, since those are conditions that should not occur). The requirements on entities of a combined scenario can be derived as explained in the previous section.

The priority list of the combined scenario must be defined by merging and sorting the priority lists of both scenarios.

5.4 Summary

In this chapter, we have shown how to design a solution using Comhordú, and how the set of solutions for a given scenario can be derived. In addition, we have discussed how scenarios can be combined. In the next chapter, we describe a tool that we have designed to support the use of the model and automate the systematic steps of the development process, such as assessing mode compatibility, deriving the solution set, and deriving the requirements on entities behaviours.

Chapter 6

Design and Implementation of ComhorMod, a Tool Supporting Comhordú

This chapter describes the design and implementation of ComhorMod, a modelling tool to support the use of the Comhordú model. This tool is intended to assist the developers of autonomous mobile entities by guiding them through the development process outlined in Chapter 5. ComhorMod encompasses a graphical user interface that allows developers to specify the application and its safety constraints graphically. From this specification, ComhorMod automatically generates requirements on the behaviour of entities to ensure the application's safety constraints. Additionally, ComhorMod generates skeletons of the implementation of entities that can be completed using existing tools provided by the MoCoA framework (Senart et al. 2006).

This chapter first describes the MoCoA framework and the associated tool chain. It then presents the architecture of ComhorMod and its development steps, as well as the key algorithms used. Finally, this chapter describes how ComhorMod can be used in conjunction with the MoCoA tool chain to generate applications, and highlights possible future work for the extension of the tool.

6.1 The sentient object tool chain

The work described in this thesis forms part of the Aithne project, the goal of which is the development of a middleware framework for sentient computing, called MoCoA. This section introduces existing work from this project, first focusing on the abstractions defined, and then describing existing tools from this framework that are of particular interest for this work. The overview presented in Section 6.1.1 is largely adapted from the description presented in (Senart et al. 2006), of which the author is a co-author.

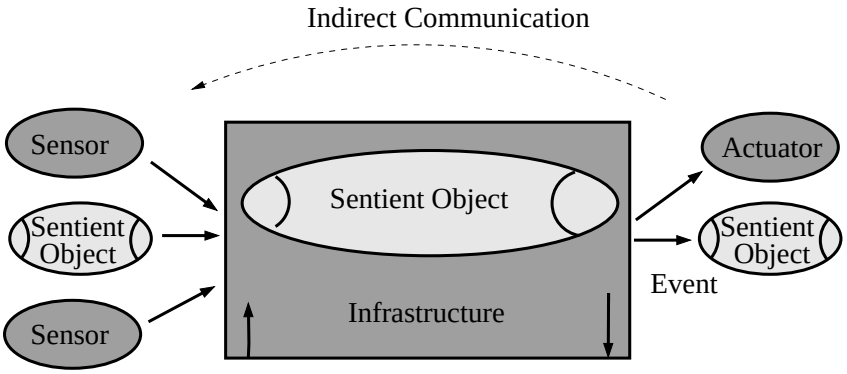


Figure 6.1: A sentient object.

6.1.1 MoCoA

MoCoA (Senart et al. 2006) is a customisable Middleware fOr Context-aware mObile Applications. It supports a small set of programming abstractions that are suitable for building a wide range of context-aware applications for deployment in a fixed or (ad hoc) mobile environment. For each of these abstractions, MoCoA provides a set of implementations via a library of components, hence allowing easy development and reuse. The abstractions are reviewed in the following sections.

6.1.1.1 Sentient objects, sensors and actuators

In the MoCoA framework, applications are structured using the sentient object abstraction (Verissimo et al. 2002, Biegel & Cahill 2004). *Sentient objects* are mobile intelligent entities, that extract, interpret and use context information obtained from sensors, other sentient objects, and their computational infrastructure, to drive their behaviour. The granularity of a sentient object is not constrained: a robot, a component of a robot, or a traffic light controller are all potential examples of sentient objects.

In this model, a *sensor* is defined as an entity that produces software events in reaction to a real-world stimulus and is an abstraction of some physical device. To act upon their environment, sentient objects use *actuators*. An actuator is any component that consumes software events and reacts by attempting to change the state of the real-world via some physical device. To facilitate mobility as well as loose coupling between dynamically varying collections of anonymous mobile devices, a sentient object is both a producer and a consumer of asynchronous events. As explained above and illustrated in Figure 6.1, a sentient object can also receive events from its infrastructure. Notice also that sentient objects may communicate with each other both directly by means of events or indirectly via the environment.

The behaviour of all sentient objects follows a specific pattern. First, a sentient object may receive input from a variety of sources (sensors, sentient objects and infrastructure) that needs to be integrated before being used in determining the overall context of the sentient object. As an example, some of the robots used in the Aithne project are equipped with ultrasonic, orientation, and location sensors. The outputs of these sensors are fused together with input from nearby traffic lights in order to determine

the robots' context with respect to obstacles, traffic lights, and their destinations.

Context recognition determines the current context based on fused data and past history. For example, that someone has left their bed is inferred from historical information that someone was in the bed and new sensor input indicating that they are no longer present.

Sentient objects are then expected to change their behaviour or act upon their environment based on some rules. This implies some form of intelligent reasoning captured in an inference component. Rules may be predefined or learnt over time.

6.1.1.2 Events, event channels and proximities

In MoCoA, sentient objects communicate using events. Proximities and event channels are used to tackle challenges arising in large-scale and geographically dispersed applications in which large volumes of events are raised.

MoCoA adopts the approach of the space-elastic model (presented in Chapter 3) for communication between sentient objects: events are sent to an area rather than to specific consumers. Therefore, MoCoA supports proximity filtering in addition to typical subject- and content-based filtering.

Event channels are defined to specify constraints on propagation and delivery of asynchronous events within a proximity. An event channel is specific to an event type and allows a producer to define the real-time guarantees that have to be maintained within a given geographical area.

To support event-based communication with event channels and proximities, the MoCoA framework uses different instantiations of the STEAM event service (Meier & Cahill 2003): STEAM for use in wireless ad hoc networks, and RT-STEAM for soft and hard real-time in ad hoc environments. In STEAM, proximities can be circular or hull-shaped. The set of filters available depends on the type of event parameters. For example, if the parameter is a location, the following filters can be used: `distanceIncreases`, `distanceDecreases`, `withinRange`, and `beyondRange`. Parameter types include location, time, integer, double and string.

6.1.1.3 Readings and facts

Sensor fusion is used to manage the uncertainty of data captured from the real world and to derive higher-level context information. In MoCoA, the fusion process relies on a set of pipelines, with each pipeline being composed of a combination of generic and sharable components (c.f. Figure 6.2). Input events in a pipeline are processed through different stages.

First, a pipeline extracts *readings* (i.e., raw values produced by a sensor, a sentient object, or the local infrastructure) from events. Then, a sequence of transformations is applied to the readings by the components present in the pipeline. The final result is a piece of higher-level information, in the form of one or more *observed facts*.

A pipeline may be composed of different components: handler, smoothing, buffer, and fusion. MoCoA provides a set of implementations of these components, e.g., fusion can perform a sum, an

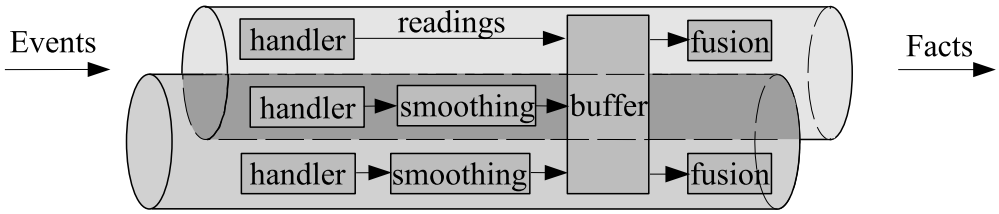


Figure 6.2: Example of two pipelines.

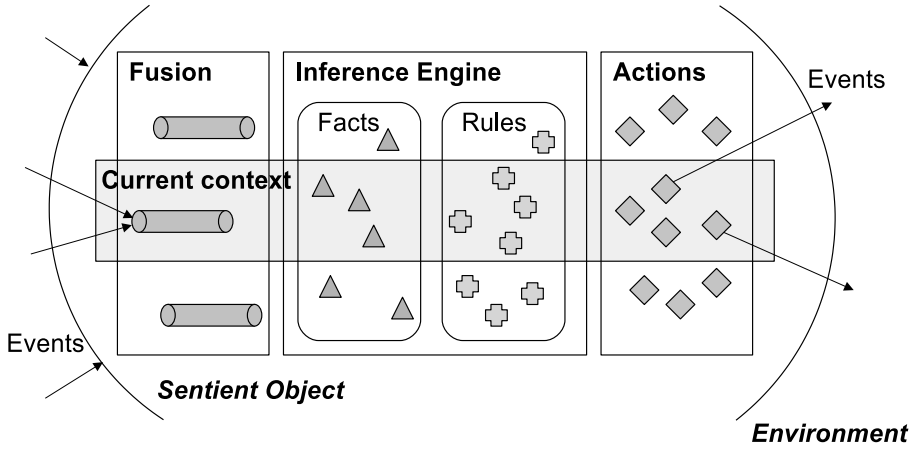


Figure 6.3: The MoCoA architecture.

average function or might rely on a Bayesian network.

6.1.1.4 Contexts

Context is defined as any information currently available in the environment that can be used to characterise the situation of an entity (Schmidt et al. 1999), such as its current location, the presence of other sentient objects in its vicinity or the state of its underlying infrastructure. In MoCoA, the contexts in which a sentient object can be during its lifetime are organised as a context graph, where only a subset of contexts can be transitioned to from the current context. Not only are contexts useful to structure complex applications, contexts are also the basic abstraction for Context-Based Reasoning (Gonzalez & Ahlers 1998) within a sentient object allowing the set of event channels, pipelines, facts, rules and actions that are relevant at any time to be filtered. Only one pipeline is active in a given context, thereby constraining the set of event channels being used. Because the number of pipelines is restricted, only some facts may be asserted in this context. Subsequently, as depicted in Figure 6.3, only a subset of rules needs to be evaluated and the permitted actions are limited.

6.1.1.5 Rules and actions

The knowledge of a sentient object is structured into facts. To ultimately determine the appropriate behaviour of a sentient object in response to its environment, an inference engine is used to infer knowledge (i.e., to assert *derived facts*) from previously asserted facts and to select the actions to be

taken. The different types of *action* that an object is able to perform are as follows: fact assertion, fact retraction, event production, and code execution. The action selection decision within a sentient object is based on *rules*.

In MoCoA, a first order logic inference engine can be used to reason about a set of facts and predefined rules in order to derive the intelligent behaviour of sentient objects. The rules take the form of `condition/actions`. Conditions are expressed in terms of asserted facts and can include operators, e.g., && , ||, ∃ and ∀.

6.1.2 MoCoA tools

This section describes the tools from the MoCoA framework that supports the development of sentient objects.

6.1.2.1 SOGen

SOGen provides a Java API that allows sentient objects to be designed using the abstractions described above, e.g., contexts, rules, events produced and consumed. From this design, SOGen automatically generates an implementation of the sentient object . The components used are chosen and customised depending on application requirements (e.g., timeliness requirements). An architecture description language based on XML is used to describe the services provided and required by a component, as well as the dependencies between components. As each component of the library exhibits its dependencies with a descriptor, MoCoA can build a customised middleware that fits the application requirements using existing components from the library.

The code generation currently provides C++ code, but as it is decoupled from the object design process, different languages may be easily incorporated without affecting the programming paradigm. The generated code can be targeted either for deployment or to be run within a simulator.

6.1.2.2 SOMod

To further assist the construction of sentient objects, MoCoA has been extended with a graphical interface. This allows sentient objects to be defined graphically using the abstractions defined above. Once a sentient object has been designed, its implementation can be automatically generated. See, for example, Figure 6.4, that shows the contexts of an entity of type car being defined with SOMod. Additionally, an XML descriptor of a sentient object may be generated so that sentient objects' specifications may be saved and edited or reused at a later date.

6.1.2.3 Sentient simulator

The sentient simulator is a discrete event simulator that can be used to test the behaviour of sentient objects. Sensors and actuators are simulated, and in particular movement is emulated. The communication between sentient objects is also emulated, in accordance to the space-elastic model: sentient

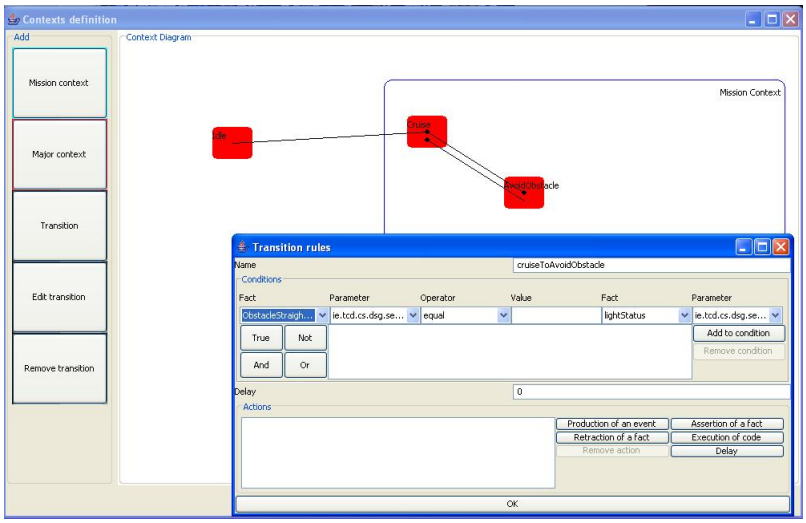


Figure 6.4: SOMod screenshot.

objects are able to communicate in real-time within a geographical proximity that varies over time. The simulator implements this model by delivering messages to sentient objects present, at the time of delivery, in the actual coverage of an event channel, and varying the actual coverage over time. The simulator and each sentient object execute as a separate process, communicating via UNIX sockets. At every time step, the simulator sends all events whose delivery time it is to the sentient objects that are in the coverage for these events, and then receives the events produced by the sentient objects and adds then to the event queue until their delivery time.

6.1.2.4 Sentient viewer

The sentient viewer is a graphical tool that allows the visualisation of trace files generated by the sentient simulator. The viewer allows to see the trajectory of sentient objects over time, their state, as well as the communication between them. Figure 6.5 shows two screenshots of a stretch of road on which 24 car sentient objects as well as one emergency vehicle sentient object are travelling. The emergency vehicle sends periodic messages to cars in its vicinity to warn them of its arrival, and is therefore recognisable by the actual coverage depicted around it. The actual coverage is depicted as a coloured area while the desired coverage is a circle. The left figure shows a time where the actual coverage is equal to the desired coverage, and the picture on the right shows a time where the actual coverage is lower than the desired coverage. To ease visualisation, the colour of both desired and actual coverage changes from green to red when the desired coverage is not covered. Notice that some cars have moved to the left lane as they received a message that the emergency vehicle was arriving.

6.1.2.5 Summary

The work described in this thesis builds on the existing tools provided by the MoCoA middleware. These tools, whose interaction is pictured in Figure 6.6 , allow the development of intelligent entities

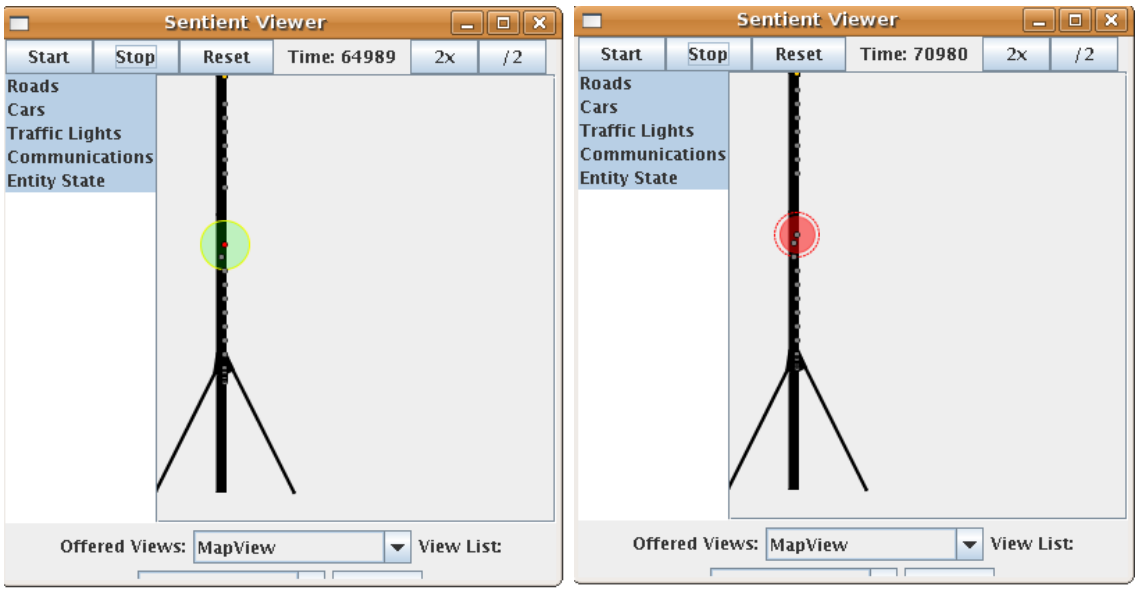


Figure 6.5: Sentient viewer screenshots.

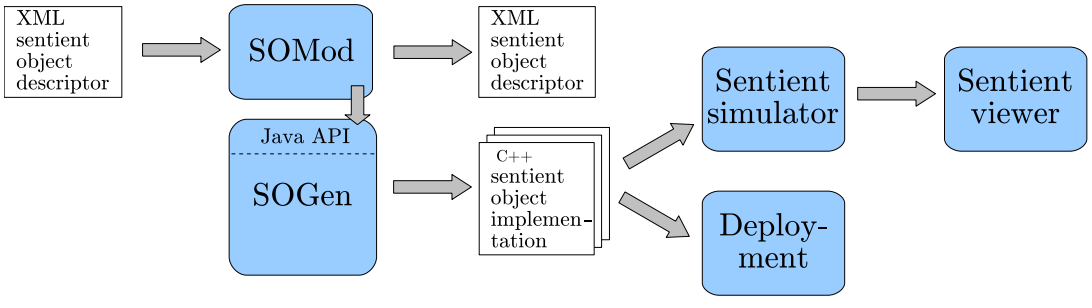


Figure 6.6: MoCoA tool chain.

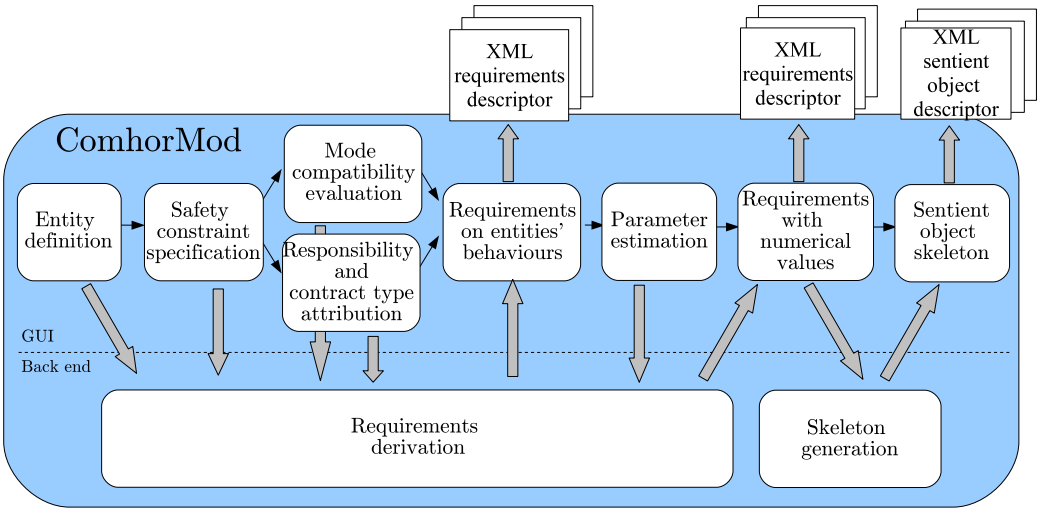


Figure 6.7: ComhorMod architecture.

called sentient objects, that use sensors and direct communication to get information about their environment, reason on this information and send commands to actuators and messages to other sentient objects. Existing tools in the MoCoA tool chain support the development of sentient objects. These tools, however, are limited to the development of a single sentient object and do not support the decisions as to how sentient objects should interact and how their behaviour can ensure safety constraints that span several entities. This is the goal of the work described in this thesis, and the ComhorMod tool.

6.2 ComhorMod

ComhorMod (for Comhordú Modelling) is a Java tool supporting the development of applications composed of autonomous mobile entities. ComhorMod encompass a graphical user interface (GUI), that guides entity developers through the process of defining an application using Comhordú. In addition, a back-end automates the systematic steps of the process, i.e., mode compatibility derivation, solutions generation and requirements derivation, and uses the results derived in Chapter 5 to automatically translate application specifications into requirements on entity behaviour.

The ComhorMod GUI is organised in seven steps corresponding to the development process. These steps and their interaction with the back-end are shown in Figure 6.7, and detailed below. While the steps must be completed in sequence, users can go back to previous steps at any time to vary the choices that they have already made.

6.2.1 Entity definition

The first step consists of defining each of the entity types in the application. To define an entity type, its state variables must first be defined, by specifying a name and a type. ComhorMod supports state

Operator	Syntax	Comment
$>, <, \geq, \leq, =, \neq$	<state-variable> <operator> <value>	<value> should be of the same type as the state variable
constant	<state-variable> **constant**	-

Table 6.1: Mode invariant syntax.

State variable type	Operators
Double, Integer	$>, <, \geq, \leq, =, \neq$, **constant**
Boolean	$=, \neq$, **constant**
Position	$=, \neq$, **constant**
Enum	$=, \neq$, **constant**

Table 6.2: Mode invariants operators.

variables of type Integer, Double, Boolean and Position (pair of Double). In addition, users can define enumerated types whose values are represented as strings (e.g., type Colour, whose values are "green", "amber" and "red"). Secondly, the modes of the entity type must be defined, by specifying a name, and optionally invariants on the state variables of this entity. As defined in Chapter 4, invariants are predicates on state variables and their possible variations that remain true while an element is in a given mode, and they are used to capture the semantics of the mode, thereby allowing the tool to reason on them. For example, an invariant that the speed of an entity is null, i.e., speedVar = 0 can be associated to the mode `stopped`. The supported mode invariant syntax and operators are defined in Tables 6.1 and 6.2 respectively. Finally, developers are required to define possible transitions between the modes. These transitions are displayed as a mode diagram, that provides an efficient means for developers to visualise the behaviour of the entity that they are currently defining. A screenshot of some of the main screens for the step 1 is shown in Figure 6.8. This figure shows that an entity of type car has already been defined, and an entity of type emergency vehicle is currently being defined. The emergency vehicle has three state variables: mode, speed, and position, and three modes defined: `stopped`, `acceleratingToV1`, and `goingAtV1`.

6.2.2 Safety constraint specification

The second step of ComhorMod consists of defining the safety constraint of the application. The safety constraint is defined as a combination of conditions on state variables that should never be fulfilled, i.e., incompatibilities. The definition of basic incompatibilities of types SOV, distance and cardinal, as defined in Chapter 4 is supported. (Note that entities of type SOS are not supported in this version, but could be very similarly to the others.) An incompatibility of each of these types can be defined on any state variable of any entity (except distance, that only applies to variables of type position), and operators applying to the type of the state variable can be selected (the operators supported are the same as for invariants, see Table 6.2). In addition, incompatibilities can be combined using $\vee$ (or)

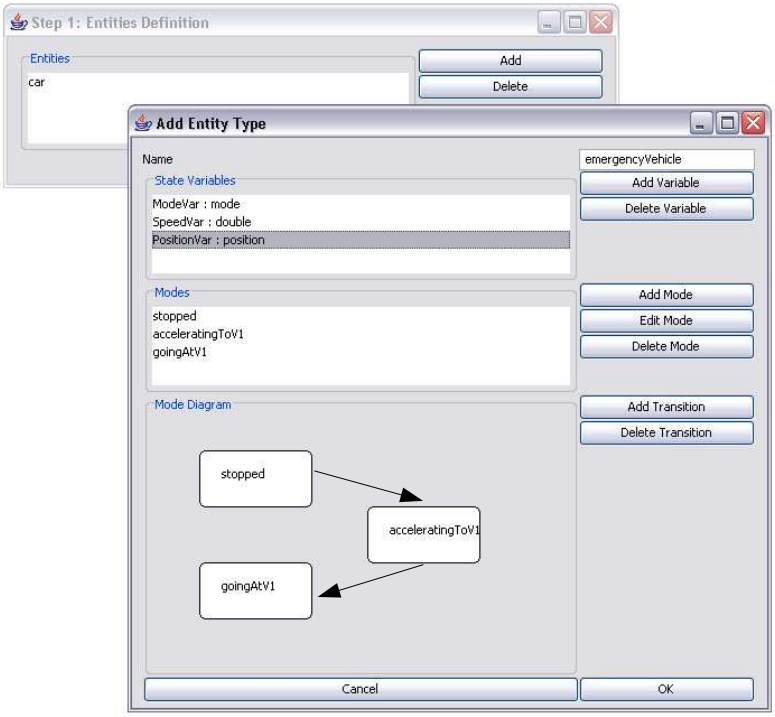


Figure 6.8: Screenshot of step 1: entity definition.

and $\wedge$ (and) operators. All incompatibilities defined must be combined into a single incompatibility before the step 2 can be completed. Figure 6.9 shows the main screens of this step and the different types of incompatibility that can be defined.

6.2.3 Mode compatibility

In step 3, the entity developer is presented with a mode compatibility matrix, showing the compatibility between the modes of different types of entities for each entity type pair (the value 'true' in a square of the matrix means that the corresponding modes of each entity types are compatible). Mode

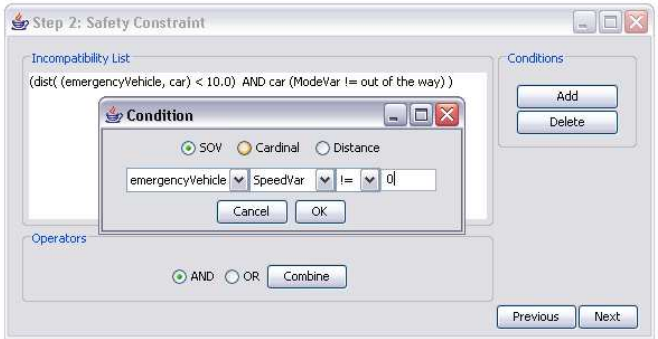


Figure 6.9: Screenshot of step 2: safety constraint specification.

Algorithm 1 Mode compatibility evaluation.

```
assessCompatibility(mode1, mode2, incompatibility)
  if incompatibility is of basic type (SOV, cardinal or distance)
    return the result of the appropriate assessCompatibility method

  else          //the incompatibility is composite,
    if the incompatibility is of type AND,
       return the disjunction of the compatibility of its operands
    if the incompatibility is of type OR,
       return the conjunction of the compatibility of its operands
```

Algorithm 2 Mode compatibility evaluation for an incompatibility of type SOV.

```
assessCompatibilitySOV(mode1, mode2, incompatibility)
  if the incompatibility SOV refers to entities for which mode1 or mode2 is a mode
    for each invariant of this mode
      if it applies on the same state variable than the incompatibility
        if it allows to assert that the incompatibility will not happen
         return true
  return false
```

compatibility is derived using the mode invariants defined in step 1, and the safety constraint defined in step 2.

To assess mode compatibility, the safety constraint is decomposed into basic incompatibilities, as shown in Algorithm 1. Then, each incompatibility is treated depending on its type. Algorithm 2 shows the example of the assessment of the compatibility of two modes with respect to an incompatibility of type SOV: invariants of the modes are checked to see if they allow to assert that the incompatibility will not happen. For example, if the incompatibility is speed > 30 and the invariant is speed ≤ 10, then the modes are compatible. The compatibility of two modes is true if and only if their invariants are sufficient to ensure that the safety constraint will not be violated.

Figure 6.10 shows a screenshot of step 3, where the compatibility of the modes of entities of types emergency vehicle and car is displayed. Note that both the mode `stopped` of the emergency vehicle and the mode `outOfTheWay` of the car are compatible with all the modes of the other entity type, and are therefore fail-safe modes.

Developers can use this step to check that they have specified the applications with enough detail to ensure that the semantics of the different modes in terms of safety have been captured, and therefore that entities will be able to make progress while ensuring the safety constraint. In particular, this requires that at least one of the entity types have at least one fail-safe mode. This is also checked automatically in the following step.

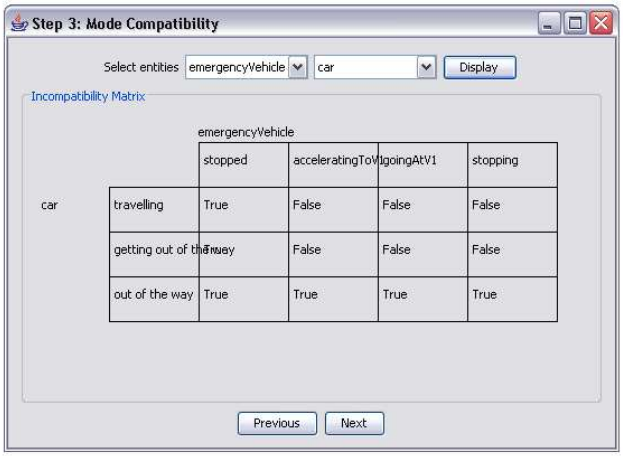


Figure 6.10: Screenshot of step 3: mode compatibility.

6.2.4 Responsibility attribution and contract choice

In step 4, the tool displays the set of solutions for the scenario. This requires the generation of all the possible combinations for this scenario. Algorithm 3 describes how these combinations are generated, by recursively setting the value for the contract types of the n-first entities (the variable combinationBeginning contains the values that have been fixed), and exploring the possible values for the other entities. Then, the set of all combinations is assessed using the criteria on fail-safe modes presented in Section 5.1.1, to derive the set of solutions, using the mode compatibility matrix derived in the previous step. Algorithm 4 shows this process: if for example, the combination contains a contract without feedback, then the algorithm checks that the (initially) responsible entity has a fail-safe mode, otherwise the combination is discarded.

Users are required to choose a combination amongst the solutions, hence choosing the entity type that is to be responsible, as well as the contract types to use. In the example shown in Figure 6.11, all three contract types are available, as both emergency vehicles and cars have fail-safe modes. If no responsible entity and contract type combination is available, the scenario as currently defined is not solvable. In this case, users should go back to previous steps to change some of the data entered, or add more data (in particular additional mode invariants allowing the tool to identify that some modes are compatible).

6.2.5 Requirements on entities behaviour

In step 5, the requirements on entity behaviours are derived, presented to the developer and output in files to allow developers to save and reuse them. These requirements are derived according to the responsible entity and contract type chosen, following the results of Chapter 5. Algorithm 5 shows how the requirements are derived for a contract with transfer via direct communication, for example.

Algorithm 3 Combination generation.

```
generateAllCombinationForAScenario
  for all elements in the scenario
    if it this is not a passive element // therefore this element can be responsible
      generateCombinations (this element, null, all elements in the scenario)

generateCombinations (respElement, combinationBeginning, elementsToTreat)
  if elementsToTreat is empty, add combinationBeginning to the list of solutions
  else
    elementToTreat = elementsToTreat.getElement()
    if elementToTreat is a passive element   // can only have a contract with transfer
      generateCombinations(respElement, combinationBeginning::
        (ContractWithoutTransfer(elementToTreat)), elementsToTreat)
    else
      generateCombinations(respElement, combinationBeginning::
        (ContractWithoutTransfer(elementToTreat)), elementsToTreat)

      for each transfer communication means
        generateCombination(respEntity, combinationBegining::
          ContractWithTransfer(elementToTreat,transferMeans),elementsToTreat);

      for each transfer communication means
        for each feedback communication means
          generateCombinations(respElement, combinationBeginning::
           (ContractWithFeedback(elementToTreat, transferMeans, feedbackMeans), elementsToTreat)
```

Algorithm 4 Check whether a combination is a solution.

```
Boolean isASolution (Combination comb)
  for all contracts of the combination
    if it is a contract without transfer
      if the responsible entity type  does not have a fail-safe mode
        return false

   if it is a contract with transfer without feedback
     if the transfer is via direct communication
       if the responsible entity type does not have a fail-safe mode
          return false
        if the other entity type  does not have a fail-safe mode
          return false

    if it is a contract with feedback
     if the responsible entity type does not have a fail-safe mode
        return false

      if both transfer and feedback are via direct communication (contract TdFd)
         OR the transfer is via indirect communication (contract TiFx)
         OR the transfer is via both direct and indirect communication (contract TdiFx)
           if the other entity type does not have a fail-safe mode
              return false

    return true // all the contracts in the combination have been examined, all are feasible
```

Figure 6.11: Screenshot of step 4: responsibility attribution and contract type choice.

Algorithm 5 Deriving the requirements for a contract with transfer via direct communication (Td).

```
deriveRequirementsContractWithTransferViaDirectCommunication(entity)
if the entity is responsible
  for each mode m of its modes
    if m is not a fail-safe mode
      add the condition that it can remain in that mode only if it delivers a message in
      a zone of size greater or equal to -CC-m-  every -period-
      add the condition that to transition to m, an entity must have sent a message at
      least -delta - m- ago, on a zone of size greater or equal to CC-m to all the
      transitions to m
else   // the entity is not responsible
 for each mode m of its modes
  if m is not a fail-safe mode
    add the condition that an entity can remain in this mode only as long as it does not
    receive a message informing it of a transfer of responsibility
```

Note how the requirements are expressed using parameters names for the different application-specific values such as period, the critical coverage CC of a mode, and the delay delta before switching to a mode.

Requirements are then expressed in an XML-based language that allows users to check them and also to save and reuse them. Figure 6.12 shows a portion of such a requirement, that states that before an entity of type emergency vehicle can enter the `acceleratingToV1` mode, it needs to have sent a message less than a time called *period* ago, and this message needs to have been delivered on a coverage whose size is a parameter called $CC - acceleratingToV1$.

The generated requirement specifications contain all the conditions to ensure that the requirements expressed in Chapter 5 are met. They do not, however, contain the scenario specifications such as entity states and modes.

6.2.6 Parameter estimation

In step 6, the numerical values of the application-specific parameters are requested. The values requested depend on the values needed for the requirements, and organised by modes to which they relate. For example, as the requirements on entities of type emergency vehicle require the calculation of the critical coverage for the mode `acceleratingToV1`, the parameters that are required to calculate this critical coverage, i.e., for a contract with feedback, *present*, *period*, $v_{\max}$(`acceleratingToV1`),

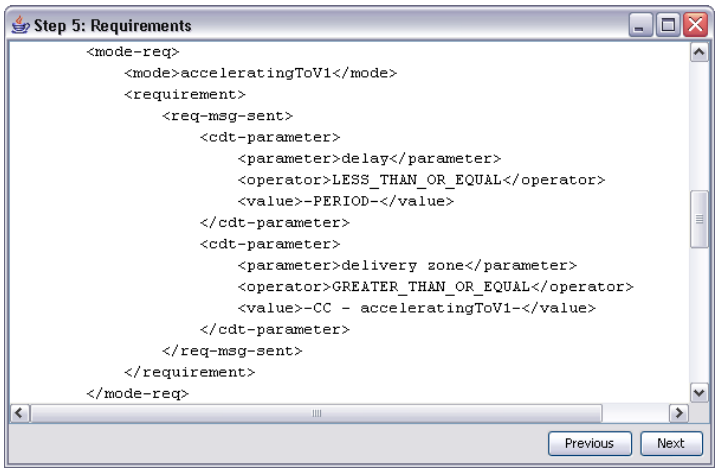


Figure 6.12: Screenshot of step 5: requirements.

SZ, $t_{warning}$, $adaptNotif$, and $R\_reaction$(`acceleratingToV1`) (cf Equation 5.11 on page 94) are requested, as shown on Figure 6.13.

When a user fills in some of the values, the values of the variables that depends on them are automatically calculated and updated. This allows the user to see how the value of the different parameters impact the requirements and choose the values so that the requirements will be implementable. For example, in the example shown Figure 6.13, the values of the critical coverage for each of the modes are automatically updated when the values of other variables are changed. This allows developers to tune the value of period, for example, to trade-off between the quantity of messages sent and the size of the coverage on which this messages must be delivered, as the size of the critical coverage is proportional to the message period.

6.2.7 Requirements with numerical values

In step 7, the requirements are updated with the numerical values provided and calculated in step 6. Figure 6.14 shows that the values of *period* and $CC - acceleratingToV1$ have been replaced by their numerical values.

6.2.8 Sentient object skeleton

Step 8 outputs and displays a sentient object skeleton in XML language that can be used as input for SOMod. To ensure compatibility with SOMod, the XML skeleton conforms to a sentient object XML schema (which is not included here due to its length). Figure 6.15 shows an excerpt of the skeleton generated for an entity of type emergency vehicle. This part shows the definition of a behaviour rule that stipulates that entities of type emergency vehicle can only travel at v_1 when messages are sent every period and are deliverable in the coverage CC_1.

The sentient object skeleton is generated by translating the mode graph into a context graph, where

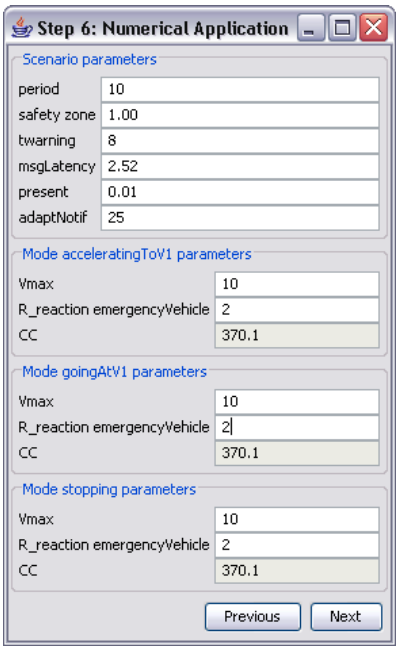


Figure 6.13: Screenshot of step 6: numerical application.

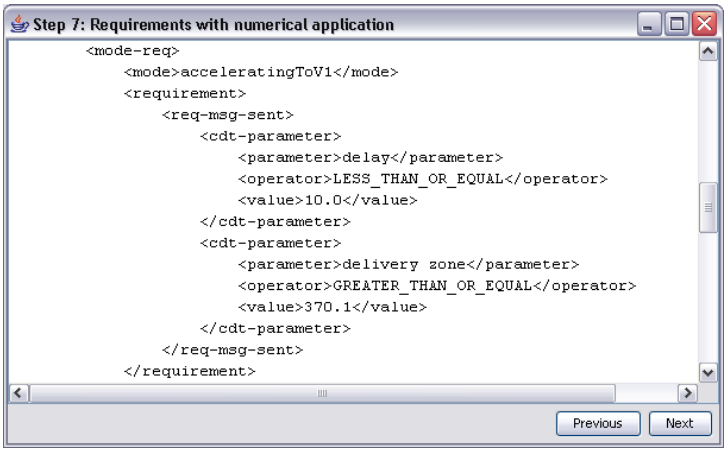


Figure 6.14: Screenshot of step 7: requirements with numerical values.

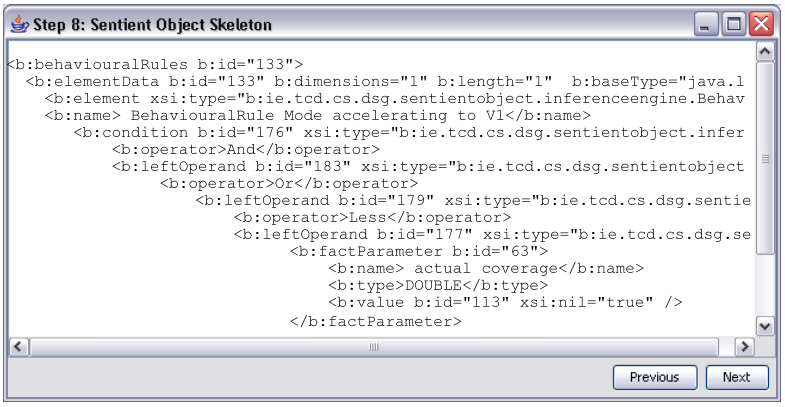


Figure 6.15: Screenshot of step 8: sentient object skeleton.

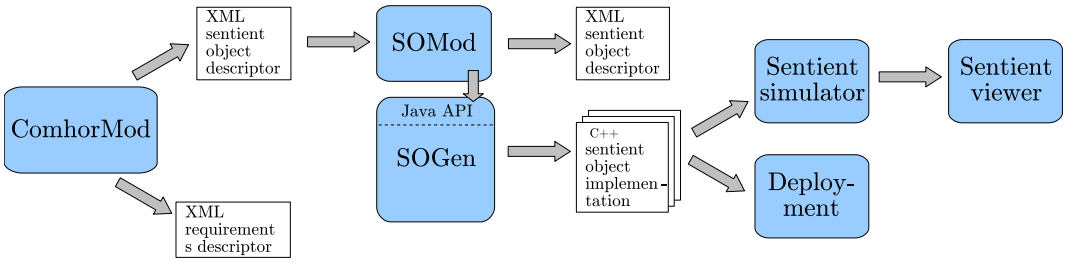


Figure 6.16: ComhorMod's integration in the MoCoA tool chain.

one major context represents each mode. The transitions between modes are replaced by transitions between the two equivalent contexts. In addition, a mission context that encompasses all the major contexts is created. This context captures the rules that must always apply for an entity, for example, message sending if the entity is responsible in a contract with transfer. Conditions on being in a mode and transitioning to a mode are translated as conditions in behaviour and transition rules. Finally, the values provided by users for the application-specific parameters are inserted as facts in the sentient object fact base. The generated skeletons encompass all the information provided in terms of entity behaviour (state and modes), as well as behaviour to ensure that all the requirements derived in the previous steps will be met. So for example, if a requirement states that messages must be delivered in a zone $CC(m)$ for an entity to be in a mode m, then a rule is added to the corresponding context that stipulates that if a message has not been delivered to $CC(m)$, then the entity must transition to one of its fail-safe modes.

Skeletons, however, specify entity behaviours only in terms of safety, and therefore skeletons need to be completed in SOMod. The skeleton of an entity of type car whose behaviour is specified in terms of the modes `travelling`, `getting_out_of_the_way` and `out_of_the_way`, for example, will encompass conditions so that a car in the mode `travelling` transitions to `getting_out_of_the_way` when it receives a message that an emergency vehicle is approaching. The skeleton does not, however, specify the behaviour of the car when it is in the mode `travelling` for example. This behaviour, which might be specified as going to a series of way points for example, needs to be specified using SOMod. In addition, the specific sensors and actuators used must be specified (they are referred to using XML descriptors).

6.3 Achievements and future work

ComhorMod extends the MoCoA tool chain to supports reasoning on the interactions between sentient objects and system-wide safety constraints. Sentient object skeletons generated by ComhorMod can be loaded into SOMod. Figure 6.16 shows the integration of ComhorMod within the MoCoA tool chain. ComhorMod currently eases significantly the development of autonomous mobile entities that fulfil system-wide safety constraints, by automating the requirements derivation, and generating a sentient object skeleton.

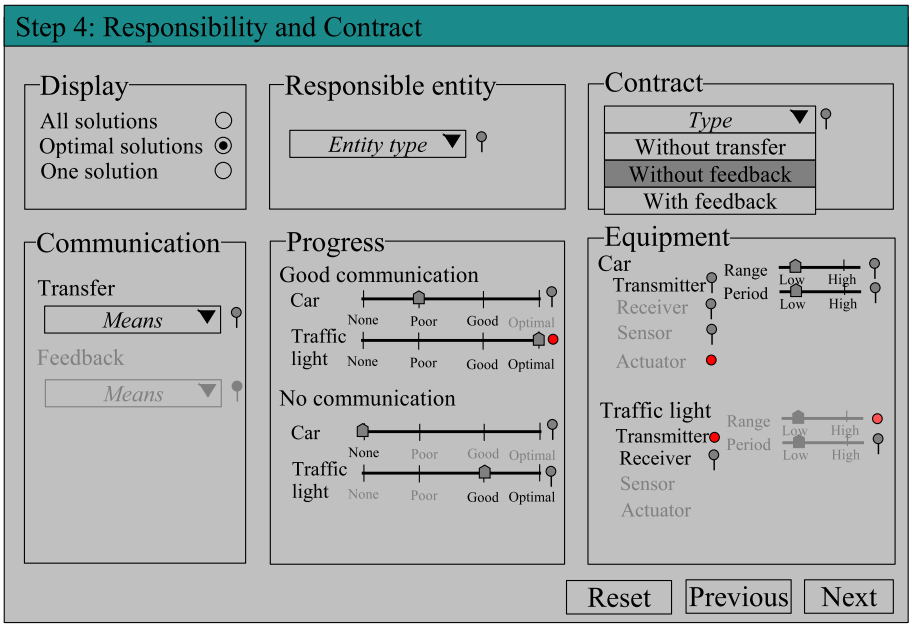


Figure 6.17: Alternative design for step 4.

An area that could be improved, however, is decision support for responsible entity selection and contract choice. As explained in Chapter 5, the responsible entity and contract type combination simultaneously influences the solvability of the scenario, the requirements on entities behaviours and the efficiency of the solution in terms of progress of entities of different types both when the communication is degraded and when it is not. A screen that would present all the parameters and their possible values, and would allow to set some of their values and see how other would vary would help developers when making this decision. The design for such a screen is shown in Figure 6.17. A developer could specify, for example, that the entities of type car are equipped with a transmitter and an actuator, but have no wireless receiver. Then only the solutions that do not require that the car receives messages via direct communication are displayed. Similarly, developers can fix the responsible entity type for example, and only the contracts corresponding to solutions with this entity type will be displayed.

Another possible extension of ComhorMod is the addition of the notions of goals and priorities; this would enable the tool to rank the possible combinations using the criteria and results presented in Section 5.1.2, hence potentially automating the combination choice. These extensions have not been implemented due to time constraints, but the results supporting them have been derived in Chapter 5.

Finally, while the design of ComhorMod enables it to support both direct and indirect communication between entities, the current implementation does not support contracts using indirect communication as no implementation of the sensor model is currently available.

6.4 Summary

This chapter first presented existing tools in the MoCoA framework that support the implementation of autonomous mobile entities. In the second section, ComhorMod, a tool that supports the use of the Comhordú model to develop autonomous mobile entities that adapt their behaviour depending on currently available information was presented. ComhorMod allows the translation of system-wide safety constraint into requirements on the behaviour of individual entities. In addition, ComhorMod generates for each entity type a sentient object skeleton in the form of an XML sentient object descriptor that can be used as input to the MoCoA tool chain, to generate entity implementations. Finally, this chapter outlined the achievements of ComhorMod and presented possible extensions.

Chapter 7

Evaluation and Results

This thesis presents Comhordú, a new coordination model supporting the development of autonomous mobile entities, and ComhorMod, a tool supporting the development of entities using Comhordú. This chapter first presents an outlook of the evaluation by presenting the aspects of the work that are evaluated as well as the evaluation strategy. The experimental configuration is then presented. Comhordú has been applied to two scenarios from the ITS domain, which are explained in turn. Finally, the results of the evaluation are discussed, and this chapter concludes with a summary.

7.1 Evaluation outlook

To guarantee that autonomous mobile entities ensure system-wide safety constraints while having access only to limited information, the approach taken in Comhordú is that entities adapt their behaviour depending on the information available. This chapter first demonstrates that this approach and the Comhordú model are useable (claim 1), by applying them to several scenarios from the ITS domain. In addition, this chapter shows how ComhorMod can be used to support the development of the entities of these scenarios (claim 2), and presents the advantages of using the tool. The solutions generated by the tool are also evaluated, in particular to show that they ensure that safety constraints are never violated (claim 3). Finally, this chapter demonstrates that Comhordú can be used to solve scenarios that are not solvable with existing coordination models (claim 4).

To validate these claims, two scenarios are presented: a pedestrian traffic light application and an emergency-vehicle warning system. These scenarios encompass entities of different types, both mobile and fixed, and exhibit strong timeliness and safety requirements. Both scenarios have been specified in terms of Comhordú's abstractions and their solutions derived using ComhorMod. The possible solutions for each scenario are discussed to demonstrate the outcomes and the flexibility of the tool. In addition, to validate the correctness of the solutions presented, three solutions using the three different contract types have been implemented using the tool chain and have been extensively tested through simulations. Finally, the last section of this chapter discusses the feasibility of implementing

these scenarios using existing coordination models.

7.2 Experimental configuration

This section describes the experimental configuration used for the evaluation. It first reviews how SOMod and the MoCoA tool chain were used for the evaluation, and then how communication was modelled.

7.2.1 Using SOMod and the MoCoA tool chain

The scenarios used for the evaluation have been modelled using ComhorMod. As described in Chapter 6, ComhorMod generates sentient object skeletons for each entity type of a scenario. These skeletons can then be used as input for SOMod, where the behaviour of entities can be completed with non safety-related features, i.e., actions that are motivated by the satisfaction of the goal rather than the safety constraints.

As the sentient object skeleton loading feature of the sentient object modelling tool SOMod is not fully implemented, its actions have been performed manually, i.e., the sentient object skeletons have been translated into calls to SOGen's API manually. Using SOGen, a C++ implementation has been generated from the Java specification of each sentient object. The implementations of the sentient objects of a scenario have been run in the Sentient Simulator to test the solutions generated. The traces generated by the simulator have been used to assess the correctness of the solutions as well as to illustrate their characteristics, as detailed in the next sections.

7.2.2 Direct communication modelling

As described in Chapter 6, the Sentient Simulator emulates the sensors and actuators of sentient objects, hence allowing sentient objects' behaviour to be tested. The simulator simulates real-time communication between sentient objects, with the range of communication varying randomly, as foreseen by the space-elastic model. Changes of the actual coverage were as follows: every $T_{coverage}$, there is a probability $p_{coverage}$ that the coverage is re-evaluated. The possible values of the coverage were divided into set of interest, and the distribution was defined amongst these sets. When the coverage is re-evaluated, there is a probability p_{DC} that it takes the value of the desired coverage, and a probability p_{value_i} that it takes each of the other set of values. For example, for the emergency vehicle scenario described below, three ranges of values are of interest for the size of the critical coverage: smaller than a first value CC_1 $[0; CC_1[$, greater or equal to CC_1 and less than the value of the desired coverage size DC $[CC_1; DC[$, and equal to the desired coverage size$[DC]$. CC_1 and DC are parameters that depend on the solution used. Except when otherwise specified, the experiments for this scenario have been conducted with the parameters described in Table 7.1. Figure 7.1 shows the variations of the actual coverage over time in one simulation. This coverage has not been derived

Parameter	Value
$T_{coverage}$	1s
$p_{coverage}$	50%
$p_{[0;CC_1[}$	20%
$p_{[CC_1;DC[}$	20%
$p_{[DC]}$	60%

Table 7.1: Communication parameters.

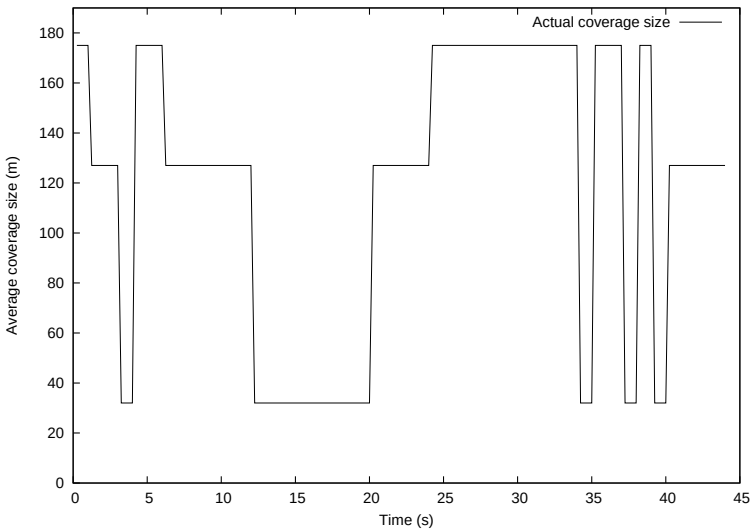


Figure 7.1: Actual coverage variations over one simulation.

from realistic data; however, it approximates the changes of coverage in a realistic setting and allows us to evaluate the safety characteristics of entity behaviours under varying condition.

7.2.3 Indirect communication modelling

The Sentient Simulator does not currently support indirect communication. To facilitate the evaluation of scenarios that rely on contracts without transfer, which require responsible entities to know about other entities to make progress, a presence sensor was implemented. This sensor allows entities to detect other entities within a proximity. Users can specify both the proximity and the latency of this sensor.

The next sections describe, for each scenario, how the scenario can be modelled using Comhordú, how a solution can be designed using ComhorMod and evaluates the solutions that have been automatically generated from these designs.

7.3 Pedestrian traffic light

Autonomous cars seem a promising approach to both reducing accidents, and alleviating traffic congestion by improving road usage. This scenario (Bouroche et al. 2006) considers pedestrian traffic lights for autonomous cars. The goal is for each traffic light to turn to red (when the colour of the light is mentioned, it is always the one intended for cars and not pedestrians) as soon as possible after the pedestrian presses its button. The safety constraint is that no car should pass through a traffic light while it is red, hence ensuring that no pedestrian is knocked down (provided that pedestrians are disciplined!). The goal is that cars can continue to travel as often as possible, and in particular when there are no pedestrians.

It is assumed that both cars and traffic lights are fitted with wireless communication facilities. It is not assumed, however, that cars are aware of the position of traffic lights. The protocols for safe driving, such as following the road and avoiding collisions with other cars, and navigation, such as routing to a destination, are outside the scope of this scenario.

This section first presents how the scenario can be modelled using ComhorMod, hence demonstrating that Comhordú is suitable for modelling applications, and eases their development. In a second part, a solution generated by ComhorMod and the MoCoA tool chain is evaluated.

7.3.1 Modelling the scenario in ComhorMod

This section demonstrates how the scenario can be modelled with the concepts of Comhordú by specifying the input that was entered for each step of ComhorMod.

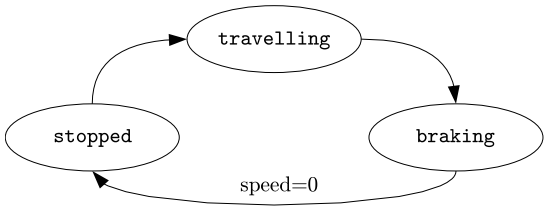


Figure 7.2: Mode diagram for the car entity type in the pedestrian traffic light scenario.

Mode	Invariant
`travelling`	speed $\leq v_{\max}$
`braking`	speed $\leq v_{\max}$
`stopped`	speed $= 0$, position constant

Table 7.2: Mode invariants for entities of type car in the pedestrian traffic light scenario.

7.3.1.1 Step 1: entities definition

The first step of ComhorMod requires developers to define and specify the entity types of a scenario. For this scenario, it is sufficient to consider the behaviour of the car as being either travelling, braking, or stopped. This corresponds to the modes of the car: `travelling`, `braking`, and `stopped`. The mode diagram of the car entity type is described in Figure 7.2. In addition to its mode, the state of the car encompasses its speed, geographical position, direction and destination. Assuming the speed of cars is bounded (for example by road speed limits), and the maximum bound is denoted $v_{\max}$, the mode invariants can be described as in Table 7.2. The goal of entities of type car can be expressed as:

$$\text{position} = \text{destination}\,,$$

where destination differs for each individual car.

The traffic light is considered to be either green or red. Switching from green to red cannot be instantaneous, as incoming cars need to be warned, therefore one mode is required to capture this behaviour. This mode can be seen as the equivalent of the amber light. Switching from red to green however does not require cars to be warned (in the worst case, cars will stop or remain stopped in front of a green traffic light which, while not being efficient, is safe). Therefore this can be instantaneous and does not require a special mode. To summarise, the modes of the traffic light are `green`, `switching_to_red`, and `red`. The state of the traffic light is defined as being its mode, position, colour, and the time of the next planned change if relevant. The mode diagram and mode invariants are described in Figure 7.3 and Table 7.3 respectively. The goal of entities of type traffic light is to be red when the button is pressed, or else to be green. This can can be formalised as:

$$\big((\text{buttonPressed} = \text{true}) \wedge (\text{colour} = \text{red})\big) \vee \big((\text{buttonPressed} = \text{false}) \wedge (\text{colour} = \text{green})\big)\,.$$

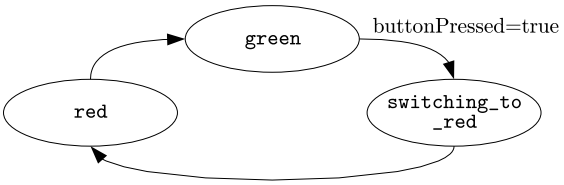


Figure 7.3: Mode diagram for the traffic light entity type in the pedestrian traffic light scenario.

Mode	Invariant
`green`	colour = green
`switching_to_red`	colour = green
`red`	colour = red

Table 7.3: Mode invariants for the traffic light entity type in the pedestrian traffic light scenario.

7.3.1.2 Step 2: safety constraints specification

The second step of ComhorMod consists of specifying the safety constraint. For this scenario, using the mode and state variable definitions, the safety constraints that no car should pass through a red light can be formalised by stating that the states of the traffic light and the car are compatible, except when they are close, the light is red, and the car is not stopped; i.e., if SZ is the size of the safety zone, which is, in this example, the pedestrian crossing:

$$s_{\text{car}}\mathcal{C}_{\text{s}}s_{\text{traffic light}} \text{iff } \neg\big(\text{distance}(s_{\text{car}}.\text{position}, s_{\text{traffic light}}.\text{position}) < SZ) \wedge (s_{\text{traffic light}}.\text{colour} = \text{red}) \wedge (s_{\text{car}}.\text{mode} \neq \texttt{stopped})\big)\,.$$

To capture that cars should be travelling except when there is a pedestrian, the priority list of the scenario can be formalised as:

$$\big(\text{if}(\text{buttonPressed} = \text{true})\ \texttt{red},\ \texttt{travelling}\big)\,.$$

7.3.1.3 Step 3: mode compatibility evaluation

Using the safety constraint and the mode invariants, the mode incompatibility matrix for entities of type car and traffic light is automatically derived by the tool and presented to the user (see Table 7.4 and corresponding screenshot on Figure 7.4). Users can see, for example, that the modes `stopped` of the entity type car and `green` of the entity type traffic light are both fail-safe modes.

7.3.1.4 Step 4: responsibility and contract type attribution

Deriving the solution set: Since both entity types of this scenario have a fail-safe mode, any contract can be used a priori, and entities of either type can be made initially responsible for the safety constraint. Therefore, the tool offers the choice between all the combinations, which, for this scenario, corresponds to the choice of either cars or traffic light entities to be responsible, as well as a contract type.

	Modes of entities of type car		
	`travelling`	`braking`	`stopped`
Modes of entities of type traffic light			
`green`	✓	✓	✓
`switching_to_red`	✓	✓	✓
`red`	✗	✗	✓

Table 7.4: Mode compatibility matrix for entities of type car and traffic light in the pedestrian traffic light scenario.

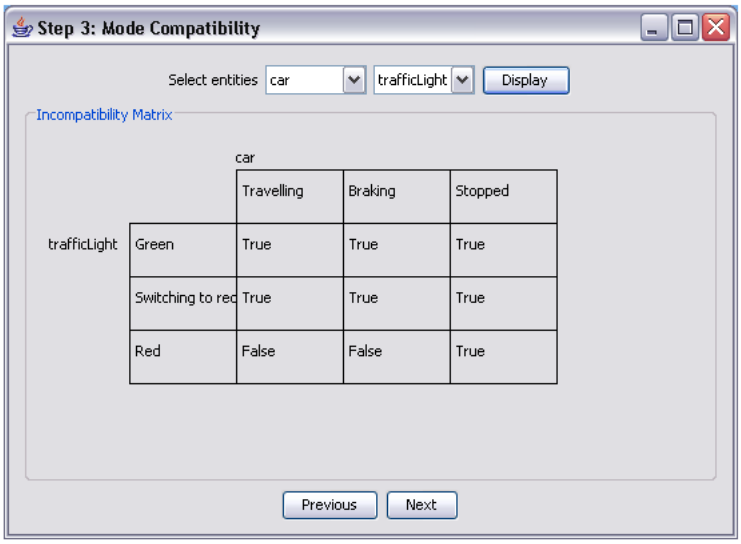


Figure 7.4: Screenshot of ComhorMod's step 3 for the traffic light scenario.

A possible contract type and responsible combination is, for example, that traffic lights be responsible and switch to red only when they know that there is no car around, hence using a contract without transfer. With this contract, cars can always travel, but pedestrians might have to wait for a long time before crossing. The possible combinations are discussed below.

Choosing a solution: Which entity type should be responsible, and which contract should be used depends on which entity type should be able to make progress when communication is ideal, and which entity type should be able to make progress when communication is degraded as explained in Chapter 5. From the priority list, it can be derived that it is more important for the traffic light to make progress in ideal communication conditions, but that it would be better if cars also made progress when the light is green. The contracts where traffic lights will be able to make good progress when the communication is optimal are: cars are responsible, using a contract without transfer, and traffic lights are responsible, using a contract with transfer without feedback via direct or indirect communication (Tx) (cf Table 5.3 on page 85).

The safety constraint, however, contains a reference to the state variable "colour" of traffic lights. Because this variable is discreet, a car cannot foresee its variations, i.e., it is not predictable, and therefore when cars are responsible, they cannot make progress, i.e., travel towards their destination, except when traffic lights inform them of the colour they intend to be in in the future (cf Section 5.1.1 on page 78). A contract without transfer where cars are responsible will therefore allow only very limited progress and so the solution is discarded. Eventually, the optimal solutions, i.e., the solutions that fulfil the safety constraint and respect the priority list of the scenario, are: the traffic light is responsible and uses a contract without feedback, either via direct or indirect communication, or both (respectively Td, Ti or Tdi). These solutions differ depending on the entity type that makes progress when communication is not available.

Contract Td: A contract Td where traffic lights are responsible, implies that traffic lights send messages to cars. In this case, traffic lights are responsible to ensure that the safety constraint will not be violated in case of degraded communication. This means that traffic lights cannot turn to red, unless wireless communication is guaranteed in a wide-enough zone.

Contract Ti: In a contract Ti, traffic lights send signals to cars, signalling their current state and planned actions. This corresponds to the traditionally chosen solution for this problem: traffic lights send signals to cars in the form of a colour beam, which changes colour to warn cars before the traffic light turns red e.g., the amber light. In this scenario, cars are responsible to ensure the safety constraint when communication is degraded. This means that if they cannot sense far enough to detect a traffic light and see its colour from far enough away so that they can stop before it, for example because there is fog, cars must either slow down or stop driving all together.

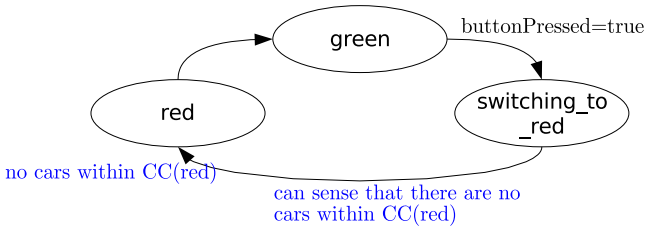


Figure 7.5: Requirements for the traffic light entity type in the pedestrian traffic light scenario.

Contract Tdi: Using a contract Tdi means that traffic lights rely on signalling to make progress. Cars must ensure that they will receive potential signals early enough to have time to stop, and therefore adapt their speed to the visibility if the signal is visual. In addition, however, traffic lights can use direct communication when their coverage is larger than the indirect communication coverage. This allows traffic lights to switch to red faster after a pedestrian has pressed their button. This example demonstrates how in a contract Tdi, entities rely on indirect communication to guarantee safety and can exploit direct communication for optimisation.

Contract without transfer: For the evaluation, a contract without transfer where the responsible entity type is the traffic light was chosen. While this solution is not optimal for this scenario, it is used to test solutions generated with a contract without transfer. In this solution, the traffic light turns to red to let pedestrian pass only when no cars are approaching the light. This requires that traffic lights are able to sense cars, and therefore makes use of the presence sensor.

7.3.1.5 Step 5: requirements on entity behaviours

From the responsible entity and contract type combination, ComhorMod automatically derives the requirements on entity behaviours, in terms of conditions for transitioning to, or remaining in, a mode. These requirements are displayed in step 5. Figure 7.5 shows (in colour) the requirements generated for a contract without feedback for entities of type traffic light and car respectively. Note that ComhorMod generates requirements in XML, but a graphical view is presented here for ease of comprehension. These requirements are that a traffic light cannot switch to red or remain red unless it know that there is no car in the critical coverage $CC(\texttt{red})$.

7.3.1.6 Step 6: parameters estimation

In step 6, developers are requested to enter the values of the application-specific parameters needed to calculate the parameters of the requirements. So for this example, as per Equation 5.2, four application-wide parameters are requested:

- the safety zone SZ, which in this scenario corresponds to the width of the pedestrian crossing;
- the time required for an entity to become present *present*;

Parameter	Value
Application-wide parameters	
SZ	2 m
present	0.5 s
period	0.5 s
latency	0.1 s
Parameters specific to the mode `red`	
$R\_reaction(\texttt{red})$	2 s
$v_{\max}(\texttt{red})$	50 km/h

Table 7.5: Numerical values of the parameters for the pedestrian traffic light scenario.

- the sensing period *period*;
- and the sensing latency *latency*.

In addition, two parameters specific to the mode `red` (as it is the only non fail-safe mode) are requested:

- the responsible entity reaction time for the mode red $R\_reaction(\texttt{red})$, i.e., the time for a traffic light to turn to green once it knows that a car is approaching;
- and the maximum speed at which cars can approach a traffic light in mode red, $v_{\max}(\texttt{red})$.

The values chosen for these parameters are presented in Table 7.5. Using these values, the tool derived a critical coverage of size 156 meters.

7.3.1.7 Step 7: requirement with numerical values

Step 7 presents the requirements derived in step 5, where the value of the critical coverage CC has been replaced by the numerical value derived in step 6.

7.3.1.8 Step 8: sentient object skeletons

Step 8 generates a sentient object skeleton for each of the entity types. The skeleton for cars captures only the three context of cars behaviour. The skeleton for traffic lights captures the three modes and the associated requirements, which have been translated into behaviour rules.

The behaviour of cars was completed in somod by adding a rule to react to events of type "way point" and rules to navigate to a way point.

7.3.2 Evaluating the solutions

Using the entities skeletons completed in SOMod, the implementation of entities of the types car and traffic light were automatically generated. These implementations were tested in the Sentient Simulator, which launched several entities and simulated their interactions. One pedestrian traffic light is set on a stretch of road, on which a number cars travel. At random times, an event simulating

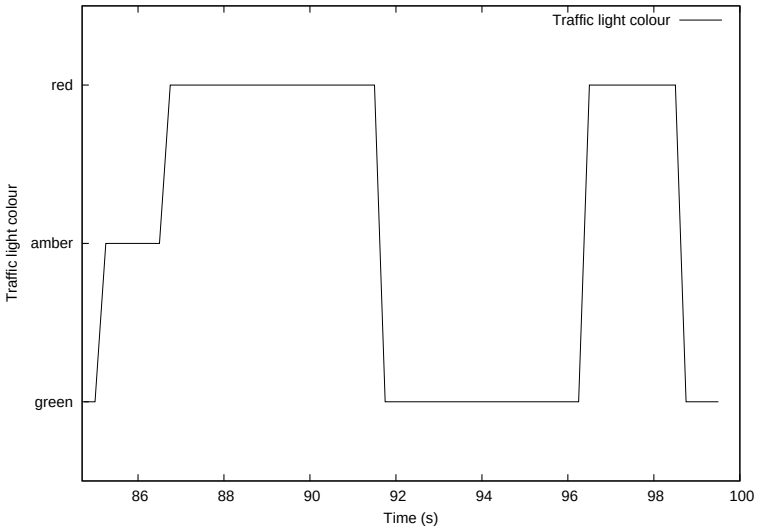


Figure 7.6: Traffic light colour changes.

that the button of the traffic light has been pressed is generated. Upon reception of such an event, the traffic light switches to the mode `switching_to_red`, and waits until there is no car approaching the crossing.

Figure 7.6 shows an excerpt of a simulation. At time 85 s, a pedestrian pressed the button. As a car is approaching the traffic light, the light only changes to amber (which corresponds to being in the `switching_to_red` mode), waiting for the car to have passed it. At time 86.75 s, the car has passed the light and there is no other car approaching so the light turns to red. At time 91.75 s a car approaches so the traffic lights reverts to green. At time 96.5 s, another pedestrian presses the button, and as no cars are in the vicinity of the light it switches to red immediately, until time 98.75 s when another car approaches. As the traffic light needs to wait until no car is approaching, the duration for which a pedestrian will have to wait before being able to cross depends on the traffic load. Experiments have been conducted with varying traffic pattern and loads. Figure 7.7 shows how the average pedestrian waiting time increases with the traffic load. Each point corresponds to the average over 25 simulations with the arrival times of cars and pedestrian generated randomly, with a uniform distribution over the simulated time. Note that the standard variation is large as it depends on the distribution of cars and pedestrians over time (e.g., whether a pedestrian requests to cross as a long platoon of cars arrives).

The most important evaluation of the solution is in terms of safety. Over 250 simulations were performed, representing over 14 hours of simulated time, involving almost 10 000 cars and over 4500 pedestrians. Over all these simulations, whose parameters are described in Table 7.6, no cars passed through a red light, hence showing that the safety constraint of the scenario was respected.

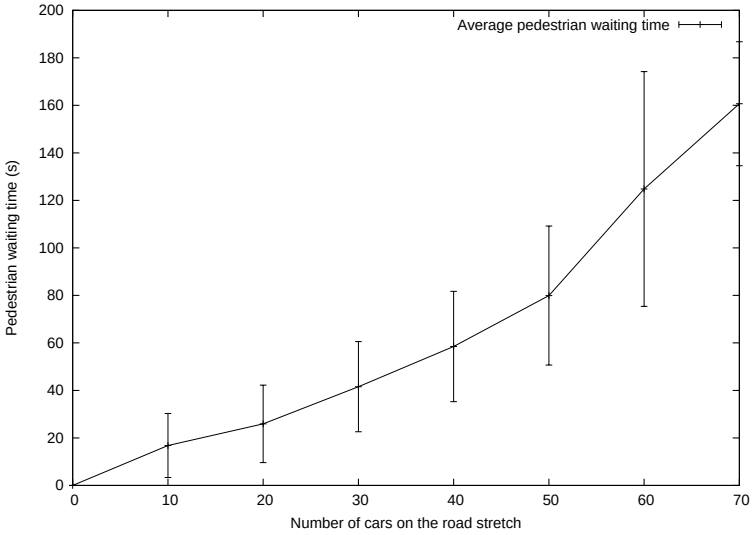


Figure 7.7: Average pedestrian waiting time.

Number of simulations	Number of pedestrians	Number of cars
15	10	10
27	10	20
27	20	20
27	30	20
25	40	20
25	50	20
25	60	20
25	70	20
34	80	20

Table 7.6: Numerical values of the parameters for the pedestrian traffic light scenario.

Mode	Invariant
`stopped`	speed $= 0$
`going_at_v`$_i$	speed $= v_i$
`braking_to_v`$_{i-1}$	speed $\leq v_i$
`accelerating_to_v`$_i$	speed $\leq v_i$

Table 7.7: Mode invariants for the emergency vehicle entity type in the early emergency vehicle arrival warning scenario.

7.4 Early emergency vehicle arrival warning

The goal of the emergency-vehicle warning system (Bouroche & Cahill 2006, Senart et al. 2008) scenario is for ordinary vehicles to be warned when emergency vehicles are approaching their location so that they can free space for them. The earlier cars are warned, the faster emergency vehicles can drive, which can be crucial in emergency situations. This application could be used both as an aid for drivers, or to govern autonomous cars and emergency vehicles. In addition to the interest of this scenario in terms of its application, it is also a compelling example to study, as it includes mobile entities of different types, that need to coordinate their behaviour in real-time in a safety-critical application.

This section first presents how this scenario was specified in ComhorMod, and solutions were generated. Then, two of these solutions are evaluated.

7.4.1 Modelling the scenario in ComhorMod

This scenario contains two types of entities: cars and emergency vehicles. In the following, only interactions between emergency vehicles and cars are considered (i.e., emergency vehicle to emergency vehicle and car to car interactions are ignored). The goal of the scenario is for emergency vehicles to travel as fast as possible, under the safety constraint that emergency vehicles should not crash into cars.

This section presents the successive steps of the design and development of this application using ComhorMod.

7.4.1.1 Step 1: entity definition

In this application, the behaviour of emergency vehicles, given its the maximum speed $v_{\max}$, and two speeds $v_0 = 0$ and v_1, is modelled as: `stopped`, `going_at_v`$_1$, `accelerating_to_v`$_1$, `braking_to_v`$_0$, `going_at_v`$_{max}$, `accelerating_to_v`$_{max}$, `braking_to_v`$_1$. The state of an emergency vehicle encompasses its mode, position, speed, and destination. The mode diagram and mode invariants of the emergency vehicle entity type are described in Figure 7.8 and Table 7.7 respectively.

The goal of an emergency vehicle is to arrive at its destination, it can be expressed as:

$$\text{position} = \text{destination}\,.$$

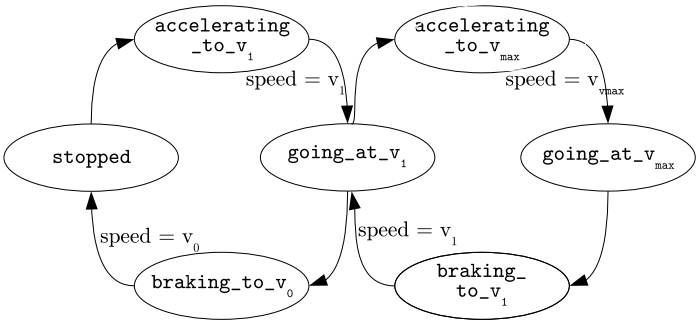


Figure 7.8: Mode diagram for an emergency vehicle entity type in the early emergency vehicle arrival warning scenario.

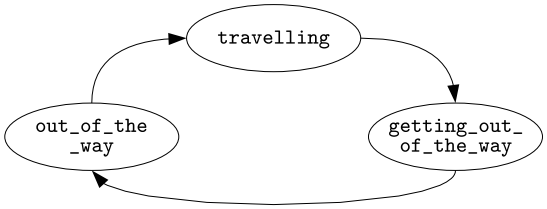


Figure 7.9: Mode diagram for the traffic light entity type in the early emergency vehicle arrival warning scenario.

The behaviour of a car can be modelled with the modes `travelling`, `getting_out_of_the_way`, and `out_of_the_way`. These modes are not the same as in the previous example, as they are defined to capture the relevant parameters for this scenario in terms of the safety constraint; here, whether the car is in the way of the emergency vehicle. The state of car entities encompasses their mode, position, speed, and destination. The mode diagram is depicted in Figure 7.9. Modes of the entity type car can be defined without any invariant. The goal of a car is to arrive at its destination, which can be expressed as:

$$\text{position} = \text{destination}\,.$$

7.4.1.2 Step 2: safety constraint specification

The safety constraint that emergency vehicles should not collide into cars can be stated as that they should not be closer than a certain distance SZ unless the emergency vehicle is stopped or the car is out of the way of the emergency vehicle (in which cases the emergency vehicle will not collide). Using the state and modes defined above, the safety constraint can be defined in ComhorMod as:

$$s_{\text{car}}\mathcal{C}s_{\text{ev}} \text{ iff } \neg\Big(\big(\text{distance}(s_{\text{car}}.\text{position}, s_{\text{ev}}.\text{position}) < SZ\big) \wedge \\ (s_{\text{ev}}.\text{mode} \neq \texttt{stopped}) \wedge (s_{\text{car}}.\text{mode} \neq \texttt{out\_of\_the\_way})\Big)\,.$$

The priority list of the scenario can be specified as

$$(\texttt{going\_at\_v}_{\max}, \texttt{going\_at\_v}_1, \texttt{travelling})\,.$$

	Modes of entities of type car		
	`travelling`	`getting_out _of_the_way`	`out_of_the _way`
Modes of entities of type emergency vehicle			
`stopped`	✓	✓	✓
`going_at_v`$_i$	×	×	✓
`braking_to_v`$_{i-1}$	×	×	✓
`accelerating_to_v`$_i$	×	×	✓

Table 7.8: Mode compatibility matrix for entities of type car and traffic light in the emergency vehicle scenario.

This list captures the fact that it is more urgent for emergency vehicles to arrive at their destinations, than cars.

7.4.1.3 Step 3: mode compatibility evaluation

In step 3, the mode compatibility matrix for entities of type emergency vehicle and cars is automatically derived (see Table 7.8). Note that `stopped` for emergency vehicles and `out_of_the_way` for cars are fail-safe modes.

7.4.1.4 Step 4: responsibility and contract type attribution

Step 4 first derives the solution set and lets users choose one of these solutions.

Deriving the solution set As the two entities have a fail-safe mode, any of the contracts can be used.

Choosing a solution Given that the first item in the priority list is a mode of emergency vehicles, the contract should favour the progress of emergency vehicles. As stated in Table 5.3, this is the case for the following combinations: contract without transfer if cars are responsible, as well as contracts with transfer if emergency vehicles are responsible.

Contract without transfer In the case of a contract without transfer where cars are responsible, cars can only make progress when they know that there is no emergency vehicle, and emergency vehicles do not signal themselves. This solution implies that the progress of cars is very limited (typically, they can drive only when they have good enough visibility to ensure that they will not impact the progress of any emergency vehicle in any way).

Contract with transfer without feedback If a contract without feedback is used, emergency vehicles warn cars of their arrival, by sending them messages either by direct or indirect communication, and cars must ensure that they get out of the way of an emergency vehicle within a pre-agreed time (t_{warning}) after being warned of its arrival.

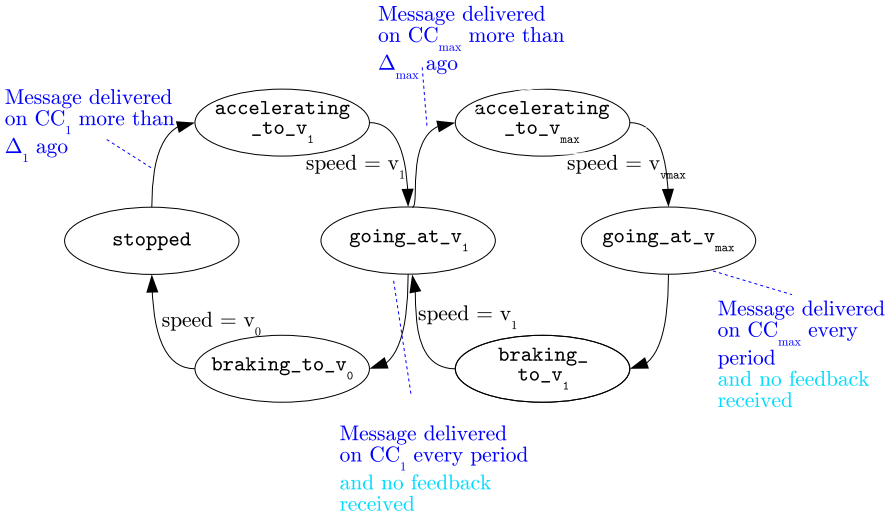


Figure 7.10: Requirements for the emergency vehicle entity type.

Contract with transfer with feedback If a contract without feedback is used, emergency vehicles warn cars of their arrival, like they would when using a contract without feedback, but cars have the choice of either getting out of the way of the emergency vehicle within a pre-agreed time (t_{warning}) or, send feedback within another pre-agreed time (t_{feedback}). Since cars may not be able to get out of the way of emergency vehicles within a given time, a contract with feedback may be more appropriate for this scenario.

The contracts that are optimal for this scenario are contracts with transfer with feedback where emergency vehicles are responsible. If the goal is that emergency vehicles be able to make progress even when communication is degraded, the contract with transfer via indirect communication and feedback via direct communication (TiFd) will be preferred. Direct communication can also be used for the transfer as an optimisation (contract TdiFd).

For the purpose of the evaluation, as indirect communication is not available to test the contract types with transfer and with feedback, two solutions have been implemented and tested: emergency vehicles responsible with contract Td and with contract TdFd.

7.4.1.5 Step 5: requirement on entity behaviours

In step 5, the tool automatically generates the requirements on entities behaviour for each entity type. The requirements generated for entities of type emergency vehicle are displayed on Figure 7.10, where the requirements that apply only to the contract with feedback are displayed in light blue. In addition, Figure 7.11 displays the requirements generated for entities of type car (the same colour convention applies).

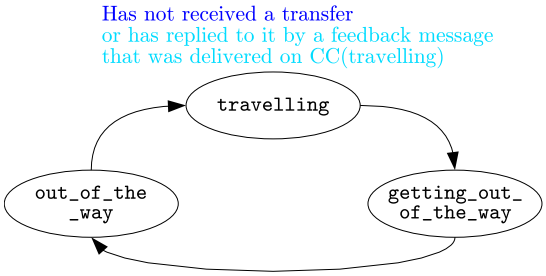


Figure 7.11: Requirements for the car entity type.

7.4.1.6 Step 6: parameter estimation

In step 6, developers are requested to enter the values of the application-specific parameters needed to calculate the parameters of the requirements. For the solution with transfer with feedback, for example, as per Equation 5.2, seven application-wide parameters are requested:

- the safety zone SZ, which in this scenario corresponds to the size of emergency vehicles ;
- the time required for an entity to become present *present*;
- the period of messages sent *period*;
- and the adaptation notification time $adaptNotif$;
- and the latency of message delivery *latency*;
- the contract parameters t_{warning} and t_{feedback}

In addition, two parameters specific for each of the non fail-safe modes are requested:

- the responsible entity reaction time for the mode m, $R\_reaction(m)$, i.e., the time for an emergency vehicle to slow down to a lower speed;
- and the maximum speed at which cars can approach an emergency vehicle when in mode m, $v_{\max}(m)$.

Note that the maximum speed and therefore the reaction time, are the same for `going_at_v`$_1$, `accelerating_to_v`$_1$ and `braking_to_v`$_0$, on one hand and `going_at_v`$_{\max}$, `accelerating_to_v`$_{\max}$, `braking_to_v`$_1$on the other hand. The values chosen for these parameters are presented in Table 7.9. The value used for the reaction time corresponds to the time it would take for a vehicle to slow down by 50 km/h, assuming a constant acceleration of the standard value $a_0 = 6\,\text{m/s}$. Using these values, the tool derived the values of the size of the critical coverage $CC_1 = 73.8\,\text{m}$ and $DC = CC_{\max} = 221.4\,\text{m}$, as well as $\Delta = 2.1\,\text{s}$.

7.4.1.7 Step 7: requirements with numerical values

In step 7, the requirements for the different entities with numerical values are generated.

Parameter	Value
Application-wide parameters	
SZ	2 m
$present$	0.5 s
$period$	2 s
$adaptNotif$	0.5 s
$latency$	0.1 s
t_{warning}	2 s
t_{feedback}	1 s
Parameters specific to the mode `going_at_v1`	
$R\_reaction$(`going_at_v1`)	2.31 s
$v_{\max}$(`going_at_v1`)	50 km/h
Parameters specific to the mode `going_at_vmax`	
$R\_reaction$(`going_at_vmax`)	2.31 s
$v_{\max}$(`going_at_vmax`)	100 km/h

Table 7.9: Numerical values of the parameters for the emergency-vehicle warning scenario.

7.4.1.8 Step 8: sentient object skeletons

In step 8, a sentient object skeleton is generated for each entity type. The behaviour of both emergency vehicles and cars were completed using SOMod, to add the management of way points.

7.4.2 Evaluating the solutions

Both the solutions with a contract Td and a contract TdFd were implemented and tested. The maximum speed for cars is set to 50 km/h. To get out of the way of an emergency vehicle, cars change lane.

7.4.2.1 Solution with contract Td

Firstly, a solution with a contract with transfer via direct communication without feedback (Td) has been generated and evaluated. As the emergency vehicle is responsible and uses transfer via direct communication, it adapts its speed depending on the current actual coverage. Figure 7.12 depicts an excerpt of the traces from one simulation . Note the actual coverage taking one of three values. The lowest one, around 30 m is smaller than $CC(v_1)$; the middle one, around 130 m is greater than $CC(v_1)$ but less than $CC(v_{\max})$; and the highest one, around 170 m, is equal to $CC(v_{\max})$. If the small variations, that are due to measurement errors due to the limited accuracy of the trace files, are discarded, it can be seen that the speed of the emergency vehicle speed also varies between three values: v_0, v_1 and $v_{\max}$. Note that variations of this speed are not instantaneous as the acceleration and deceleration rates are bounded to reflect the kinematics of vehicles. The variations of the emergency vehicle speed are coupled to the actual coverage, but with a small offset, as it takes $adaptNotif$ for the emergency vehicle to be warned of the coverage variation.

The average emergency vehicle speed depends on the communication profile: if the communication is consistently good, emergency vehicles will be able to travel at $v_{\max}$, otherwise they will have to slow

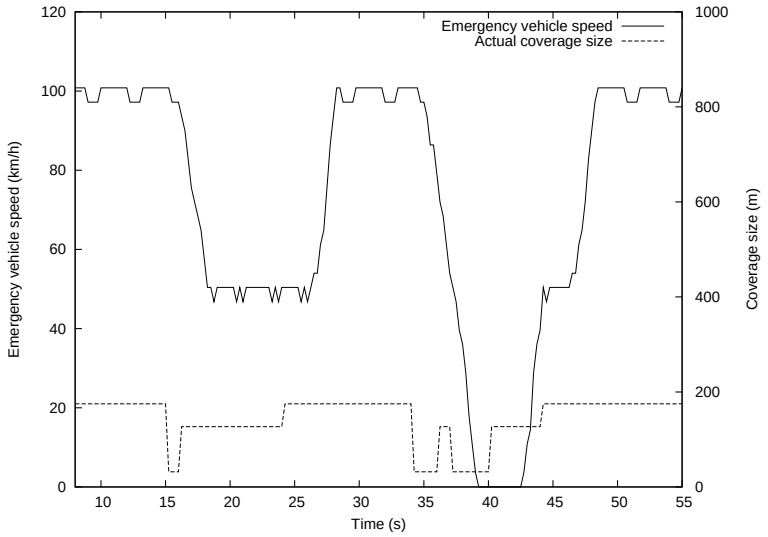


Figure 7.12: Variations of both the actual coverage and the emergency vehicle speed over time, during one simulation.

down to ensure that cars will have time to be warned and change lane before they arrive. Figure 7.13 shows the average speed of emergency vehicle as a function of the probability that the desired coverage be covered (i.e., p_{DC} c.f. Section 7.2). Each point on the graph corresponds to the average speed of an emergency vehicle averaged over 10 simulations; the standard deviation is also depicted. Note that when the communication is perfect the average speed is slightly lower than 100 km/h because at the start of each simulation emergency vehicles are stopped.

Overall, over 220 simulations of the emergency vehicle travelling on a stretch of road have been performed, totalling over 18 hours of simulated time. In these simulations, an emergency vehicle travelled amongst over 6000 cars, and never crashed into any of them, hence demonstrating the correctness of the solution.

7.4.2.2 Solution with contract TdFd

A solution with a contract TdFd, where cars can send feedback if they cannot get out of the way of the emergency vehicle has also been tested. To test the behaviour of the emergency vehicle, a proportion of cars sent feedback that they could not get out of the way of the emergency vehicle the first time they received a message from an emergency vehicle. Upon receiving feedback, an emergency vehicle starts breaking. It does not accelerate again until one of its subsequent messages has been delivered to a critical coverage and no feedback has been received within the allocated time, i.e., within t_{feedback} after the delivery time of the message. Figure 7.14 shows an excerpt of a simulation where, around time 34s, an emergency vehicle received a feedback message from a vehicle and slowed down, even

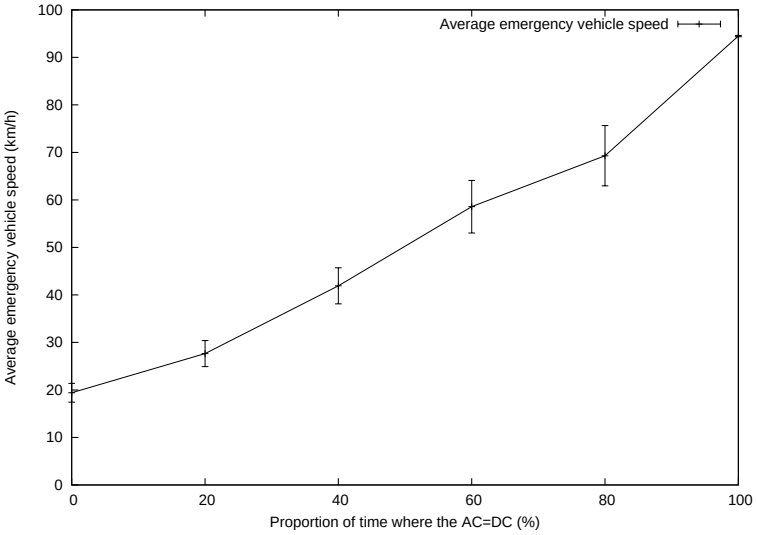


Figure 7.13: Variations of the average emergency vehicle speed as a function of the communication profile.

though the critical coverage remained constant. At time 39s, the emergency vehicle has sent another message and received no feedback, so it accelerates again. At time 43s, it slows down again, because it has been notified of an actual coverage change. Once it is notified that the actual coverage has reverted back to DC, it accelerates again.

Every time an emergency vehicle receives a feedback message from another vehicle, it needs to slow down. Therefore, the average speed of an emergency vehicle varies as a function of the proportion of cars that send feedback. Figure 7.15 shows the variations of the average emergency vehicle speed as a function of the number of vehicles that send feedback. Each point is the average of 20 simulations. Note that the standard deviation is important as the speed of the emergency vehicle varies significantly depending on whether it meets the vehicles that send feedback, and also whether these send feedback at the same time; hence making the emergency vehicle stop once for a number of feedback messages). These simulations have been run with a communication profile with $p_{DC} = 60\%$, and it can be seen that when no vehicle sends feedback to an emergency vehicle, its average speed is the same as with the previous solution.

Overall, over 420 simulations of this solution have been run, totalising almost 24 hours of simulated time. In these simulations, an emergency vehicle travelled amongst over 7700 cars, and never collided into any of them, hence showing the reliability of the solution.

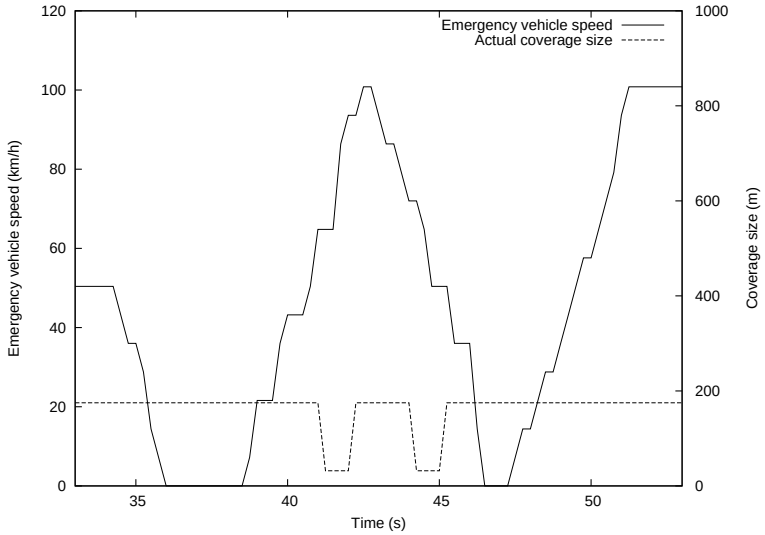


Figure 7.14: Variations of both the actual coverage and the emergency vehicle speed over time, during one simulation.

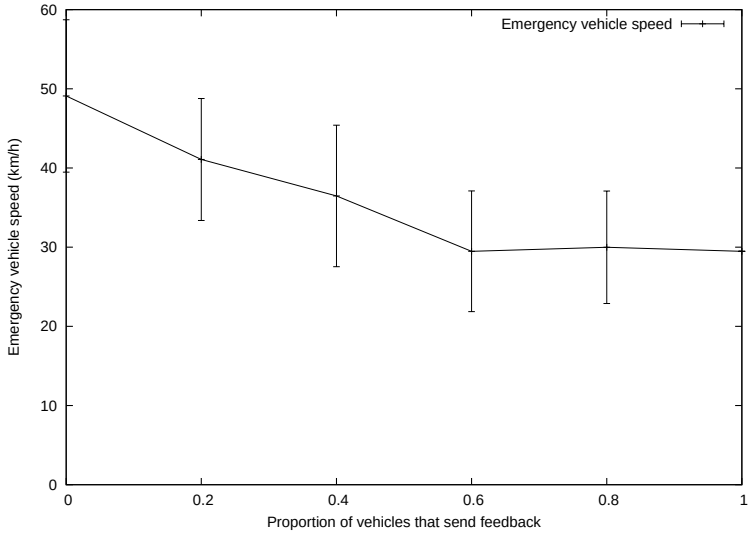


Figure 7.15: Variations of the average emergency vehicle speed as a function of the proportion of vehicles that send feedback.

7.5 Results

The previous sections have shown that Comhordú can be used to model scenarios corresponding to realistic applications from the ITS domain encompassing both mobile and fixed entities, hence validating claim 1. While only two scenarios have been presented in this chapter, many more have been modelled, such as cars arriving at an unsignalised junction, autonomous driving on a single lane, autonomous overtaking, and obstacle detection for example. Comhordú was suitable to capture the system-wide safety constraints of all these scenarios, and can be used to translate them into requirements on the behaviour of individual entities.

In addition, the two scenarios used in the evaluation were specified using ComhorMod, which generated skeleton implementations. These implementation skeletons were completed and the full implementation generated almost automatically using the MoCoA tool chain, hence validating claim 2. The only step of the development that was not automated was the translation of sentient object skeletons into calls to the SOGen Java API because this feature is not yet implemented (c.f. Section 7.2). This manual translation process, which is completely systematic and will therefore be fully automated, proved particularly cumbersome; this highlights the usefulness of the tool chain.

Amongst the solutions generated, three, corresponding to each contract type, were thoroughly tested, under varying communication and traffic patterns. These experiments showed the reliability of the solutions for each of the scenarios, allowing both mobile and stationary entities to make progress when communication is sufficient, while ensuring system-wide safety constraints even when communication is not sufficient, hence validating claim 3.

Finally, both scenarios studied require communication between entities (because, as noted above, the solution to the pedestrian scenario that relies solely on sensing data does not allow pedestrian to ever cross the road when there are many cars). While the existing solutions, such as use of siren and light beam, to these scenario are based on indirect communication, the use of direct communication could allow emergency vehicles to drive faster when direct communication is available on a wide area; hence potentially saving lives and could allow better pedestrian protection. It is not realistic, however, to assume that real-time communication can be guaranteed in the presence of obstacles to message transmission such as vehicles, trucks and buildings. Therefore, existing consensus-based solutions cannot be applied to these scenarios as they would not provide sufficient reliability. In addition, the TCB does not guarantee timely notification of timing errors, and therefore could not provide the reliability required by these scenario either, hence claim 4 is validated.

7.6 Summary

This chapter demonstrated how Comhordú can be applied to solve scenarios with strong real-time and reliability requirements, where safety constraints span several entities. These scenarios and their safety constraints have been modelled using the abstractions defined in Comhordú. The systematic process

described in Chapter 5 was used to translate these system-wide safety constraints into requirements on the behaviours of individual entities. Different solutions can be selected depending on the scenario's goals and priority list. The development process was facilitated by ComhorMod that automatically generated the requirements and their numerical values, as well as a skeleton allowing these entities to be implemented. The implementations of solutions corresponding to each of the contract types have been thoroughly tested, demonstrating their correctness. Finally, this chapter demonstrated that the scenario studied cannot be solved with existing coordination models.

Chapter 8

Conclusions and Future Work

This thesis presented Comhordú, a coordination model for autonomous mobile entities, which supports the development of entities that can adapt their behaviour depending on currently available information to ensure system-wide safety constraints. This chapter summarises the most significant achievements of the work described in this thesis and assess its contribution to the state of the art. In addition, some perspectives on the model presented in the body of the thesis are given, and some suggestions for future work are outlined.

8.1 Achievements

The motivation for the work described in this thesis arose out of the increasing presence of autonomous mobile entities in our everyday environment. As these operate in the same environment as each other and as humans, such entities need to coordinate their behaviour to ensure system-wide safety constraints. As the safety of humans and possibly crucial or expensive infrastructure is at stake, the coordination of entities is safety-critical, i.e., a violation of the safety constraints could result in a catastrophe. In addition, because entities interact with their environment, they need to cope with its pace, therefore their safety constraints imply stringent real-time requirements on coordination.

A review of existing work in the MAS, MRS, ITS, coordination models, and real-time systems communities has shown that none of the existing work is suitable for addressing the challenges of high-reliability and timeliness posed by the coordination of such autonomous mobile entities in a generic way. In particular, it was shown that the consensus paradigm used by most existing work is not appropriate to the kind of mobile environments.

The coordination of autonomous mobile entities is particularly challenging because the amount of information available to entities about the behaviour of other entities and their environment varies considerably over time and distance, due to the limitations of communication over wireless networks and of sensing. For this reason, this thesis builds on the space-elastic model, an existing real-time communication model that provides real-time feedback on currently available information. A sensing

and indirect communication model that provides the same feature was defined, providing entities with an alternative communication means to supplement direct communication.

Building on both the space-elastic model and the sensing and indirect communication models, Comhordú, a coordination model that allows entities to adapt their behaviour depending on available information was defined. This model enables the distribution of system-wide safety constraints over autonomous entities, via contracts between entities. In addition, a systematic development process for applications composed of autonomous mobile entities using Comhordú was presented. This process allows application developers to translate system-wide safety constraints into requirements on the individual behaviour of autonomous mobile entities. The design of a development tool, ComhorMod, that supports the development process of autonomous mobile entities by automating the systematic steps of Comhordú was also presented.

The usefulness of the coordination model was demonstrated on two scenarios from the ITS domain that cannot be solved using existing coordination models. These scenarios were implemented using ComhorMod, which was shown to be suitable for such tasks and significantly eased the development efforts required, by allowing solutions to be automatically generated. The generated solutions were extensively tested to demonstrate that they allow entities to make progress while guaranteeing the scenarios' safety constraints.

8.2 Perspectives

Notwithstanding the achievements presented in the previous section, the model presented in this thesis will serve mostly as a starting point for research in coordination models for autonomous mobile entities.

Coordinating autonomous mobile entities is a very challenging problem because it requires stringent reliability and timeliness requirements to be met by applications deployed in unknown, dynamic environments, where the amount of information available varies significantly over time and distance. To make matters worse, as entities are autonomous, any solution must be fully distributed, and because entities are not necessarily aware of each other a priori, they must adopt a defencive approach where the safety constraint could be violated at any time unless actively enforced.

This thesis investigated how this problem can be tackled assuming direct and indirect real-time communication models that provide timely feedback on the zone in which communication is available. Instead of relying on consensus, entities make progress independently while ensuring the safety constraints by obeying to contracts imposing requirements on their behaviour. These requirements dictate how entities must react when the amount of available information via sensing and direct and indirect communication is degraded. This approach was shown to allow entities to make progress when it is safe to do so while always ensuring system-wide safety constraints.

While an implementation of the direct communication model assumed is available, the sensing and

indirect communication model has not yet been implemented. The guarantees of this model might prove too strong to be provided reliably, and might need to be relaxed to be implementable. In particular, inspiration could be drawn from existing work that deals with sensor unreliability at the coordination level by having entities attach information about their environment that caused them to initiate a specific action, to coordination messages related to that action (Farinelli et al. 2007). This approach allows information about the environment to be shared amongst several entities, and possibly fused. A simplistic example of such an approach would be that when a traffic light at a junction changes colour to allow an incoming emergency vehicle to pass through the junction, cars are informed incidentally by the traffic light message that an emergency vehicle is arriving and can get out of its way.

8.3 Future work

As is always the case in research, and particularly with this work that investigated a new research direction, many issues are worthy of a more detailed investigation.

Comhordú presents a number of results that are explained, but not formally proven. A significant extension to this work would be to provide a formal proof of Comhordú, by analytically demonstrating that the requirements derived for each contract type are sufficient to ensure the safety constraint.

In addition, while the abstractions defined in Comhordú allow to solve numerous applications, their scope is limited, and could be extended in the following ways:

- define more invariants to capture more detailed semantics of modes, such as bounded variation rates over time and distance;
- extend the set of basic incompatibilities to support, for example, safety constraints that impose a condition on the sum of the values of two state variables;
- extend the set of contracts to allow more of the entities behaviour to be automatically generated.

Similarly, as detailed in Chapter 6, the implementation of the MoCoA tool chain needs to be completed, and tool support can be extended to ease programming of autonomous mobile applications even more. In addition, an interesting research area is how to program the behaviour of entities that obey several contracts to fulfil several safety constraints. As the mode set increases exponentially with each safety constraint, the behaviour of entities becomes more and more complicated. For this reason, dynamic reconfiguration of entity behaviour is a promising research area.

Finally, the Model-Driven Engineering (MDE) community, similarly to the work described in this thesis, uses high-level models to specify a system and aims to automatically generate implementations from such models. This work could be expressed using MDE concepts and integrated with existing work in this community.

8.4 Summary

This chapter summarised the motivations for, and the most significant achievements of the work described in this thesis. In particular, it outlined how this work contributes to the state of the art of the coordination of autonomous mobile entities, by exploring an alternative to consensus. In addition, some perspectives on the coordination model presented in the thesis were given, and some suggestions for future work presented.

Bibliography

Alighanbari, M., Kuwata, Y. & How, J. P. (2003), Coordination and control of multiple uavs with timing constraints and loitering, *in* 'Proceedings of the American Control Conference', Vol. 6, IEEE, pp. 5311–5316.

Arbab, F. (1998), What do you mean, coordination?, *in* 'Bulletin of the Dutch Association for Theoretical Computer Science (NVTI)'.

Aufrere, R., Gowdy, J., Mertz, C., Thorpe, C., Wang, C.-C. & Yata, T. (2003), 'Perception for collision avoidance and autonomous driving', *Mechatronics* **13**(10), 1149–1161.

Babaoglu, Ö., Meling, H. & Montresor, A. (2002), Anthill: A framework for the development of agent-based peer-to-peer systems, *in* L. E. Rodrigues, M. Raynal & W.-S. E. Chen, eds, 'Proceedings of the International Conference on Distributed Computing Systems (ICDCS)', IEEE Computer Society, pp. 15–22.

Baber, J., Kolodko, J., Noel, T., Parent, M. & Vlacic, L. (2005), 'Cooperative autonomous driving: Intelligent vehicles sharing city roads', *IEEE Robotics & Automation Magazine* **12**(1), 44–49.

Beard, R. W., McLain, T. W., Goodrich, M. A. & Anderson, E. P. (2002), 'Coordinated target assignment and intercept for unmanned air vehicles', *IEEE Transactions on Robotics and Automation* **18**(6), 911–922.

Becker, L. B., Nett, E., Schemmer, S. & Gergeleit, M. (2005), 'Robust scheduling in team-robotics', *Journal of Systems and Software* **77**(1), 3–16.

Beckers, R., Holland, O. E. & Deneubourg, J.-L. (1994), From local actions to global tasks: Stigmergy and collective robotics, *in* R. A. Brooks & P. Maes, eds, 'Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems (Artificial Life IV)', MIT Press, pp. 181–189.

Bickford, C., Teo, M. S., Wallace, G., Stankovic, J. A. & Ramamritham, K. (1996), A robotic assembly application on the Spring real-time system, *in* 'Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium (RTAS '96)', IEEE Computer Society, pp. 19–28.

Biegel, G. & Cahill, V. (2004), A framework for developing mobile, context-aware applications, *in* 'Proceedings of the 2^{nd} IEEE International Conference on Pervasive Computing and Communications (PerCom 2004)', IEEE Computer Society, pp. 361–365.

Bouraoui, L., Petti, S., Laouiti, A., Fraichard, T. & Parent, M. (2006), Cybercar cooperation for safe intersections, *in* 'Proceedings of the IEEE Conference on Intelligent Transportation Systems (ITSC)', IEEE, pp. 456–461.

Bouroche, M. & Cahill, V. (2006), Coordination of autonomous mobile entities, *in* B. Koldehofe, ed., 'Proceedings of the 4th MiNEMA Workshop', pp. 59–64.

Bouroche, M., Hughes, B. & Cahill, V. (2006), Building reliable mobile applications with space-elastic adaptation, *in* 'Proceedings of the Mobile Distributed Computing workshop (MDC)', IEEE Computer Society, pp. 627–632.

Brooks, R. R. & Iyengar, S. S. (1998), *Multi-sensor fusion: fundamentals and applications with software*, Prentice-Hall, Inc.

Buckley, S. J. (1989), Fast motion planning for multiple moving robots, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)', IEEE, pp. 322–326.

Cabri, G., Ferrari, L., Leonardi, L., Mamei, M. & Zambonelli, F. (2006), Uncoupling coordination: Tuple-based models for mobility, *in* P. Bellavista & A. Corradi, eds, 'The Handbook of Mobile Middleware', 1 edn, Auerbach, chapter 10, pp. 229–256.

Carlson, J. (2002), Languages and methods for specifying real-time systems, Technical report, Mälardalen Real-Time Research Centre.

Caro, G. D. & Dorigo, M. (1998), 'AntNet: Distributed stigmergetic control for communications networks', *Journal of Artificial Intelligence Research* **9**, 317–365.

Cawkwell, J. (2000), A visually guided AGV for use as passenger transport in urban areas, *in* 'Proceedings of the Intelligent Transportation Systems Conference (ITSC)', IEEE Computer Society, pp. 311–315.

Ciancarini, P. (1996), 'Coordination models and languages as software integrators', *ACM Computing Surveys* **28**(2), 300–302.

Clark, C. M. (2004), Dynamic Robot Networks: A Coordination Platform for Multi-Robot Systems, PhD thesis, Department of Aeronautics and Astronautics, Stanford University, CA, USA.

Clark, C. M., Rock, S. M. & Latombe, J.-C. (2003), Dynamic networks for motion planning in multi-robot space systems, *in* 'Proceedings of the International Symposium of Artificial Intelligence, Robotics and Automation in Space'.

Cristian, F. (1991), 'Understanding fault-tolerant distributed systems', *Communications of the ACM* **34**(2), 56–78.

Cunningham, R. & Cahill, V. (2002), Time bounded medium access control for ad hoc networks, *in* A. Schiper, R. Baldoni & R. Prakash, eds, 'Proceedings of the ACM International Workshop on Principles of Mobile Computing (POMC)', ACM Press, New York, NY, USA, pp. 1–8. Invited paper, not peer reviewed.

Deugo, D., Weiss, M. & Kendall, E. (2001), Reusable patterns for agent coordination, *in* A. Omicini, F. Zambonelli, M. Klusch & R. Tolksdorf, eds, 'Coordination of Internet agents: models, technologies, and applications', Springer-Verlag, pp. 347–368.

Dorigo, M., Caro, G. D. & Gambardella, L. M. (1999), 'Ant algorithms for discrete optimization', *Artificial Life* **5**(2), 137–172.

Dresner, K. & Stone, P. (2004), Multiagent traffic management: A reservation-based intersection control mechanism, *in* 'Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)', ACM, pp. 530–537.

Dresner, K. & Stone, P. (2005), Multiagent traffic management: An improved intersection control mechanism, *in* 'Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)', ACM Press, pp. 471–477.

Empey, D. (2002), 'PATH vehicles will roll at demo 2003', *Intellimotion* **10**(1), 8–12.

Eugster, P. T., Garbinato, B. & Holzer, A. (2005), Location-based publish/subscribe, *in* 'Proceedings of the IEEE International Symposium on Network Computing and Applications (NCA)', IEEE Computer Society, pp. 279–282.

Farinelli, A., Iocchi, L. & Nardi, D. (2004), 'Multirobot systems: a classification focused on coordination', *IEEE Transactions on Systems, Man, and Cybernetics, Part B* **34**(5), 2015–2028.

Farinelli, A., Nardi, D., Scerri, P. & Ingenito, A. (2007), Dealing with perception errors in multi-robot system coordination, *in* M. M. Veloso, ed., 'Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)', pp. 2091–2096.

Fasli, M. (2001), On commitments, roles, and obligations, *in* B. Dunin-Keplicz & E. Nawarecki, eds, 'Revised Papers from the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems (CEEMAS)', Vol. 2296 of *Lecture Notes In Computer Science*, Springer-Verlag, pp. 93–102.

Federal Communications Commission (1999), Fcc 99-305, fcc report and order, Technical report.

Ferber, J. (1999), *Multi-Agent Systems: An Introduction to Distributed Artifical Intelligence*, Addison-Wesley.

Fitzpatrick, A., Biegel, G., Clarke, S. & Cahill, V. (2002), Towards a sentient object model, *in* 'Proceedings of the OOPSLA 2002 Workshop on Engineering Context-Aware Object-Oriented Systems and Environments (ECOOSE)'.

Fok, C.-L., Roman, G.-C. & Hackmann, G. (2004), A lightweight coordination middleware for mobile computing, *in* R. D. Nicola, G. L. Ferrari & G. Meredith, eds, 'Proceedings of the 6th International Conference on Coordination Models and Languages (COORDINATION)', Vol. 2949 of *Lecture Notes in Computer Science*, Springer, pp. 135–151.

Fong, T. & Nourbakhsh, I. (2005), 'Interaction challenges in human-robot space exploration', *Interactions* **12**(2), 42–45.

Frey, D. & Roman, G.-C. (2007), Context-aware publish subscribe in mobile ad hoc networks, *in* A. L. Murphy & J. Vitek, eds, 'Proceedings of the International Conference on Coordination Models and Languages (COORDINATION)', Vol. 4467 of *Lecture Notes in Computer Science*, Springer, pp. 37–55.

Friedman, R. (2003), Fuzzy group membership, *in* A. Schiper, A. A. Shvartsman, H. Weatherspoon & B. Y. Zhao, eds, 'Proceedings of the Future Directions in Distributed Computing Workshop', Vol. 2584 of *Lecture Notes in Computer Science*, Springer, pp. 114–118.

Fujimura, K. (1992), *Motion Planning in Dynamic Environments*, Springer-Verlag.

Gaertner, G. & Cahill, V. (2004), 'Understanding link quality in 802.11 mobile ad hoc networks', *IEEE Internet Computing* **8**(1), 55–60.

Garcia-Molina, H. (1982), 'Elections in a distributed computing system', *IEEE Transactions on Computers* **31**(1), 48–59.

Gelernter, D. (1985), 'Generative communication in Linda', *ACM Transactions on Programming Languages and Systems (TOPLAS)* **7**(1), 80–112.

Gelernter, D. & Carriero, N. (1992), 'Coordination languages and their significance', *Communications of the ACM* **35**(2), 97–107.

Gellersen, H.-W., Beigl, M. & Schmidt, A. (1999), Environment-mediated mobile computing, *in* 'Proceedings of the 1999 ACM symposium on Applied computing (SAC '99)', ACM Press, pp. 416–418.

Gerkey, B. P. & Mataric, M. J. (2003), Multi-robot task allocation: analyzing the complexity and optimality of key architectures, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)', IEEE, pp. 3862–3868.

Gervasi, V. & Prencipe, G. (2004), 'Coordination without communication: the case of the flocking problem', *Discrete Applied Mathematics* **144**(3), 324–344.

Goldberg, D., Cicirello, V., Dias, M. B., Simmons, R., Smith, S., Smith, T. & Stentz, A. (2002), A distributed layered architecture for mobile robot coordination: Application to space exploration, *in* 'Proceedings of the International NASA Workshop on Planning and Scheduling for Space'.

Gonzalez, A. J. & Ahlers, R. (1998), 'Context-based representation of intelligent behavior in training simulations', *Transactions of the Society for Computer Simulation International* **15**(4), 153–166.

Grassé, P.-P. (1959), 'La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la theorie de la stigmergie: Essai d'interpretation des termites constructeurs', *Insectes Sociaux* **6**, 41–83.

Gray, J. (1978), Notes on data base operating systems, *in* 'Operating Systems, An Advanced Course', Vol. 60 of *Lecture Notes In Computer Science*, Springer-Verlag, pp. 393–481.

Guo, Y. & Parker, L. E. (2002), A distributed and optimal motion planning approach for multiple mobile robots, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)', IEEE, pp. 2612–2619.

Hallé, S., Chaib-draa, B. & Laumonier, J. (2003), Car platoons simulated as a multiagent system, *in* 'Proceedings of Agent Based Simulation 4'.

Hallé, S. & Chaib-draa, B. (2005), Collaborative driving system using teamwork for platoon formations, *in* F. Klügl, A. Bazzan & S. Ossowski, eds, 'Proceedings of AAMAS-04 Workshop on Agents in Traffic and Transportation (ATT)', Vol. 8 of *Whitestein Series in Software Agent Technologies*, Birkhäuser.

Hallé, S., Laumonier, J. & Chaib-draa, B. (2004), A decentralized approach to collaborative driving coordination, *in* 'Proceedings of the International IEEE Conference on Intelligent Transportation Systems (ITSC)', IEEE, pp. 453–458.

Hartenstein, H., Bochow, B., Ebner, A., Lott, M., Radimirsch, M. & Vollmer, D. (2001), Position-aware ad hoc wireless networks for inter-vehicle communications: the Fleetnet project, *in* 'Proceedings of the ACM international symposium on Mobile ad hoc networking and computing (MOBIHOC)', ACM, pp. 259–262.

Hazewinkel, M., ed. (1994), *Encyclopaedia of Mathematics*, Springer-Verlag.

Hirose, S. & Fukushima, E. F. (2002), 'Development of mobile robots for rescue operations', *Advanced Robotics* **16**(6), 509–512.

Holland, O. & Melhuish, C. (1999), 'Stigmergy, self-organization, and sorting in collective robotics', *Artificial Life* **5**(2), 173–202.

Hughes, B. (2006), Hard Real-Time Communication for Mobile Ad Hoc Networks, PhD thesis, Dept. of Computer Science, Trinity College Dublin.

Hughes, B. & Cahill, V. (2003), Achieving real-time guarantees in mobile wireless ad hoc networks, *in* 'Proceedings if Real-Time Systems Symposium, Work-in-progress Session (RTSS)', IEEE, pp. 37–40.

Ijspeert, A. J., Martinoli, A., Billard, A. & Gambardella, L. M. (2001), 'Collaboration through the exploitation of local interactions in autonomous collective robotics: The stick pulling experiment', *Autonomous Robots* **11**(2), 149–171.

Ioannou, P. & Stefanovic, M. (2005), 'Evaluation of ACC vehicles in mixed traffic: Lane changes effects and sensitivity analysis', *IEEE Transactions on Intelligent Transportation Systems* **6**(1), 79–89.

Iocchi, L., Nardi, D. & Salerno, M. (2001), Reactivity and deliberation: A survey on multi-robot systems, *in* 'Balancing Reactivity and Social Deliberation in Multi-Agent Systems, From RoboCup to Real-World Applications (selected papers from the ECAI 2000 Workshop and additional contributions)', Vol. 2103 of *Lecture Notes In Computer Science*, Springer-Verlag, pp. 9–34.

ISO (1996), 'Information technology - syntactic metalanguage - Extended BNF'. International Standards Organization and International Electrotechnical Commission (ISO/IEC).

Jacquet, J.-M., Bosschere, K. D. & Brogi, A. (2000), On timed coordination languages, *in* A. Porto & G.-C. Roman, eds, 'Proceedings of the International Conference Coordination Languages and Models (COORDINATION)', Vol. 1906 of *Lecture Notes in Computer Science*, Springer, pp. 81–98.

Jacquet, J.-M. & Linden, I. (2007), Towards a theory of refinement in timed coordination languages, *in* A. L. Murphy & J. Vitek, eds, 'Proceedings of the International Conference on Coordination Models and Languages (COORDINATION)', Vol. 4467 of *Lecture Notes in Computer Science*, Springer, pp. 113–131.

Jennings, N. R. (1993), 'Commitments and conventions: The foundation of coordination in multi-agent systems', *The Knowledge Engineering Review* **8**(3), 223–250.

Julien, C. & Roman, G.-C. (2002), Egocentric context-aware programming in ad hoc mobile environments, *in* 'Proceedings of the ACM SIGSOFT symposium on Foundations of software engineering (FSE)', ACM Press, pp. 21–30.

Julien, C. & Roman, G.-C. (2004), Active coordination in ad hoc networks, *in* R. D. Nicola, G. L. Ferrari & G. Meredith, eds, 'Proceedings of the 6th International Conference on Coordination Models and Languages (COORDINATION)', Vol. 2949 of *Lecture Notes in Computer Science*, Springer, pp. 199–215.

Julien, C. & Roman, G.-C. (2006), 'EgoSpaces: Facilitating rapid development of context-aware mobile applications', *IEEE Transactions on Software Engineering* **32**(5), 281–298.

Karam, N., Chausse, F., Aufrère, R. & Chapuis, R. (2006), Collective localization of communicant vehicles applied to collision avoidance, *in* 'Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)', IEEE, pp. 442–449.

Kato, S., Tsugawa, S., Tokuda, K., Matsui, T. & Fujii, H. (2002), 'Vehicle control algorithms for cooperative driving with automated vehicles and intervehicle communications', *IEEE Transactions on Intelligent Transportation Systems* **3**(3), 155–161.

Keil, D. & Goldin, D. Q. (2003), 'Indirect interaction and decentralized coordination'. extended draft.

Keil, D. & Goldin, D. Q. (2005), Indirect interaction in environments for multi-agent systems, *in* D. Weyns, H. V. D. Parunak & F. Michel, eds, 'Proceedings of the International Workshop on Environments for Multi-Agent Systems (E4MA)', Vol. 3830 of *Lecture Notes in Computer Science*, Springer, pp. 68–87.

Killijian, M.-O., Cunningham, R., Meier, R., Mazare, L. & Cahill, V. (2001), Towards group communication for mobile participants, *in* 'Proceedings of the ACM Workshop on Principles of Mobile Computing (POMC)', pp. 75–82.

Kolodko, J. & Vlacic, L. (2003), 'Cooperative autonomous driving at the intelligent control systems laboratory', *IEEE Intelligent Systems* **18**(4), 8–11.

Konolige, K., Fox, D., Ortiz, C., Agno, A., Eriksen, M., Limketkai, B., Ko, J., Morisset, B., Schulz, D., Stewart, B. & Vincent, R. (2004), Centibots: Very large scale distributed robotic teams, *in* D. L. McGuinness & G. Ferguson, eds, 'Proceedings of the National Conference on Artificial Intelligence, Conference on Innovative Applications of Artificial Intelligence', AAAI Press - The MIT Press, pp. 1022–1023.

Kopetz, H. (1997), *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publisher.

Kopetz, H. (2001), The temporal specification of interfaces in distributed real-time systems, *in* T. A. Henzinger & C. M. Kirsch, eds, 'Proceedings of the International Workshop on Embedded Software (EMSOFT)', Vol. 2211 of *Lecture Notes in Computer Science*, Springer, pp. 223–236.

Kopetz, H. & Kim, K. H. (1990), Temporal uncertainties in interactions among real-time objects, *in* 'Proceedings of the ninth Symposium on Reliable Distributed Systems', IEEE Computer Society Press, pp. 165–174.

Kubota, N., Nojima, Y., Baba, N., Kokima, F. & Fukuda, T. (2000), Evolving pet robot with emotional model, *in* 'Proceedings of the Congress on Evolutionary Computation', Vol. 2, IEEE Computer Society, pp. 1231–1237.

Limniotes, T., Mourlas, C. & Papadopoulos, G. A. (2002), Event-driven coordination of real-time components, *in* 'Proceedings of the International Conference on Distributed Computing Systems Workshops (ICDCSW)', IEEE Computer Society, pp. 589–594.

Lynch, N. A. (1996), *Distributed Algorithms*, Morgan Kaufmann.

Lyons, A. (1998), 'UML for real-time overview'. RATIONAL Software Corporation Whitepaper.

Malone, T. W. & Crowston, K. (1994), 'The interdisciplinary study of coordination', *ACM Computing Surveys* **26**(1), 87–119.

Mamei, M. & Zambonelli, F. (2004), Self-maintained distributed tuples for field-based coordination in dynamic networks, *in* 'Proceedings of the 2004 ACM Symposium on Applied Computing (SAC)', ACM Press, pp. 479–486.

Mamei, M. & Zambonelli, F. (2005), Programming stigmergic coordination with the tota middleware, *in* 'Proceedings of the fourth international joint conference on Autonomous Agents and Multiagent Systems (AAMAS)', ACM Press, pp. 415–422.

Martins, P., Sousa, P., Casimiro, A. & Verissimo, P. (2004), Dependable adaptive real-time applications in wormhole-based systems, *in* 'Proceedings of the International Conference on Dependable Systems and Networks (DSN)', IEEE Computer Society, pp. 567–572.

Martins, P., Sousa, P., Casimiro, A. & Veríssimo, P. (2005), 'A new programming model for dependable adaptive real-time applications', *IEEE Distributed Systems Online* **6**(5).

McDonald, A. B. & Znati, T. F. (1999), 'A mobility-based framework for adaptive clustering in wireless ad hoc networks', *Journal on Selected Areas in Communications* **17**(8), 1466–1487.

Meier, R. & Cahill, V. (2003), Exploiting proximity in event-based middleware for collaborative mobile applications, *in* J.-B. Stefani, I. M. Demeure & D. Hagimont, eds, 'Proceedings of the IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)', Vol. 2893 of *Lecture Notes in Computer Science*, Springer-Verlag, Heidelberg, Germany, pp. 285–296.

Meier, R., Cahill, V., Nedos, A. & Clarke, S. (2005), Proximity-based service discovery in mobile ad hoc networks, *in* L. Kutvonen & N. Alonistioti, eds, 'Proceedings of the International Conference on Distributed Applications and Interoperable Systems (DAIS)', Vol. 3543 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 115–129.

Meier, R., Hughes, B., Cunningham, R. & Cahill, V. (2005), Towards real-time middleware for applications of vehicular ad hoc networks, *in* L. Kutvonen & N. Alonistioti, eds, 'Proceedings of the 5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)', Vol. 3543 of *Lecture Notes in Computer Science*, Springer-Verlag GmbH, pp. 1–13.

Meyer, B. (1992), 'Applying "design by contract"', *Computer* **25**(10), 40–51.

Michaud, F., Lepage, P., Frenette, P., Letourneau, D. & Gaubert, N. (2006), 'Coordinated maneuvering of automated vehicles in platoons', *IEEE Transactions on Intelligent Transportation Systems* **7**(4), 437–447.

Mock, M. (2004*a*), Expressing real-time requirements on object-interactions, *in* 'Proceedings of the 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)', IEEE Computer Society, pp. 201–208.

Mock, M. (2004*b*), *On the Real-Time Cooperation of Autonomous Systems*, Fraunhofer Series in Information and Communication Technology, Shaker Verlag, Aachen.

Mock, M., Frings, R., Nett, E. & Trikaliotis, S. (2000), Continuous clock synchronization in wireless real-time applications, *in* '19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)', IEEE Computer Society, pp. 125–132.

Mock, M., Nett, E. & Schemmer, S. (1999), Efficient reliable real-time group communication for wireless local area networks, *in* J. Hlavicka, E. Maehle & A. Pataricza, eds, 'Proceedings of the Third European Dependable Computing Conference (EDCC-3)', Vol. 1667 of *Lecture Notes In Computer Science*, Springer-Verlag, pp. 380–400.

Murphy, A. L. & Picco, G. P. (2004), Using coordination middleware for location-aware computing: A Lime case study, *in* R. D. Nicola, G. L. Ferrari & G. Meredith, eds, 'Proceedings of the 6th International Conference on Coordination Models and Languages (COORDINATION)', Vol. 2949 of *Lecture Notes in Computer Science*, Springer, pp. 263–278.

Murphy, A. L. & Picco, G. P. (2006), Using Lime to support replication for availability in mobile ad hoc networks, *in* P. Ciancarini & H. Wiklicky, eds, 'Proceedings of the International Conference on Coordination Models and Languages (COORDINATION)', Vol. 4038 of *Lecture Notes in Computer Science*, Springer, pp. 194–211.

Murphy, A. L., Picco, G. P. & Roman, G.-C. (2001), LIME: A middleware for physical and logical mobility, *in* 'Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS)', IEEE Computer Society, pp. 524–533.

Murphy, A. L., Picco, G. P. & Roman, G.-C. (2006), 'LIME: A coordination model and middleware supporting mobility of hosts and agents', *ACM Transactions on Software Engineering and Methodology (TOSEM)* **15**(3), 279–328.

Nadeem, T., Dashtinezhad, S., Liao, C. & Iftode, L. (2004), 'TrafficView: Traffic data dissemination using car-to-car communication', *ACM Mobile Computing and Communications Review (M2CR)* **8**(3), 6–19.

Naranjo, J. E., González, C., de Pedro, T., García, R., Alonso, J., Sotelo, M. A. & Fernández, D. (2006), AUTOPIA architecture for automatic driving and maneuvering, *in* 'Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)', IEEE, pp. 1220–1225.

Neskovic, A., Neskovic, N. & Paunovic, G. (2000), 'Modern approaches in modeling of mobile radio systems propagation environment', *IEEE Communications Surveys and Tutorials* **3**(3), 2–12.

Nett, E., Gergeleit, M. & Mock, M. (2001), Mechanisms for a reliable cooperation of vehicles, *in* 'Proceedings of the IEEE International Symposium on High-Assurance Systems Engineering (HASE)', IEEE Computer Society, pp. 75–81.

Nett, E., Gergeleit, M. & Streich, H. (1997), Flexible resource scheduling and control in an adaptive real-time environment, *in* 'Proceeding of the IASTED International Conference on Artificial Intelligence and Soft Computing'.

Nett, E. & Schemmer, S. (2004), An architecture to support cooperating mobile embedded systems, *in* 'Proceedings of the 1st conference on Computing Frontiers (CF)', ACM Press, pp. 40–50.

Object Management Group (OMG) (2005), 'UML profile for schedulability, performance and time v1.1'.
URL: *http://www.omg.org/docs/formal/05-01-02.pdf*

Østergaard, E. H., Mataric, M. J. & Sukhatme, G. S. (2001), Distributed multi-robot task allocation for emergency handling, *in* 'Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', IEEE, pp. 821–826.

Oxford English Dictionary Online (1989). Accessed on the 23/08/2007.
URL: *http://www.oed.com*

Papadopoulos, G. A. & Arbab, F. (1998), Coordination models and languages, Technical Report SEN-R9834, CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands.

Parents, M. & Gallais, G. (2002), Intelligent transportation in cities with CTS, *in* 'Proceedings of the Conference on Intelligent Transportation Systems (ITSC)', IEEE Computer Society, pp. 826–830.

Parker, L. E. (1994), Alliance: An architecture for fault tolerant cooperative control of heterogeneous mobile robots, *in* 'Proceedings of the 1994 IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS)', pp. 776–783.

Parker, L. E. (1998), 'Alliance: An architecture for fault-tolerant multi-robot cooperation', *IEEE Transactions on Robotics and Automation* **14**(2), 220–240.

Parker, L. E. (1999), 'Cooperative robotics for multi-target observation', *Intelligent Automation and Soft Computing* **5**(1), 5–19.

Parker, L. E. (2003), The effect of heterogeneity in teams of 100+ mobile robots, *in* A. C. Shultz, L. E. Parker & F. E. Schneider, eds, 'Proceedings of the NRL Workshop on Multi-Robot Systems', Kluwer Academic Publishers.

Paromtchik, I. E. & Laugier, C. (1996), Motion generation and control for parking an autonomous vehicle, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation', IEEE, pp. 3117–3122.

Parunak, H. V. D. (2003), Making swarming happen, *in* 'Proceedings of Conference on Swarming and Network Enabled Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR)'.

Ranjit, N. (2007), Multiagent referral systems: Maintaining and applying trust and expertise mode, Master's thesis, North Carolina State University.

Rekleitis, I. M., Dudek, G. & Milios, E. E. (2001), 'Multi-robot collaboration for robust exploration', *Annals of Mathematics and Artificial Intelligence* **31**(1-4), 7–40.

Schelfthout, K. (2006), Supporting coordination in mobile networks: a middleware approach, PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium.

Schelfthout, K. & Holvoet, T. (2005), Coordination middleware for decentralized applications in dynamic networks, *in* 'Proceedings of the 2nd international doctoral symposium on Middleware (DSM)', ACM Press, pp. 1–5.

Schelfthout, K., Weyns, D. & Holvoet, T. (2005), Middleware for protocol-based coordination in dynamic networks, *in* 'Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing (MPAC)', ACM Press, pp. 1–8.

Schelfthout, K., Weyns, D. & Holvoet, T. (2006), 'Middleware for protocol-based coordination in mobile applications', *IEEE Distributed Systems Online* **7**(8).

Schemmer, S. (2004), A Middleware for Cooperating Mobile Embedded Systems, PhD thesis, Fakultät fur Informatik , Otto-von-Guericke-Universität Magdeburg.

Schemmer, S. & Nett, E. (2003), Achieving reliable and timely task execution in mobile embedded applications, *in* 'Proceedings of the IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS)', IEEE Computer Society, pp. 61–68.

Schemmer, S., Nett, E. & Mock, M. (2001), Reliable real-time cooperation of mobile autonomous systems, *in* 'Proceedings of the Symposium on Reliable Distributed Systems (SRDS)', IEEE Computer Society, pp. 238–246.

Schermerhorn, P. & Scheutz, M. (2006), Social coordination without communication in multi-agent territory exploration tasks, *in* 'Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems (AAMAS)', ACM Press, pp. 654–661.

Schmidt, A., Aidoo, K. A., Takaluoma, A., Tuomela, U., Laerhoven, K. V. & de Velde, W. V. (1999), Advanced interaction in context, *in* H.-W. Gellersen, ed., 'Proceedings of the First International Symposium on Handheld and Ubiquitous Computing (HUC'99)', Vol. 1707 of *Lecture Notes in Computer Science*, Springer, pp. 89–101.

Schoonderwoerd, R., Bruten, J. L., Holland, O. E. & Rothkrantz, L. J. M. (1996), 'Ant-based load balancing in telecommunications networks', *Adaptive Behavior* **5**(2), 169–207.

Schraft, R. D. (1994), 'Mechatronics and robotics for service applications', *IEEE Robotics & Automation Magazine* **1**(4), 31–35.

Selic, B. (1998), Using UML for modeling complex real-time systems, *in* 'Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES '98)', Springer-Verlag, London, UK, pp. 250–260.

Senart, A., Bouroche, M. & Cahill, V. (2008), 'Modelling an emergency vehicle early-warning system using real-time feedback', *International Journal of Intelligent Information and Database Systems, Special issue on "Information Processing in Intelligent Vehicles and Road Applications"* **2**(1). To appear.

Senart, A., Cunningham, R., Bouroche, M., O'Connor, N., Reynolds, V. & Cahill, V. (2006), MoCoA: Customisable middleware for context-aware mobile applications, *in* 'Proceedings of the 8th International Symposium on Distributed Objects and Applications (DOA)', Vol. 4276 of *LNCS*, Springer Verlag, pp. 1722–1738.

Sheng, W., Yang, Q. & Guo, Y. (2006), Experimental testbed and distributed algorithm for cooperative driving in vii simulation, *in* 'Proceedings of the IEEE Conference on Intelligent Transportation Systems (ITSC)', IEEE, pp. 1627–1632.

Simmons, R. G., Singh, S., Hershberger, D., Ramos, J. & Smith, T. (2000), First results in the coordination of heterogeneous robots for large-scale assembly, *in* D. Rus & S. Singh, eds, 'Proceedings of Experimental Robotics VII (ISER '00)', Vol. 271 of *Lecture Notes in Control and Information Sciences*, Springer-Verlag, pp. 323–332.

Singh, K., Nedos, A. & Clarke, S. (2006), TransMAN: A group communication system for manets, *in* S. Chaudhuri, S. Das, H. Paul & S.Tirthapura, eds, 'Proceedings of the 8th International Conference on Distributed Computing and Networking (ICDCN)', Vol. 4308 of *Lecture Notes in Computer Science*, Springer, pp. 430–441.

Skeen, D. & Stonebraker, M. (1983), 'A formal model of crash recovery in a distributed system', *Transactions on Software Engineering (TSE)* **9**(3), 219–228.

Stankovic, J. A., Ramamritham, K., Niehaus, D., Humphrey, M. & Wallace, G. (1999), 'The Spring system: Integrated support for complex real-time systems', *The International Journal of Time-Critical Computing Systems* **16**, 223–251.

Ueki, J., Mori, J., Nakamura, Y., Horii, Y. & Okada, H. (2004), Development of vehicular-collision avoidance support system by inter-vehicle communications, *in* 'Proceedings of the Vehicular Technology Conference (VTC)', Vol. 5, IEEE, pp. 2940–2945.

Varaiya, P. (1993), 'Smart cars on smart roads: Problems of control', *IEEE Transactions on Automatic Control* **38**(2), 195–207.

Verissimo, P. & Almeida, C. (1995), 'Quasi-synchronism: a step away from the traditional fault-tolerant real-time system models', *Bulletin of the Technical Committee on Operating Systems and Application Environments (TCOS)* **7**(4), 35–39.

Verissimo, P., Cahill, V., Casimiro, A., Cheverst, K., Friday, A. & Kaiser, J. (2002), Cortex : Towards supporting autonomous and cooperating sentient entities, *in* 'Proceedings of European Wireless 2002', pp. 595–601. Invited paper, not peer reviewed.

Veríssimo, P. & Casimiro, A. (2002), 'The timely computing base model and architecture', *IEEE Transactions on Computers - Special Issue on Asynchronous Real-Time Systems* **51**(8), 916–930.

Veríssimo, P. & Casimiro, A. (2003), Event-driven support of real-time sentient objects, *in* 'Proceedings of the IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS)', IEEE Computer Science, pp. 2–9.

Veríssimo, P., Casimiro, A. & Fetzer, C. (2000), The timely computing base: Timely actions in the presence of uncertain timeliness, *in* 'Proceedings of the International Conference on Dependable Systems and Networks', IEEE Computer Society Press, New York City, USA, pp. 533–542.

Wang, K. & Li, B. (2002), Efficient and guaranteed service coverage in partitionable mobile ad-hoc networks, *in* 'Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE INFOCOM', IEEE, pp. 1089–1098.

Warren, C. W. (1990), Multiple robot path coordination using artificial potential fields, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation', IEEE, pp. 500–505.

Weigel, T., Gutmann, J.-S., Dietl, M., Kleiner, A. & Nebel, B. (2002), 'Cs freiburg: Coordinating robots for successful soccer playing', *IEEE Transactions on Robotics and Automation* **18**(5), 685–699.

Weyns, D. & Holvoet, T. (2008), 'Architectural design of a situated multiagent system for controlling automatic guided vehicles', *International Journal on Agent Oriented Software Engineering (IJAOSE), Special Issue on Multiagent Systems and Software Architecture* . to appear.

Weyns, D., Schelfthout, K. & Holvoet, T. (2005), Exploiting a virtual environment in a real-world application, *in* D. Weyns, H. V. Parunak & F. Michel, eds, 'Proceedings of the 2nd International Workshop on Environments for Multiagent Systems (E4MAS)', Vol. 3830 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 218–234.

Weyns, D., Schelfthout, K., Holvoet, T. & Lefever, T. (2005), Decentralized control of E'GV transportation systems, *in* M. Pechoucek, D. Steiner & S. G. Thompson, eds, 'Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)', ACM, pp. 67–74.

Weyns, D., Steegmans, E. & Holvoet, T. (2004), Towards commitments for situated agents, *in* 'Proceedings of the IEEE International Conference on Systems, Man & Cybernetics', Vol. 6, IEEE, pp. 5479–5485.

Yared, R., Cartigny, J., Defago, X. & Wiesmann, M. (2007), Locality-preserving distributed path reservation protocol for asynchronous cooperative mobile robots, *in* 'Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems (ISADS)', IEEE Computer Society, pp. 188–195.

Yolum, P. (2005), Towards design tools for protocol development, *in* F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh & M. Wooldridge, eds, '4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-05)', ACM, pp. 99–105.

Zhao, Y. (1997), *Vehicle Location and Navigation Systems*, Artech House Publishers.