A Systematic Approach to Safe Coordination of Dynamic Participants in Real-time Distributed Systems

Mong Leng Sin

21st December 2013

Thesis 10211

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.

Acknowledgements

A year after I have first submitted the thesis, and six months after I had my VIVA, I felt that this is the right time to write this page so as to give appreciation to everyone that help made the PhD journey possible (and enjoyable).

First and foremost, I would like to thank DSO National Laboratories and my colleges for believing in me and sponsoring the study program.

To my great supervisors, Prof. Vinny Cahill and Dr. Mélanie Bouroche, for patience in trying to understand my work when I could not explain them well, and for the guidance when I am too ambitious and tries to do everything.

To all the people in my research group, DSG, for taking the journey together; in particular, Zhang Shu for all the break time chit-chat that kept me sane, Mithileash, Dan and Jan for keeping the lab lively, Marco for all the serious discussions, and many others that made my time in Trinity College a worldly experience.

To Wenfu and Alex for being my guarantor to DSO; taking the risk of a half million dollars debt in the event that I do not return to Singapore.

To Jing for providing the most cozy home in Dublin and the opportunity to meet my wife. To Jing's daughters, Shannon and Celine for being so cute.

To my mother and elder brother for keeping everything intact in Singapore. To my sister, Ling Zhi, for the listening ear.

Last but not least, to my girlfriend turned wife, Zhang Kun, for our baby, JunYi, and all the colors throughout the PhD journey.

Besides all the thanks, I am sorry, Mong Xing, for not being there during your last few years, for not realizing that you was gone until this thesis is completed.

Abstract

Computer systems that employ autonomous robots have been demonstrated in many areas including entertainment (e.g., robot soccer), defense (e.g., reconnaissance) and homeland security (e.g., disaster rescue). To ensure successful operation, autonomous robots in these applications must coordinate their behaviors towards achieving the goals of the system while respecting safety requirements. In a safety critical system, violations of the safety requirements might endanger human safety and possibly damage crucial or expensive infrastructure, therefore, the system must be reliable despite having unreliable components (e.g., communication, sensors, or actuators). In addition, system scalability is difficult to achieve because an increase in robot numbers results in a disproportionate increase in the required resources such as processing power, memory and communication bandwidth. Furthermore, mobile robots may be allowed to move in and out of the operational area (we term these robots 'dynamic participants'), resulting in the additional complexity of managing a varying number of robots in the environment. Indeed scalability problems are often tackled by applications resorting to using a fixed number of robots or assuming some upper bounds on robot numbers. These challenges of reliability (building a reliable system over unreliable communication and hardware), and scalability (varying robot numbers) have only been addressed jointly for specific problems.

This thesis presents Comheolaíocht - a systematic approach to design coordination protocols for dynamic participants in real-time distributed systems allowing autonomous mobile robots to achieve their goals safely. In particular, the approach ensures that a system's safety constraints are respected in the event of robot breakdown and imperfect communications, and in the presence of dynamic participants, allowing entities to move in and out of the operational area. The thesis presents an approach to the design and development of multi-robot coordination protocols using three systematic steps. The first step, *system modeling*, captures the system specification by recording the entitys' parameters and behaviors, and any constraints that exist in the application. The step promotes reusablility by providing an abstraction for commonly seen constraints (e.g., the 'robots cannot perform some actions at the same time' constraint is modeled by a shared resource problem abstraction). These abstractions allow the comparison and use of established resource sharing approaches (i.e., scheduling or mutual exclusion). The second step, *system analysis*, analyses the system specifications captured in the first step to obtain two results. Firstly, it determines whether Comheolaíocht can provide a reliable solution to the multi-robot coordination system. Secondly, if a reliable solution exists, the analysis step outlines a coordination strategy that defines what entities can/cannot do in various situations so as to ensure safety in the system. These two results are obtained by analyzing variables and behaviors whose impact might violate the system constraints, and whether the robot has control over these variables and behaviors. In particular, a coordination strategy for a reliable solution exists if the robot can deterministically control its variables such that it can avoid violating the constraints during the periods which the robot needs to coordinate with other robots. The third step, *protocol derivation*, derives the system's coordination protocols by integrating the coordination strategy from step two with our coordination pattern (CwoRIS). CwoRIS guarantees exclusive access to resources despite imperfect communication and contributes to the reliability non-functional requirement. The derived protocols support scalability by taking advantage of locality in space and time; only the necessary information is sent to the relevant parties.

Comheolaíocht is evaluated on two fronts. Firstly, the thesis demonstrates that the resulting coordination protocols are reliable; it shows that the derived protocols prevent deadlocks and live-locks and ensure exclusive access to shared resources despite communication errors. Secondly, we demonstrate a practical application with a simulation of an intelligent transportation system in which autonomous vehicles coordinate by driving safely on a road. Results in the simulation reinforce the argument that the derived protocol is i) reliable as there are no collisions, and ii) scalable as there is an upper bound on required resources.

The thesis shows that Comheolaíocht can be used to develop protocols for multi-entity systems that are both reliable and scalable; and that are not supported by existing coordination methods.

Contents

Chapter 1

Introduction

This thesis presents Comheolaíocht - a systematic approach to design coordination protocols for dynamic participants in real-time distributed systems allowing autonomous mobile robots to achieve their goals safely. Using Comheolaíocht's three steps, developers can develop reliable distributed coordination protocols for multiple autonomous systems that are scalable with a varying number of mobile robots (entities). These developed protocols provide scalability by utilizing local communication and provisioning for entities' arriving and leaving the local operational area; this thesis terms such entities *'dynamic participants'*, the property '*dynamic participation*' (DP) describes a system with dynamic participants. These developed protocols also allow entities to reliably achieve their goals while ensuring system-wide safety constraints are met, despite having only unreliable communications, sensors and actuators.

This chapter first describes the motivations for this work. After this, the chapter defines the concept of coordination and gives an overview of existing related work. Section 1.3 provides a categorization of coordination problems and formally introduces the set of problems that this thesis addresses. Section 1.4 describes the challenges to be overcome and Section 1.5 describes the approach taken by this work. Section 1.6 presents the goals and contributions. Section 1.7 outlines the thesis scope, and issues that will not be covered. Finally, a road map of the thesis and a summary of the chapter are presented.

1.1 Motivation

Advances in robotics and mobile communication have enabled the development of autonomous mobile robot systems. Instead of using a single complex robot, the potential advantages of deploying multiple autonomous mobile robots are numerous [Dudek et al., 1996, Cao et al., 1995, Sotzing et al., 2007]. A multi-entity system's intrinsic parallelism provides robustness to single entity failures, and in many cases, can guarantee better time efficiency [Dudek et al., 1996]. In addition, some tasks may be inherently too complex (or impossible) for a single robot to accomplish [Cao et al., 1995].

As systems can either have centralized or distributed control. The issue of whether centralized or distributed methods are better is controversial. On one hand, centralized methods might suffer from single points of failure, create choke points when system load increases, and/or require special equipment (e.g., for the centralized controller). On the other hand, centralized methods are relatively simpler to implement and optimize, and provide a naturally centralized portal to track and control robots. In this thesis, a distributed controlled system is favored because the intended applications, multi-entity systems, are naturally distributed. Although there are multi-entity coordination solutions using a centralized approach (e.g., the intersection manager [Dresner, 2009]), such methods have the disadvantages outlined above.

Computer systems that employ multi-entity coordination have been demonstrated in many applications including intelligent transportation systems [Dresner and Stone, 2006a, Dao et al., 2008a, Qu et al., 2010], robotic soccer [Lau et al., 2009, Mota et al., 2011], multi-robot exploration [MacKenzie, 2003, Elizondo-Leal et al., 2008], and construction [Wawerla et al., 2002, Werfel and Nagpal, 2006]. On one hand, each of these applications must achieve their own functional requirements such as scoring goals in robotic soccer or completing a construction. On the other hand, non-functional requirements like reliability and scalability are just as important [Emmerich, 2000].

Most multi-entity systems have their own specification of reliability. For instance, autonomous underwater vehicles in a mine-countermeasure mission must find all the mine-like-objects despite vehicles' failures [Sariel et al., 2006a, Sotzing et al., 2007], and autonomous vehicles in an intelligent transportation system must not crash [Bouroche, 2007, Dresner, 2009]. Such multi-entity systems may have some *goals* that must eventually be satisfied or some *safety constraints* that must be satisfied at all times. The autonomous underwater vehicles discovering all mine-like objects is an example of a system's goal and vehicles not crashing is an example of a system's safety constraint. The combination of many different entities may result in a complex system where the entities' goals and safety constraints are interdependent. To ensure successful operation, these autonomous robots must coordinate their behaviors towards achieving the goals while respecting the safety requirements. A categorization of the set of coordination problem goals and constraints is provided in Section 1.3.

In addition to being reliable, a multi-entity system may be required to scale in the number of entities deployed. For instance, users may want to deploy more autonomous underwater vehicles to search for the mine-like-objects so as to complete the operation faster. In addition, some systems may require support for dynamic participation. For instance, users may want to introduce additional entities to an operational

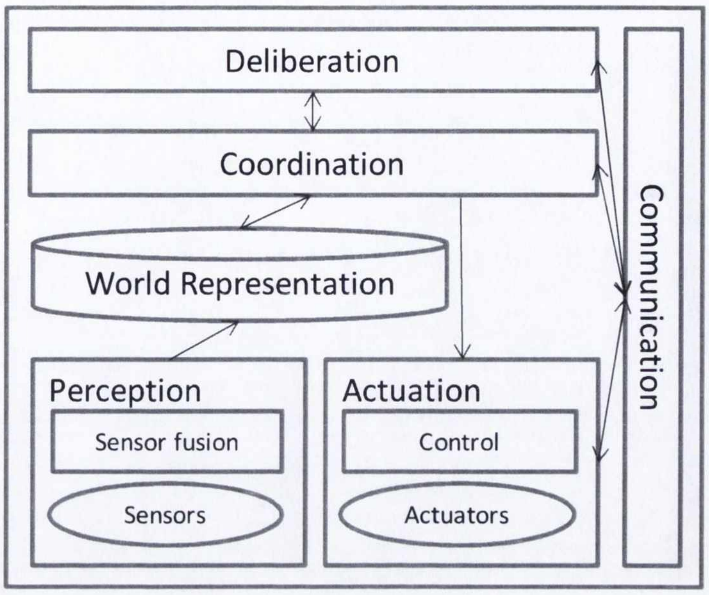


Figure 1.1: A typical robot architecture

system without restarting the system (thereby increasing efficiency), or failed robots can be removed for repair without stopping the whole system (thereby reducing system-down time).

1.2 Background

A multi-entity system is composed of many entities, where each entity itself can be considered a system. The first sub-section presents the different components within an entity and introduces the responsibilities of the coordination component. Section 1.2.2 then defines coordination in the context of this thesis. Finally, Section 1.2.3 provides an overview of existing work on coordination and shows that no existing generic protocols supports both reliability and dynamic participants.

1.2.1 Distributed multi-robot/agent systems

The typical components in a mobile entity are sensors and actuators, perception, control, world representation, deliberation, communication and coordination [Sotzing et al., 2007, Ren and Sorensen, 2008, Campbell et al., 2010]; a typical architecture with the common components for a robot is shown in Figure 1.1. A robot uses its sensors to read information, and fusion algorithms to combine the data into a common picture of the robot's situation; together, the sensors and fusion algorithms form the robot's perception system [Zhang et al., 2008, Broggi et al., 2008, Aragues et al., 2011]. Information from the perception system could be stored in a common format at a centralized place, or could be transferred directly to the appropriate modules. Information store at the centralized place forms the robot's world representation. Some work implements the world representation using a real-time database [Sotzing et al., 2007, Lau et al., 2009, Mota et al., 2011]. Others define a common data format (Papp et al. [2008] world model) or ontology concepts [Tsai et al., 2008, Hulsen et al., 2011] describing information in the world representation. A robot's actuators allow it to perform actions in the environment, while the robot's control component sends commands to the actuators to carry out the desired actions. The actuation component (control and actuators) is responsible for performing the actions accurately. A robot's deliberation component plans the robots actions in order to achieve its goals; examples of such plans are route planning [Jagadeesh and Srikanthan, 2008] and learning [Panait and Luke, 2005, Bazzan, 2009, Aragues et al., 2011]. Multi-robots systems usually include two more modules, communication and coordination. Communication can include many layers with each layer focusing on delivery capabilities like accessing physical communication devices, routing of messages and providing geo-casting capabilities. The robot's coordination layer lies in the middle (Section 1.2.2 formally defines the role of coordination). In summary, an informal description of each component is: the deliberation component tells a robot the tasks to be done, the perception component reads information from the environment and stores it in a common world representation, the coordination component takes the tasks from deliberation as input, reads the world representation of the current situation, uses the communication component to interact with other robots to find out when/whether each task may be performed, and tells the actuation component to perform the task at some particular time.

1.2.2 Coordination

There are many definitions of coordination [Omicini and Ossowski, 2003, Parunak et al., 2003]. In order to scope this work, this thesis adopts the definition by Bouroche [2007] which classifies coordination as:

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

and 'interaction' is defined as:

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

'Result' captures the goal of the system/entity.

1.2.3 Coordination protocols

While there are many real-time middleware systems (e.g., [Casimiro and Verissimo, 2002, Schemmer, 2004, Allouche and Daigle, 2006, Bouroche, 2007]) that support entity coordination, to the author's knowledge, there are no generic coordination protocols that support dynamic participants, ensure entity safety in the presence of imperfect communication and entity failure, and ensure that there are no deadlocks or live-locks.

Section 1.2.3.1 defines the properties of a reliable multi-entity system in the context of this work. Section 1.2.3.2 then defines real-time system and provides an overview of how Comheolaíocht specifies and implements the real-time requirements in a coordination system. Lastly, Section 1.2.3.3 describes the dynamic participants property.

1.2.3.1 Reliable multi-entity systems

A system's reliability can be defined in terms of safety and liveness properties. The *safety property* assets that nothing bad happens and the *liveness property* asserts that something good eventually happens [Kramer and McGee, 1999]. The ISIS project [Birman et al., 1984] is a high-level programming platform for reliable distributed systems. ISIS strategically inserts checkpoints for transparent replication, these replications allows the developed distributed system to achieve forward progress and fault tolerance [Birman et al., 1985].

A system's safety constraints implement the safety property, it describe conditions that must be satisfied at all times. If a safety constraint is not satisfied, the system is considered to have failed; in a safety critical system such failures may cost human lives [Kopetz, 2011]. Many multi-entity coordination protocols support maintaining a system's safety property despite communication failures [Casimiro and Verissimo, 2002, Schemmer, 2004, Bouroche et al., 2006, Dresner and Stone, 2006a, Veríssimo, 2006]. While most protocols allow safety constraints to be specified, to our knowledge, only Bouroche [2007] and Schemmer and Nett [2003b] provide methods to ensure mutual exclusion (i.e., that entities will not access the same shared resources at the same time) in a multi-entity system with dynamic participants and imperfect communications. Although Schemmer and Nett [2003b] provide scheduling of exclusive use of resources, it requires a centralized entity and assumes that there are at most omission degree (OD) number of consecutive incorrect messages. To our knowledge, support for entity failure has only been seen in specific applications [Sariel et al., 2006b, Yared et al., 2007, Dresner and Stone, 2008a]; no generic protocols ensure system safety properties despite entity failure.

The liveness property ensures that the system's and entities' goals are achieved. This implies that the system should not have deadlocks or live-locks [Tai, 1994, Ho et al., 2005, Padua, 2011]. Many mutual exclusion protocols have been shown to be free from deadlocks and live-locks [Ricart and Agrawala, 1981, Maekawa, 1985, Agrawal and Abbadi, 1991]. However, few multi-entity coordination protocols are evaluated against this requirement. To our knowledge, multi-entity coordination protocols that are proven deadlock and live-lock free only support specific applications (e.g., Yared et al. 2007) or do not support dynamic participants [Allouche and Daigle, 2006].

1.2.3.2 Real-time Programs

Real-time programs must guarantee response within strict time constraints, these constraints are often referred to as *deadlines* [Ben-Ari, 2006]. In Comheolaíocht, the development of protocols for the coordination of real-time distributed systems follows three steps:

1. System's constraints with respect to events happening sequence (time) are specified (See Chapter 3).
2. System's model is analysed to determine whether the situations where real-time decisions are required (See Chapter 4).
3. *Deadlines* are mapped into the physical space and exists in the form of *decision points*; an entity must make some decisions before arriving at the decision point. (See Chapter 5).

1.2.3.3 Dynamic participation

A system where entities may join or leave the system is sometimes called an open system [Bouroche, 2007]. However the term is also used to describe computer systems that provide some combination of interoperability, portability, and open software standards [Zimmermann, 1980, Garud and Kumaraswamy, 1993]. The term 'open system' is also used in areas such as physics [Von Bertalanffy et al., 1950], control theory [Pritschow et al., 1993] and organizational behavior [Pondy and Mitroff, 1979]. On the other hand, the terms dynamic participants and dynamic participation are seldom used, and the few uses also implies the property of entities entering and leaving the system. For instance, Chung et al. [1994] describe dynamic participation in a conference as "participant to dynamically join and leave a computer-based conference that is already in progress". Moreover, besides entities joining or leaving, allowing dynamic participants in this thesis admits a potentially *high-rate* of entities joining and leaving the system. Two properties of dynamic participation systems are:

Simultaneous arrival Due to many entities arriving and leaving, entities joining the system at the same time is common.

Absence without notice Entities may (want to) leave the system at any time; they may not wait for the system to 'register' their leaving before being absent. This implies that an entity cannot be depended upon to notify the system that it is leaving.

Most protocols support dynamic participants by requiring entities to go through a joining and leaving step [Murphy et al., 2001, Schemmer and Nett, 2003b, Bouroche, 2007]. The joining step is used to update new comers with information about the scenario; it is implemented either by freezing everyone else [Murphy et al., 2001], using a fixed entity to send the update information [Schemmer and Nett, 2003b] or waiting and listening for a certain period [Cunningham and Cahill, 2002]. The usage of a fixed entity implies some form of centralized decision making that faces the problems discussed earlier (Section 1.1), and freezing everyone sacrifices efficiency especially when there could be many entities entering and leaving the system; as such, this work adopts the wait and listen method.

Most middleware systems detect entity absence by using time-outs, such that entities that are not heard from for a period of time are removed from the system [Schemmer and Nett, 2003b, Cunningham and Cahill, 2002]. However, a silent entity may either have left the system or failed. Unlike an entity who left the system, a failed entity is still physically in the system's environment and may cause harm to other entities. For instance, a broken down autonomous vehicle may stop suddenly, and crash with another vehicle. To the author's knowledge, there are no protocols defining entity behaviors in preparation for and reaction to other entities failure. Comheolaíocht allows developers to specify an entity's behavior when it fails, and makes allowance and defines contingency behaviors for other entities when a failure is detected. Note: undetected failures are not in the scope of this work.

1.3 Coordination Problem Categorization

This section provides a categorization of coordination problems based on event ordering; an 'event' in this context can be either an entity performing some action, something happening in the environment or the passage of time. The first subsection presents a classification of event ordering constraints. Following this, a classification of event ordering at a system-level is presented. The categorization gives an overview of which problems can be solved by using Comheolaíocht and which other problems are not covered.

1.3.1 Events ordering constraints

This section provides a classification of coordination events based on their ordering constraints within a coordination problem. These ordering constraints will be used in Chapter 3 for modelling events occurance. Lamport [1978] presented the *happened before relation* for specifying time in distributed systems, the relation consists of three conditions between two events a and b:

1. If both a and b is in the same process and a comes before b then $a \to b$,

2. If a is the sending of a message of a process and b is the receipt of the same message by another process, then $a \to b$.

3. If $a \to b$ and $b \to c$ then $a \to c$.

Chapter 5 derives distributed coordination protocols using properties in the happened before relation [Lamport, 1978]. However, the constraints specified in Chapter 3 and the classification of coordination problems below refers to the physical time. Let $\mathbf{E}$ represents the set of events in a coordination problem, two events, $a, b \in \mathbf{E}$, can be subject to one of the following constraints with respect to the physical time:

- $\mathbf{bf}(a, b)$: a must happen before b.
- $\mathbf{st}(a, b)$: a must happen at the same time as b.
- $\mathbf{dt}(a, b)$: a must not happen at the same time as b.
- $\mathbf{nr}(a, b)$: a and b have no ordering requirements.

Based on these four constraints, this section provides a classification of coordination events.

1.3.1.1 Same-time events

Same-time events have the constraint $\mathbf{st}(a, b)$; the events, a and b, must happen at the same physical time. Entities who are required to perform events at the same time (e.g., multi-robot formation problems [Zavlanos and Pappas, 2007], rendezvous problem [Kingston et al., 2005, Munz et al., 2008] and flocking problem [Olfati-Saber, 2006, Ren and Sorensen, 2008, Munz et al., 2008, Xiong et al., 2010]), usually assume the consensus property [Fischer et al., 1985, Ren et al., 2005, Olfati-Saber et al., 2007, Chockler et al., 2008]. It has been shown that consensus cannot be solved deterministically in asynchronous distributed systems in the presence of failures [Fischer et al., 1985, Lynch, 1996]. Furthermore, Chockler et al. [2005] show that knowledge of the total number of nodes is required to solve consensus in a synchronous system even with the assumption of eventual collision free messages; this knowledge is not freely available in a system with dynamic participation.

The same-time event constraint is transitive and reflexive, i.e.,:

- Transitive: $\mathbf{st}(a, b) \wedge \mathbf{st}(b, c) \Rightarrow \mathbf{st}(a, c)$
- Reflexive: $\mathbf{st}(a, b) \Rightarrow \mathbf{st}(b, a)$

Comheolaíocht focuses on systems with dynamic participants and does not assume perfect communications. These properties make achieving consensus (which may be required by same-time events) difficult if not impossible. As such, this work focuses on problems with different time events (Section 1.3.1.2) and sequential events (Section 1.3.1.3), and does not support problems with same-time event constraints.

1.3.1.2 Different-time events

Different-time events have the constraint $\mathbf{dt}(a, b)$, the events, a and b, must not happen at the same physical time. We define $\mathbf{dt}(a, b)$ as 'a must not happen at the same time as b'. ($\neg\mathbf{st}(a, b)$ is ambiguous and is not used since it may also mean 'not (a must happen at the same time as b)', i.e., that there are no ordering constraints requiring that a and b must happen at the same time). An example of a different-time event problem is collision avoidance [Ferlis, 2002, Yared et al., 2007, Frese and Beyerer, 2011], where entities must not occupy the same space at the same time in the environment, otherwise the entities collide. Problems with different-time constraints can be abstracted as shared-resource problems, where the entities performing a and b require the same piece of shared resource. In this abstraction, entities cannot use the same shared resource at the same time (they must use it at different times). Exclusive access to shared resources can be ensured either by scheduling or by mutual exclusion. Scheduling involves allocating some time period for each entity to access the shared resource. Mutual exclusion protocols ensure that only one process (entity) can be within the critical section (using the shared-resource) at one time. Current centralized scheduling methods [Pinedo, 2008], usually require that the system has a clear idea about resources usage patterns whereas distributed scheduling methods [Liu et al., 2008, Zhou et al., 2010, Sharma et al., 2010] seldom provide guarantees to exclusive allocations during communication failures. Mutual exclusion use either token- [Chang et al., 1990, Naumann et al., 2002] or permission-based [Maekawa, 1985, Wu et al., 2008, Borran et al., 2008a, Attiya et al., 2010] methods. Dynamic participation makes the management of tokens more challenging, while limited knowledge of the number of nodes prevents consensus in permission-based approaches.

The different-time event constraint is reflexive but it is not transitive:

- Reflexive: $\mathbf{dt}(a, b) \Rightarrow \mathbf{dt}(b, a)$
- Not transitive: $\mathbf{dt}(a, b) \wedge \mathbf{dt}(b, c) \nRightarrow \mathbf{dt}(a, c)$

Comheolaíocht focuses on solving different-time events. As mentioned, dynamic participants and imperfect communication makes achieving consensus difficult if not impossible, therefore, protocols developed using the Comheolaíocht approach do not assume consensus. Instead, these protocols use a concept introduced by Bouroche [2007] whereby:

1. Entities are responsible for respecting the different-time event ordering constraints

2. An entity may sent a proposal to relevant entities announcing the entity's intention to perform some actions that may violates a contraint(s)

3. Every entity that receives such a proposal implicitly agrees to the proposal and gives way to the sender

4. The entity may perform its actions only after it is certain that every entity has given way to it

A protocol implementing this concept does not require consensus; only the sender knows the outcome of its proposal (whether everyone has receives the proposal), a receiving entity only knows that it must respect the constraints.

1.3.1.3 Sequential events

Sequential events have the constraint $\mathbf{bf}(a, b)$, i.e., event a must happen before event b. Most problems with sequential constraints implement special rules for specific situations, making such solutions not easily reusable. An example of a problem exhibiting sequence constraints is the construction problem [Wawerla et al., 2002, Werfel and Nagpal, 2006], where robots construct a building by placing objects in some order.

The sequential event constraint is transitive but not reflexive:

- Transitive: $\mathbf{bf}(a, b) \wedge \mathbf{bf}(b, c) \Rightarrow \mathbf{bf}(a, c)$

- Not reflexive: $\mathbf{bf}(a, b) \Rightarrow \neg\mathbf{bf}(b, a)$

Comheolaíocht supports problems with sequencing constraints using two tools: priority and preemption.

1.3.1.4 No time relation events

Two events may not have time constraints between them $\mathbf{nr}(a, b)$, however, this does not imply that the events are unrelated or that the problems are trivial. An example of a problem with no time relation is task allocation [Huang et al., 2008, Gerkey and Matarić, 2004]. The task allocation problem involves matching robots to tasks so that certain parameters are optimized (e.g., completion time).

The no time relation event constraint is reflexive but not transitive:

- Reflexive: $\mathbf{nr}(a, b) \Rightarrow \mathbf{nr}(b, a)$

- Not transitive: $\mathbf{nr}(a, b) \wedge \mathbf{nr}(b, c) \nRightarrow \mathbf{nr}(a, c)$

This work only focuses on reliability and scalability, it does not provide support for the optimization of solutions.

1.3.2 System event ordering

This section provides a characterization of coordination problems by the ordering constraints on the events produced. In particular, coordination problems are categorized as total-order, partial-order and no-order based on their requirements on event ordering.

In this section, let $\mathbf{C}$ represent the full set of ordering constraints of the problem; that is, $\mathbf{C}$ is the transitive closure and reflexive closure of the event ordering constraints. The table below is a summary on the reflexive and transitive relations on event ordering constraints.

	Same-time	Different-time	Sqeuential	No time relation
Transitive	yes	no	yes	no
Reflexive	yes	yes	no	yes

1.3.2.1 Total order

For $a, b \in \mathbf{E}$, let $\mathbf{bf}(a, b)$ and $\mathbf{st}(a, b)$ represent the constraint that 'a must happen before b' and 'a must happen at the same time as b' respectively. A coordination problem is a member of the 'total order' group if and only if

$$\forall a, b \in \mathbf{E}, a \neq b, \mathbf{C} \cap \{\mathbf{bf}(a, b), \mathbf{bf}(b, a), \mathbf{st}(a, b)\} \neq \emptyset$$

That is, coordination problems in the total-order category are composed of events that occur in a fixed sequence; every event has a constraint of either happening before, after or at the same time as another (different) event. An example of such a problem is for two robots (A, B) to pull a long pole out of the ground. In this scenario, each robot can perform one of the three actions of 'grab' (g), 'pull' (p) or 'release' (r). A fixed sequence to get the pole out of the ground is "g-A, p-A, g-B, r-A, p-B, g-A, r-B, p-A ..." Note that the sequence of actions that robots can take is fixed; such that there does not exist two events in the sequence that can be swapped. Coordination problems with a total ordering have a unique event sequence, therefore this sequence can usually be calculated in advance to save online processing.

1.3.2.2 Partial order

The partial-order category features events with a constrained ordering relative to some other events but does not require a fixed sequence of all events.

$$\neg \mathbf{t_{ord}}(\mathbf{E}) \wedge \exists a, b \in \mathbf{E}, a \neq b, \mathbf{C} \cap \{\mathbf{bf}(a, b), \mathbf{bf}(b, a), \mathbf{st}(a, b), \mathbf{dt}(a, b)\} \neq \emptyset$$

That is, some events in a partial-order coordination problem have a constraint of either happening 'before', 'after', 'at the same time' or a 'different time' as each other and these events do not have a fixed ordering. An example of a partial-order problem is coordinating vehicles in a road intersection [Naumann et al., 2002, Dresner, 2009]. At a road intersection, there are constraints such as 'vehicles in the same direction can move at the same time with the vehicle in front moving first' and 'vehicles with paths that intersect each other must not cross the intersection at the same time'. Although partial-order problems have fewer constraints than total-order problems and look simpler, the partial-order problems are seldom provided with a predefined operating sequence. Therefore coordination methods have to discover the event sequences at run time that maintain the ordering constraints.

Comheolaíocht focuses on solving partial-order problems. However, only support for the $\boldsymbol{bf}$ and $\boldsymbol{dt}$ constraints are implemented.

1.3.2.3 Non ordered

The category of non-ordered coordination problems does not have any sequencing constraints between the events.

$$\forall a, b \in \mathbf{E}, a \neq b, \mathbf{nr}(a, b)$$

These events can happen in any sequence and the problem is simpler with respect to event ordering. On the other hand, these problems are usually coupled with some other constraints. An example of this problem category is the exploration problem (Section 2.2.3), where a group of robots needs to search a fixed area optimizing some parameter such as minimal energy usage or fastest completion time. Robots in the exploration problem can search their assigned areas without ordering constraints.

1.3.3 Summary

In summary, Comheolaíocht supports the definition and implementation of different-time ordering and sequence ordering constraints. Both mutual exclusion and scheduling versions are supported for implementing different-time ordering constraints. Problems with sequence ordering constraints can be supported in Comheolaíocht using priority and preemption. As such, Comheolaíocht supports a sub-set of problems with partial-order events; the subset with same-time ordering constraints is not supported.

1.4 Challenges

This section describes the challenges facing a coordination protocol for dynamic participants in a distributed real-time system. The section groups these challenges into three categories: scalability, safety, and progress.

1.4.1 Scalability and limited resources

In some systems, there may be many entities. However, an increase in entity numbers may result in a disproportionate increase in the required resources like processing power, memory and bandwidth.

When the number of entities increases, the total available processing power and memory available increases proportionally (linearly) while the available network bandwidth does not change (constant). However, if each entity needs to perform some calculation or record information about all other entities, then the total required processing power and memory increase rate is polynomial to the number of entities. The required communication depends on the coordination protocol, but even when every entity only sends one message there is still a linear increase in communication usage in limited constant bandwidth. This disproportionate increase in required resources and availability creates a barrier to the ability for a system to scale in entity numbers. Section 2.1.4.1 presents a more detailed analysis of this non-uniform scaling problem. However, in order to support scalability, a coordination protocol must ensure that the rate of increase in required resources is slower than or at least at the same degree as that of the available resources (i.e., constant for bandwidth, and linear for memory and processing power usage).

One approach is to exploit the observation that entities in mobile environment are typically interested in events produced by other entities in the vicinity [Meier and Cahill, 2002] and limiting coordination to the local context. However, this results in dynamic participation (with respect to the local context), i.e., entities may arrive simultaneously and becomes absent without notice. Entities entering the system at the same time (simultaneous arrival) compete for resources in two dimensions: the environmental resource of space and the computational resource of bandwidth, memory and processing power. This implies that the protocol needs to ensure that physical entities do not collide in the physical space while they try to acquire the computational resources (e.g., bandwidth) required for coordination.

In order to be scalable, Comheolaíocht must ensure that usage growth is slower than or at the same degree as resource growth; to limit usage growth, protocols developed using Comheolaíocht only sends and calculates relevant information. The utilization of geographic relevance to preserve resources results in distributed computation and dynamic participants; protocols developed using Comheolaíocht must handle both simultaneous arrival and the absence without notice properties.

1.4.2 Reliability: safety and errors

Figure 1.1 shows the coordination component lying at the center of perception, actuation, deliberation and communication. This implies that the coordination component has to overcome the limitations of the components it is using (perception, actuation and communication) to support the deliberation component. The challenge is to support a deliberation layer with safety-critical requirements using components that might exhibit failures.

Comheolaíocht aids coordination protocols design that take into consideration of errors, the handled errors includes 1) entities' inaccurate localization and control, 2) entity breakdowns and 3) imperfect communications. Inaccuracies in the perception and actuation components (e.g., GPS, control) are handled by prediction algorithms [Dao et al., 2008b, Dao, 2008] and fusing information from multiple overlapping sensors [Broggi et al., 2008, Ferguson et al., 2008, Campbell et al., 2010]. Current research shows that such localization and control is not perfect. In Comheolaíocht, inaccuracy bounds (i.e., localization and other sensory inputs) may be specified. Because entities may breakdown due to disturbances from the environment or wear and tear, coordination protocols developed using Comheolaíocht must handle situations where entity (or components within an entity) breakdown, the method allows the specification of the entity's actions during a breakdown and a time upperbound for detection of the component breakdown. By treating an entity (or component) breakdown as actions, Comheolaíocht is able to reuse tools for the analysis of entity's behaviour; however, Comheolaíocht must therefore handle actions that may be invoked non-intentionally. In addition, mobile entities typically communicate over a wireless network where message collisions are hard to detect, causing unpredictable latency or missing messages; this implies that communication is highly unreliable [Gaertner and Cahill, 2004, Hughes et al., 2004].

Entities in a system usually have some objectives to satisfy (Section 2.1.2.4). Safety constraints specify a condition that must be true at all times. In particular, the violation of the safety constraint is unacceptable in safety critical systems as the failure may cost human lives (Section 2.1.2.2). Systems that provide safety guarantees even when sub-systems fail are called fault-tolerant systems. Fault-tolerant systems can be designed by incorporating additional components and algorithms which attempt to ensure that occurrences of erroneous states do not result in later system failures [Randell et al., 1978]. Dealing with faults usually involves three steps: detection, isolation and recovery [Li and Ramaswami, 1997].

One of the challenges in failure detection is that entities cannot distinguish between deficient communication and the absence of other entities [Bouroche, 2007]. Another challenge in a multi-entity system is that the failure of a single entity might propagate and cause other entities to fail; the coordination system must isolate the fault. For instance, Dresner and Stone [2008a] describe a situation where a vehicle failure in a cross junction may result in many vehicles colliding. Although their protocol for 'preventing catastrophic failures' lowers the number of chained collisions, it does not totally prevent chain-collisions, causing the implementation to be unsuitable in the real environment where human lives are at stake.

In order to be reliable, protocol developed using Comheolaíocht must ensure that the safety constraints are satisfied at all times despite inaccuracies in both the perception and actuation components. Delays and losses of messages in the communication component must also be considered. In order to prevent system failures, Comheolaíocht guides the allocation of allowances for detecting, isolating and recovery from faults.

1.4.3 Reliability: progress, live-locks and deadlocks

A system should be reliable not only in ensuring that the safety constraints are never violated, it should also allows the entities to progress toward their goals. The undesirable properties that may impede progress in a coordination protocol include deadlocks and live-locks [Tai, 1994, Ho et al., 2005]. The former refer to processes that block each other, thus preventing either from executing. The latter refer to processes that prevent each other from progressing, but do not actually block the execution. Another undesirable property is starvation, where some entity never gets a chance to progress; unlike a live-lock, other entities do progress.

In a multi-entity system, entities having physical bodies implies that the problem of deadlock, live-lock and starvation may occur both in the physical environment and the computational internal representations. For example a simple deadlock can happen between two vehicles, x and y, on a road. If vehicle x is physically in-front of vehicle y, vehicle y cannot move until x moves. If a protocol does not consider the physical environment and allocates y to move first (x waits for y to move), this results in a deadlock where both x and y wait for each other.

In essence, a reliable protocol must not only ensures that the safety constraints will never be violated, it must also allow the entities to eventually achieve their goals, such a protocol must not has a deadlock or live-lock.

1.5 Approach

This thesis presents an approach for designing and developing a multi-robot coordination system using three systematic steps (an overview of the steps is shown in figure 1.2).

The first step, *system modeling*, captures the system specification by recording the entity's parameters and behaviors, and any constraints that exist in the application. This step promotes reusablility by

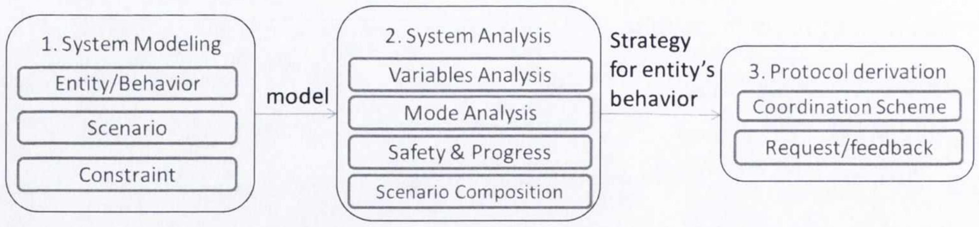


Figure 1.2: Overview of Comheolaíocht's 3-steps

providing an abstraction for commonly seen constraints. In particular, different-time event ordering constraints that describe situations where entities cannot perform some events at the same time are modeled as shared resource problems. These abstractions promote reusability and allow the comparison and use of established resource sharing approaches (i.e., scheduling and mutual exclusion).

The second step, *System analysis*, analyses the system specifications captured in the first step to obtain two results. Firstly, it determines whether Comheolaíocht can provide a reliable solution to the multi-entity coordination system. The reliable solution needs to i) ensure that entities can be safe (respect all constraints) in spite of the errors demonstrated by the perception (e.g., GPS inaccuracies) and actuation (e.g., entity breakdowns) components, and ii) ensure that entities can achieve progress (satisfies their goal). Secondly, if a reliable solution exists, the analysis step outlines a coordination strategy that defines what entities can/cannot do in various situations so as to ensure safety in the system. These two results are obtained by analyzing variables and behaviors whose impact might violate the system constraints, and whether the entity has control over these variables and behaviors. In particular, a coordination strategy for a reliable solution exists if the entity can deterministically control its variables such that it can avoid violating the constraints during the periods when the entity needs to coordinate with other entities.

The third step, *protocol derivation*, derives the system's coordination protocols by integrating the coordination strategy from step two with our coordination pattern - Contract without Response with Inference and Score (CwoRIS, [Sin et al., 2011]). The CwoRIS pattern is designed to operate on top of a (1) geocast protocol that provides (2) ordered delivery of messages, (3) bounded message latency and (4) real-time feedback, Section 5.1 describe the assumed underlying communication protocol in detail. CwoRIS guarantees exclusive access to resources despite imperfect communication and contributes to the reliability non-functional requirement. The derived protocols support scalability by taking advantage of locality in space and time; only the necessary information is sent to the interested parties.

1.6 Goal and Contributions

As motivated by Section 1.1, the goal of this thesis is to develop an approach to the design of coordination protocols for dynamic participants in real-time distributed systems allowing the entities achieve their goals while respecting the safety constraints. Since this work is built with Bouroche's thesis [Bouroche, 2007], this section highlights only the contributions and goals added to this baseline. The contributions of this thesis are fivefold:

1. A taxonomy on coordination problems classifying what constitutes 'hard' coordination problems; the taxonomy defines which properties make a coordination problem more difficult to solve.
2. The application of scheduling concepts to constraint satisfaction for distributed coordination.
3. A coordination pattern, CwoRIS, that provides distributed scheduling and mutual exclusion for dynamic participants ensuring exclusive access to resources despite imperfect communication.
4. A systematic process that allows developers to derive a protocol for a multi-entity coordination problem focusing on reliability (not violating the safety constraints and eventually satisfying the goals) and scalability (support for dynamic participants and local coordination).
5. A distributed coordination protocol for autonomous vehicles crossing an intersection developed using Comheolaíocht.

1.7 Scope

This thesis defines Comheolaíocht's three step process to design protocols for multi-entity coordination. The work also describes CwoRIS, a coordination pattern that ensures entities' exclusive access to shared resources.

This thesis focuses on the coordination component of a multi-entity system which helps entities to decide whether to perform some tasks and when to perform them. It is assumed that the higher deliberation layer that tells an entity what tasks to perform are implemented. The thesis also assumes that there are supporting lower layers providing perception, actuation, world representation and communication (Section 1.2.1). In addition, this work focuses on problems involving different-time and sequential ordering constraints (Section 1.3.1).

Entities in the system are assumed to be physical, mobile, utilize imperfect wireless communications and the system is assumed to be safety critical. In order to support such a system, the derived protocols must allow entities to act/react while ensuring that its constraints are not violated even when communication is lossy. In addition, protocols developed using Comheolaíocht do not rely on any centralized component thereby avoiding the problem of a centralized point of failure.

Comheolaíocht's support for dynamic participants with local coordination addresses the scalability requirement. Support for the reliability requirement is addressed by ensuring that the specified safety constraints are not violated, and that entities using derived protocols will not be involved in deadlocks or live-locks. Protocols derived using Comheolaíocht is assumed to run on a real-time operating system with local timers (e.g., Real-time Linux) and use a communication protocol that supports (1) geo-casting, (2) ordered delivery, (3) bounded message latency and (4) real-time feedback (e.g., Hughes [2006], Slot et al. [2010]; See Chapter 5.1 for a detail description of these requirements).

Comheolaíocht's steps are able to identify entity coordination that cannot be supported, and the reasons for which no support is possible are made known to the developer. However, it is up to the developer to (re)design an entity that can be implemented.

The definition and use of Comheolaíocht is illustrated with a number of examples from the intelligent transportation system domain (ITS). ITS, however, is not the subject of this thesis. The intelligent transportation system example is implemented using the C++ programming language and operates using the Player/Stage simulator [Gerkey et al., 2003] on top of the Real-time Linux (See Chapter 6.4).

This work assumes that entities comply with the developed protocols. Note that issues of trust and security are not considered in this thesis.

1.8 Road-map

The remainder of this thesis is organized as follows. Chapter 2 reviews the state of the art in multi-entity coordination. The next three chapters present the Comheolaíoch systematic steps for deriving a coordination system. Chapter 3 presents the steps for modeling the multi-entity system. Chapter 4 presents the steps for analyzing the system model with regard to safety and progress. Chapter 5 presents the derivation of protocols and CwoRIS - our real-time distributed coordination pattern. Chapter 6 proves that developed protocols are scalable and reliable. This evaluation also presents simulation results to reinforce the scalability and reliability proves. Finally, Chapter 7 concludes this thesis and outlines possible future work.

1.9 Summary

This chapter outlines the goals and scope of this thesis. In particular, the chapter presented a categorization of coordination problems and described the thesis focus on the set of coordination problems with different-time constraints. The chapter began by presenting the main motivation of this thesis; that distributed multi-robot coordination should be scalable and reliable. The main challenges for providing a scalable, and reliable system that ensures progress of the entities are presented. The chapter also provides a brief overview of existing work and showed that existing work on multi-entity coordination does not tackle dynamic participation and ensure safety when the components on which it relies are imperfect. The approach of this work is then described along with its goal and contributions.

Chapter 2

Related work

This thesis deals with the coordination of dynamic participants in safety critical real-time distributed systems. As explained in the introductory chapter, the problem that this thesis addresses is the design of a systematic approach for the development of coordination protocols for autonomous mobile entities. Two requirements of this work are reliability and scalability, Figure 2.1 shows a summary of the properties to be evaluated in this work (i.e., both in this related work chapter, and the evaluation chapter). In order for a system to be reliable, its safety constraints must be satisfied at all times and its goals must eventually be satisfied. This must hold even despite potential communication and entity faults. Mobile entities may move in and out of the operation area, providing a system with dynamic participants where entities may arrive simultaneously or leave without notification, therefore, a scalable protocol must be able to handle changes in entity numbers during operation.

Coordination of mobile autonomous entities has been studied mainly by researchers from the robotics, control and computer science fields. This chapter reviews existing work on coordination from the different communities focusing on the computer science domain.

To our knowledge, no recent work has been done on analyzing complexity parameters in multi-entity coordination, Section 2.1 provides such an analysis by characterizing complex coordination problems, and describes the difficulties in developing coordination protocols for scalable and reliable multi-entity systems. After this, Section 2.2 reviews multi-robot applications focusing on application specific multi-robot coordination protocols. The section shows that while many applications require the set of constraints supported by Comheolaíocht, only specific protocols for multi-robot application supports both scalability and reliability.

Section 2.3 surveys multi-entity systems based on their organizational structures [Horling and Lesser, 2004] and evaluates how these organizations contributes to the scalability of the system. Section 2.4

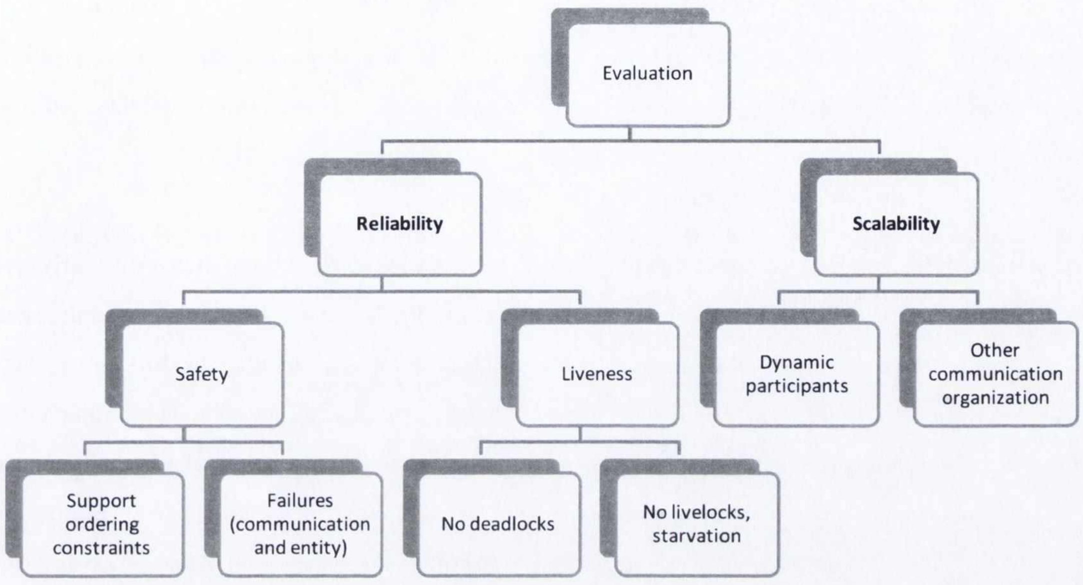


Figure 2.1: Evaluation properties

then reviews existing coordination protocols supporting event ordering safety constraints, in particular, the section focuses on the different-time and sequence constraints that this thesis addresses. Section 2.5 reviews protocols that support the coordination of mobile entities in real-time. Finally, Section 2.6 provides a comparison of Comheolaíocht with other protocols and shows that there are no generic protocols that supports the same set of properties.

2.1 A Taxonomy of Coordination Problems

This section presents a taxonomy of multi-entity coordination problems. This taxonomy differentiates parameters that make a coordination problem more complex; complex coordination problems have more difficulties due to various factors (e.g., unknowns, failures, computation speed) in achieving its required properties (i.e., safety and goal constraints). While there are many other taxonomies of multi-entity coordination, not many of them characterise complex problems. Our taxonomy serves three purposes: firstly, it provides a common definition of the terms used in this work. Secondly, the taxonomy helps a developer to better understand and categorize a coordination problem; such an understanding allows a developer to consider the different characteristics of his problem, and thirdly, ideas from work provides solutions addressing these parameters are highlighted; such ideas may be adopted by a developer.

The first subsection presents related work on taxonomies of entities coordination. The following section presents our coordination taxonomy and discusses the parameters that make a problem complex. Section 2.1.4 summarizes the coordination challenges under two categories: scalability and reliability.

2.1.1 Previous Taxonomies

There are many surveys and taxonomies of multi-robot systems. The most notable early work is from Dudek et al. [1996] and Cao et al. [1995]. Dudek et al. [1996] provide a taxonomy of multi-agent robotics focusing on the agents' communication capabilities (range, topology and bandwidth), processing capability and reconfigurability. Cao et al. [1995] surveyed cooperative robotics focusing on group architecture: centralized/decentralized, robot differentiation, communication structure and whether a robot model other robots' behavior. The authors listed research efforts in addressing resource conflicts, learning and geometric problems. These early papers contributed to shaping current research. However, being early papers, the challenges identified are either solved or are currently being investigated by various groups.

A more recent paper by Farinelli et al. [2004] classifies coordination problems according to whether the agents are aware of each other, whether the agents follow a set of pre-defined rules, and whether the agents follow some leader or think for themselves. The paper further classifies coordination systems by the way in which the agents communicate, the team composition, whether they are reactive or deliberative, and the team size. Parker [2008] presented three axes of distributed intelligence in multi-robot systems: goals, awareness of others and whether actions advance the goals of others. Parker uses combination values from these three axes to define various research areas like collective, cooperative, collaborative and coordinative systems. The paper further classifies approaches to distributed intelligence in mobile robotics as bio-inspired, organizational and social, and knowledge-based systems. The author differentiates between the various multi-agent coordination terminologies using these three axes and provides an overview of the research being done. However, none of this work highlights what makes a coordination problem difficult.

Grabowski and Christiansen [2005] measure the complexity in the ability of a robot to share information, coordinate actions, or convey intentions using their simplified taxonomy. They position robot coordination along a single spectrum according to the degree of cooperation of the robots. The lower end of the spectrum includes robot systems that rely on an external medium to coordinate actions and manage conflict. The middle of the spectrum includes systems that coordinate indirectly through a third party or shared resource (e.g., centralized controller). The upper end of the spectrum covers robot teams that model their environment and neighbors to predict the effects of interaction. While the single spectrum is simple to understand, the authors' focus is on the complexity of robot coordination solutions (how the entities coordinate), and not on the complexity of the underlying coordination problems.

Gerkey and Matarić [2004] provide a formal analysis and taxonomy of task allocation. Their analysis characterizes task allocation by robot-task capability (whether the robot is capable of handling more than a single task), task-robot requirement (whether a task requires a single or multiple robots to complete) and assignment period (instantaneous or time-extended). For each combination of characteristics, their analysis provides the computational and communication complexity. An equally authoritative survey by Dias et al. [2006] evaluates various algorithms according to their planning (sub-divided into task allocation, decomposition and execution), solution quality (in terms of their optimality guarantees) and scalability characteristics. Both papers provide an excellent description of the scaling complexities with regard to computation and communication. However they focus only on task allocation and not on general coordination problems.

2.1.2 Taxonomy

Based on the definition of coordination (Section 1.2.2) as "the management of interactions both amongst entities, and between entities and their environment, towards the production of a result", five axes describing a coordination problem are identified. The first two axes, *entity* and *environment*, are directly mentioned in the definition. The third axis specifies the *relationship* between entities, which forms the basis for entity interaction in the definition. The fourth axis, *objective*, captures the definition's 'production of a result' and the fifth axis, *communication protocol*, addresses the 'management of interactions'. In this section, unless otherwise specified, the complexity comparison assumes that all other properties are kept constant.

In the classification below, properties supported by Comheolaíocht are underlined.

2.1.2.1 Entity

Entities are the main actors in a coordination problem. This section describes parameters of an entity in a coordination problem.

- **<u>Physical</u>/virtual**

  Entities can either be physical (have a body) or virtual (exist only in a computer world).

  Coordination amongst physical entities usually deals with more complex environment parameters (Section 2.1.2.2) while virtual entities coordination usually focuses more on complex relationship parameters (Section 2.1.2.3).

- **Participation**

  The participation axis captures the number of entities in the system.

  Constant Under constant participation, entities do not leave the system and no new entity enters the system. These systems usually assume that every entity knows the total number of entities. Since knowledge of the total number of entities is required to arrive at consensus in a distributed synchronous system [Chockler et al., 2005], coordination problems with constant participation problems are less complex.

  Dynamic Under dynamic participation, entities may leave the system and new entities may join the system. Some ways of handling dynamic participants include

  - Enforce constant

    One approach is to disallow the number of entities to change when a coordination protocol is in progress. Attiya et al. [2010] enforce a constant number of entities by using a set of 'doors' to ensure that the current set of nodes participating is fixed before performing their mutual exclusion protocol.

  - Membership

    Membership has been used for recording the population of entities within the system, Meier and Cahill [2002] and Schemmer and Nett [2003b] provide membership information by requiring that every entity registers/announces their joining/leaving. Slot et al. [2010] use sensors and communication to calculate the set of entities that might be within an area.

  - Upper/lower-bounds

    Another approach is to specifying upper/lower bounds on entity numbers performing coordination, for instance, Borran et al. [2008a] assumes that the number of nodes is between an upper bound, n, and a lower bound, $\frac{n}{2}$, for their quorum based LastVoting algorithm.

  Infinite/unknown Under infinite/unknown participation, the number of entities are either uncountable or not known to the other entities. An example application area where there are infinite number of entities are swarm-based systems. Swarm systems [Bonabeau et al., 1999, Peleg, 2005, Cornejo et al., 2009] usually do not require knowledge of the number of entities to coordinate; the systems could coordinate by manipulating the environment (e.g., Ioannidis et al. [2011]).

  In Comheolaíocht, the whole system may have an infinite/unknown number of participants, however, the system is modeled as having dynamic participants local to some area.

- **Differentiation**

  Entity differentiation captures whether there are more than one type of entity in the system [Cao et al., 1995, Farinelli et al., 2004, Grabowski and Christiansen, 2005]; whether a system has homogeneous or heterogeneous entities.

  Homogeneous Entities with the same *behavior* and physical characteristics.

  Heterogeneous Entities with different behaviors and physical characteristics.

  Heterogeneous entities are generally more complex because the many entity-types might imply that each has a different objective (Section 2.1.2.4).

- **Knowledge**

  The knowledge axis records what an entity knows. The meta-data includes:

  Subject Who or what the knowledge is about. Three values are identified: *self*, *other* entities and *environment*. As its name implies, knowledge of self describes what an entity knows about itself. Knowledge of other entities describes what the entity knows about entities: such knowledge could either be pre-programmed, obtained from sensors or through communication. An entity's knowledge of the environment describes what it knows about the environment.

  State/Model State describes the current situation of the subject; for example, an entity's self state may capture its current position and speed. Model describes knowledge about how an entity acts/reacts. For example, a RoboCup [Kitano et al., 1997] entity's knowledge of the environment's model may include the soccer ball's trajectory (reaction) when it is kicked.

  Quality Defines whether the knowledge is complete and accurate. A piece of knowledge is complete if all information that is required on a subject's state/model is available. The knowledge is accurate if there is no discrepancy between the actual value and the perceived information.

  Additional knowledge usually results in a problem being less complex. In fact, a system where every entity is an oracle and knows everything would just need to calculate everyone's behavior (assuming deterministic behavior) in order to coordinate (e.g., [Schermerhorn and Scheutz, 2006]).

2.1.2.2 Environment

The environment axis captures parameters about the system in which the entities operates.

- **Failure**

  A *failure* is an event that denotes a deviation between the actual service and the specified or intended service, occurring at a particular point in real time [Kopetz, 2011].

  Perception Failures can be *consistent* or *inconsistent* [Kopetz, 2011]. In a consistent failure scenario, all entities see the same (possibly wrong) results. In an inconsistent failure situation, different users may perceive different false results.

  Effect Depending on the effect a failure has on the environment, failures can be classified as either *benign*, or *safety-critical*. Benign failure classifies systems where the advantage of the system outweighs the failure cost. Safety-critical failure classifies systems where failure may cause failure cost that are orders of magnitude higher than the normal utility of a system (e.g, where system failure may result in the loss of human lives).

- **Static/Dynamic**

  An environment may be static or dynamic [Russell and Norvig, 2010]. A static environment remains unchanged between entities' actions while a dynamic environment may change while an entity is deliberating its next move. Dealing with a dynamic environment is more complex.

- **Discrete/Continuous**

  An environment could be discrete or continuous [Russell and Norvig, 2010]; for example, a light-switch has the discrete values of 'on' and 'off', while the speed of a vehicle has continuous values. The continuous environment is more complex; a common method is to discretize a continuous environment. For instance Dresner [2009] represents an intersection (a continuous real-world location) as a set of grids.

2.1.2.3 Relationship

The relationship axis captures how an entity acts when towards other entities in the system.

- **Honest/dishonest**

  A dishonest entity could lie in order to gain an advantage in an interaction [Fullam et al., 2005]. Systems with lying entities are more complex.

- **Selfish/selfless**

  An entity can either be selfish or selfless. A selfish entity only performs actions to forward its own goal. A selfless entity may act to advance a group or global goal even at the expense of its own goal. Selfish entities are usually less complex since they only need to consider their own goal, however, selfish entities could be dishonest making a coordination problem more complex.

- **Rational/Irrational**

  A rational entity will always act to optimize either its own, a group or global goal [Russell and Norvig, 2010]. A rational entity deterministically selects the best action based on some (internal and/or external) state. In contrast, an irrational entity may perform behaviors without any particular purpose (e.g., random). Coordination of rational entities in a multi-entity system is less complex because an entity can predict another entity's behavior.

  Note: Entities with learning may perform some random actions, however, these entities perform these actions with a purpose (i.e., to learn), therefore, they are considered as rational.

  Note: Although a rational entity deterministically take actions based on some state, such actions may have non-deterministic results; such non-deterministic results can be recorded in an entity's knowledge of the model (Section 2.1.2.1).

- **Team/Coalition/None**

  Entities banding together in a group can either form a team or coalition [Horling and Lesser, 2004]. Coalitions are short-term groups that happen when entities with similar goals decide to act together in order to better achieve their individual goals. Members in a coalition often leave once their goals are achieved and their obligations paid. Teams are more persistent; a team is usually created by design. Entities within a team either have common goals or share a global goal. An example of a coalition is vehicle platooning [Stankovic et al., 2000] and a team example might be entities in the RoboCup competition [Kitano et al., 1997]. Coalitions are more complex than teams due to membership management issues such as managing entities joining and leaving. Comheolaíocht does not support groups.

- **Aware/Not aware of others**

  An entity may either be aware_or not aware of the presence of other entities. Awareness in this context refers to entities who reason about other entities' actions and intentions. Entities who are not aware may sense other entities' and react to them, but may otherwise perform no reasoning to understand their intent or future plans [Parker, 2008].

In order to limit the scope of this work, only simpler parameters (honest, selfish and rational entities that do not participate in groups) are supported in this work.

2.1.2.4 Objective

The objective axis of a coordination problem describes what an entity (or the coordination system) is attempting to achieve.

- **Safety constraint**

  The safety constraint axis in a coordination objective captures conditions that must always be true.

- **Goal**

  The goal objective describes what conditions the entity/system wants to make true.

- **Optimization**

  The objective of some coordination systems is to achieve a goal with some parameters maximized or minimized. This work does not provide support for optimization objectives.

2.1.2.5 Communication protocol

The communication protocol axis describes how entities pass information.

- **Direct/indirect**

  - Direct

    Direct communication (as opposed to indirect communication) is based on message passing.

  - Indirect

    Indirect communication is performed through the environment (e.g., by being present or depositing pheromones).

    Indirect communication has the potential to communicate with some unknown entities that are not currently present. However, communication through the environment can be limited by the sensitivity of sensors and actuators. Indirect communication is usually designed in a way that is specific to an application and is difficult to reuse.

- **Semantic**

  The semantic of a communication protocol captures the messages' meaning. Such meaning can generally be grouped under three categories:

  - Command

    In this case, the message content involves telling another entity what to do. When an entity sends a command to another entity, it expects the other entity to carry out its actions based on the message contents. Commands are usually seen in centralized systems where there is a central leader making decisions for other entities.

  - Request/negotiation

    In request or negotiation message, an entity asks/proposes to another entity (or group of entities) if it can do something in the future. In most cases, the receiving entity has the power to reject a proposal. In a negotiation protocol, a rejected proposal may be followed by a counter proposal.

  - Status report

    In a status report, an entity tells other entities about its current situation.

  Of the three forms, requests are the most complex for entity coordination because of the multiple rounds of messaging and ambiguity on whether a request is accepted. Status reporting is complex when entities do not know what other entities may do (in the future) based on the current status report; for example, in a system with irrational entities.

2.1.3 Supported multi-entity systems

To summarize, the requirements on systems supported by Comheolaíocht for reliable and scalable coordination are the following (each criteria is listed followed by a list of possible values - the required value is in bold):

Entity

- **Physical**/virtual
- Participation: constant/**dynamic**/(infinite/unknown)
- Differentiation: homogeneous/**heterogeneous**

Environment

- Failure
  - Perception: **consistent**/inconsistent
  - Effect: benign/**safety-critical**
- Static/**dynamic**
- Discrete/**continuous**

Relationship

- **Honest**/dishonest
- **Selfish**/selfless
- **Rational**/not-rational
- Group: **none**/team/coalition
- Relation with other entities
  - Awareness of others: **aware**/not aware
  - Intention towards others: good/**neutral**/bad
  - Type of goals: **individual**/shared.

Objective

- **Safety constraint**/**goal**/optimization

Communication protocol

- **Direct**/indirect
- Semantic: command/**request**/negotiation/status report

Comheolaíocht supports the more complex parameters from the entity, environment (except inconsistent failures), objective (except optimizations) and communication protocol axis. Inconsistent failures are only partially supported with communication and entity failures. The more complex parameters in the ralationship axis and optimizations are not supported in Comheolaíocht because although these parameters affects coordination, these issues are usually addressed in the deliberation higher-layer.

2.1.4 Complex multi-entity coordination

The taxonomy above considers the complexity of each property in isolation, this section provides an analysis of the complexity when these properties are considered together. In particular, the section focuses on the complexity of achieving a scalable and reliable system.

2.1.4.1 Scalability: participation, limited resources and organization

Many multi-entity solutions work on a limited number of robots. A reason for this may be the lack of funds coupled with the logistics involved in operating many robots. However, an unexplored issue is the scalability of many of those solutions.

Multi-entity systems have limited computational power, memory and communication bandwidth. Increasing the number of entities in a system increases the total available computational power and memory. Assuming that there are n robots in the system, then the system has n times the total amount of memory and computation power than a system with only one robot. Essentially, total processing power and memory increases linearly with the increase in entity numbers - $O(n)$. However, bandwidth does not grow with number of entities - $O(1)$. With the exception of entities in embarrassingly parallel systems [Foster, 1995], coordinating entities need to exchange information (requiring bandwidth), remember these information (requires memory), and perform some calculations (requires computation power).

Using a brute force method where every entity communicates with every other entity, point-to-point communication requires n^2 messages and broadcast communication requires n messages. If every entity remembers some facts about every other entity, the memory required per agent increases linearly with the number of entities - $O(n^2)$ in the whole system. Compared to the resource growth, linear in computation power, linear in memory and constant in bandwidth, the growth of resource usage is much faster. With this brute force method, the bottleneck to scaling up a multi-entity system is the limitation in bandwidth.

Due to the limited bandwidth, many coordination methods organize agents into teams, hierarchies or federations [Horling and Lesser, 2004]. Such organization controls the communication that an entity makes with other entities. For example, by limiting communication to a graph-based structure (e.g., hierarchical) the number of communicating pairs decreases to $n - 1$, i.e., linear in the number of agents. Dividing the agents into teams and controlling the maximum number of entities in each team reduces intra-team communication to a 'constant'. Inter-team communications, represented by the team's spokesperson, can be built using other organizational structures.

With bandwidth out of the way, the remaining bottlenecks are computation and memory. Organization structures also include methods to distribute (and reduce) calculations. An example is the hierarchical model; each leaf in the tree sends information to its parent while the parents fuse the received information before forwarding it to the grandparents. With information fusion, all nodes only process data from their immediate children and do not need to consider values from other nodes. The hierarchical organization distributes calculation such that each node performs a constant amount of calculation; the tree structure reduces the overall computation requirement to linear growth. Therefore, the hierarchical fusion method is scalable with regards to processing requirement and agent population growth. However the hierarchical fusion method pays the price of reduced accuracy; information from the individual nodes is lost during the fusion process. Auctions serve as a second example of the distribution of computations. A simple auction mechanism requires every agent to calculate its own utility and send its bid to a central auctioneer. The auctioneer then selects the best valued bid and announces the results. This market mechanism is known to allow heterogeneous entities to coordinate [Dias et al., 2006, Zlot and Stentz, 2006]. Market methods distribute the complex utility calculation, while the central auctioneer only deals with numbers. However, clearance (i.e., deciding the winner) for a simple auction has a complexity of $O(n)$ at the auctioneer and the complexity for auction clearance increases with more complex combinatorial auctions [Dias et al., 2006].

The differences in the rate of growth between resource availability and requirement is one of the main factors driving researchers to search for better organizational methods. Meier and Cahill [2002] observe that entities in the mobile environment are typically interested in events produced by other entities in their vicinity. Since embodied entities take up physical space, there is a natural upper bound on the maximum number of entities that could possibly be in an area. However, the use of geography to create a resource-usage upper-bound among mobile entities gives rise to an environment with dynamic participants. Combining these observations, this work provides scalability by focusing on supporting dynamic participants and coordination with entities within geographic relevance.

2.1.4.2 Reliability: objectives, failure cost and errors

Entities in a system usually have some objectives to satisfy (Section 2.1.2.4). The safety constraint specifies a condition that must be true at all times and violation of this constraint implies that the system has failed. System failure in a safety-critical system is unacceptable as it may cost human lives (Section 2.1.2.2). In addition, components, entities and communication are imperfect. Traditionally, fault-tolerant systems can be designed by incorporating additional components and algorithms that attempt to ensure that the occurrences of erroneous states does not result in later system failures [Randell et al., 1978]. Dealing with faults usually involves two steps: detection and repair.

Failure detection in the communication subsystem has been reported by many papers. Chockler et al.

[2008] described failure detectors in wireless networks, which can be utilized for achieving consensus when the communication channel is imperfect [Chockler et al., 2005]. SEAR [Hughes, 2006, Hughes and Cahill, 2009] implements real-time geo-cast and provides feedback on the area a message is delivered to. Whereas many other protocols use acknowledgment and timeout [Kato et al., 2002, Chaimowicz et al., 2004, Lau et al., 2009] to detect communication failure. In order to repair a failure, applications that do not require real-time communication usually resend the messages, and real-time systems may require the entities to adapt to the degradation of communication quality [Verissimo et al., 2000, Bouroche, 2007].

Inaccuracies at the component level (e.g., GPS, sensors) can be alleviated by using prediction models [Dao et al., 2008b, Dao, 2008] and information fusion from multiple sensors [Broggi et al., 2008, Ferguson et al., 2008, Campbell et al., 2010]. Even so, it is not practical to assume that sensor readings are perfect. For instance, Dresner [2009] provides allowances for vehicle position error by representing each vehicle with a boundary, thereby making the vehicle representation bigger. Other than position errors, Yared et al. [2007] handle errors in turning angle and travel distance by having robots reserve a cone-shaped area. While Comheolaíocht does not handle component-level errors, the protocol allows inaccuracies to be addressed by having each entity specify the resources it requires. The advantage is that this method supports heterogeneous entities - for instance, a small entity (physically) or an entity with good sensors and actuators may request less resources.

Handling of entity-level failure has also been considered in many systems. For instance, the DARPA Grand Challenge 2005 and DARPA Urban Challenge 2007 required that each vehicle has a remote control emergency stop button that the officials held during the competition [Campbell et al., 2010]. Task allocation systems for mine sweeping autonomous underwater vehicles [Sariel et al., 2006b, Sotzing et al., 2007, Fallon et al., 2010] detect failures by having each autonomous underwater vehicle send heartbeats messages, failure is detected by timeouts (after missing heartbeats for a certain period). Dresner and Stone [2008a] handle vehicle breakdown in a junction by flooding emergency messages. Their method only mitigates catastrophic disaster by lowering the number of collisions; it does not guarantee that no collision will happen in the event of a breakdown. Comheolaíocht assumes that entity failures can be detected by some other systems with an upper time bound; our protocol uses this constant time with the entity's attributes (e.g., breaking distance) to calculate and allocate allowances in the event of an entity breakdown. The method trades efficiency (delay) for safety; such trade-offs are intentional because Comheolaíocht targets safety-critical applications.

2.2 Multi-Robot Systems

A number of projects have investigated the coordination of robots in multi-entity systems. This section surveys multi-robot systems, lists the properties that they need and shows that most applications require the properties supported by Comheolaíocht (Figure 2.1). Although the majority of these work demonstrate the same set of properties, the solutions are specific to the application scenarios and could not be easily reused. The following sub-section reviews related work in the intelligent transportation systems domain and especially the intersection collision avoidance scenario as this scenario serves as the running example of this thesis. Section 2.2.2 presents systems participating in robotic soccer competitions, and Section 2.2.3 presents systems for multi-entity exploration. Section 2.2.4 describes some other multi-entity applications. Finally, Section 2.2.5 provides an analysis of the systems surveyed.

2.2.1 Intelligent Transportation System

Owing to increasing road congestion and the large number of accidents and fatalities on the road, the United States, Europe and Japan are looking into building intelligent infrastructures to manage roads and vehicles. In the United States, the Federal Highway Administration (FHWA) presented an update of the benefits and costs of their intelligent transportation system [Maccubbin et al., 2003]. The Intelligent Car Initiative[1] (2006) is a policy framework set up by the European Commission to consolidate all activities relating to intelligent automobiles. Its objective is to improve road safety in the European Union, and in particular to reduce the annual 40 thousand road fatalities and 1.2 million road accidents, to decrease the number of traffic jams, and to reduce fuel consumption and road transportation's CO_2 emissions. The Intelligent transportation systems are not just about autonomous vehicles. Project groups participating in the Intelligent Car Initiative include PReVENT [Schulze et al., 2008] and the CVIS[2]. The PReVENT project is developing safety-related applications, using advanced sensors and communication devices integrated into on-board systems for driver assistance. The CVIS project is designing, developing and testing new technologies for vehicular communications.

Autonomous vehicles have been demonstrated in a number of situations. The DARPA Grand Challenge 2005 and DARPA Urban Challenge (DUC) 2007 show autonomous vehicles crossing the dessert and navigating in an urban environment with other vehicles [Broggi et al., 2008, Ferguson et al., 2008, Campbell et al., 2010]. For instance, the vehicle which came first in the DUC, 'Boss' [Ferguson et al., 2008], uses three main components; detection, prediction and avoidance. r Campbell et al. [2010] describe the state of art, share experience of the DUC, and present the challenges for autonomous driving. Since the focus

[1]http://ec.europa.eu/information_society/activities/intelligentcar/index_en.htm
[2]http://www.cvisproject.org/

of the DUC is on single vehicles, these papers provided designs concentrating on the implementation of a single vehicle and do not tackle cooperation among vehicles. However, lessons learned from these physical implementations are relevant to our work; for instance, the paper by Campbell et al. [2010] describes an error in Caltech's vehicle, Alice: when Alice's intersection handling logic was active, another part of the higher-level logic planner switched into a new state when it detected a nearby vehicle, this de-activated the intersection handling logic causing Alice to execute some (as the paper describes it) 'very unsafe behavior'. In order not to repeat Alice's mistakes, Comheolaíocht provides steps for defining and combining different scenarios so that entities may have different behaviors in different scenarios and yet are able to ensure their integrity when transiting into another scenario.

Autonomous vehicles that coordinate to achieve road safety have been demonstrated in multiple situations. As early as year 2000, Kato et al. [2002] demonstrated five autonomous vehicle driving in a platoon, doing maneuvers of lane changing, lane merging, and leaving the platoon. In another experiment, two INRIA vehicles were demonstrated to perform road lane following, overtaking, signalized intersection crossing, leader following and obstacle avoidance [Baber et al., 2005].

2.2.1.1 Intersection

According to the U.S. National Agenda for Intersection Safety [Stollof and Kalla, 2002], in the year 2000, there were more than 2.8 million intersection-related crashes representing 44 percent of all reported crashes. Approximately 8,500 fatalities (23 percent of total fatalities) and almost one million injury crashes occurred at or within an intersection. The cost to society for these intersection-related crashes is approximately $40 billion every year. The U.S. project group CICAS[3] and the European project group INTERSAFE[4] focus on avoiding collision in intersections.

Traffic lights are typically used at intersections to reduce the risk of accidents; they are optimized to minimize a vehicle's wait in an intersection. For instance, a blackboard-based architecture can be used between several intersection agents to adapt and respond to traffic conditions in real-time [Roozemond, 1999]. Traffic lights can be coordinated at multiple junctions to optimize either the north-south or east-west traffic flow [de Oliveira et al., 2005]. The total waiting time can also be minimized by distributing credits to vehicles and allowing the side with most credits the green light [Balan and Luke, 2006]. While these methods optimize vehicle waiting time in a junction, they do not prevent collisions completely.

Autonomous vehicles in intersections A system addressing the intersection collision avoidance scenario should be decentralized, support dynamic participants, handle imperfect communication and ensures

[3]http://www.its.dot.gov/cicas/index.htm
[4]http://prevent.ertico.webhouse.net/en/prevent_subprojects/intersection_safety/intersafe/

	Decentralized	DP	Imperfect communication	No deadlock/starvation
Naumann et al.	yes	no	no	no
Yared et al.	yes	yes	no	yes
Dresner and Stone	no	yes	yes	no
VanMiddlesworth et al.	yes	yes	no	no
Schemmer and Nett	no	yes	yes	no
Bouroche et al.	yes	yes	yes	no

Table 2.1: Intersection collision avoidance

no deadlock or starvation. Although centralized solutions are easier to optimize than decentralized solutions, they are vulnerable to single points of failure and require dedicated infrastructure, which might be expensive. Vehicles could arrive and leave the intersection at any time, providing a natural dynamic participation environment. As mentioned, mobile entities typically communicate over unreliable wireless networks, therefore an implementation of the intersection collision avoidance scenario must addresses imperfect communication as such failures may cost human lives. A protocol with deadlock/starvation implies that some vehicles may never get to cross the intersection, which is undesirable in practice. Table 2.1 provides an overview of the systems that provide coordination of autonomous vehicles crossing an intersection. None of these solutions support all the properties of decentralization, handling dynamic participants, handling imperfect perception, actuation and communication, and ensuring no deadlock or starvation.

Some collision avoidance protocols do not support unreliable communications [Naumann et al., 2002, Li and Wang, 2005, Yared et al., 2007]. Naumann et al. [2002] were the first to propose using communication to coordinate autonomous vehicles crossing a junction. They suggested passing a token between the vehicles with the vehicle holding the token being allowed to cross. The protocol is demonstrated with three robots. Token-based approaches are susceptible to the lost token problem and are not suitable for a system with dynamic participants (Section 2.4.3.1). Li and Wang [2005] describe cooperative driving by comparing vehicles' trajectories. Based on the assumption of inter-vehicular communication and trajectory planning using control systems, they presented a centralized algorithm that calculates the set of vehicles that may cross the junction at the same time. Similarly, their protocol did not consider the effects of imperfect communication and vehicle control. Yared et al. [2007] provided a coordination algorithm for generic multiple robot collision avoidance; the robots do not necessarily cross an intersection. Their work is interesting because it handles robot actuation errors (i.e., position, translation and rotation errors) and supports deadlock prevention; each robot internally models a wait-for graph to prevent deadlocks in reservations. Robots in their work communicate using broadcast to reserve areas in which the vehicle might be traveling; communication failures are not considered in their algorithm.

Approaches for centralized intersection collision avoidance include [Dresner and Stone, 2004, 2005, 2006b, 2007, 2008a,b, Dresner, 2009] and Schemmer et al. [2001].

Dresner and Stone presented multiple papers describing a multi-agent approach for coordinating autonomous vehicles driving through an intersection. Their protocol handles imperfect perception, actuation and communication, however the protocol is centralized and is not shown to be free of deadlock or starvation [Dresner and Stone, 2004, 2005, 2006b, 2007, 2008a,b, Dresner, 2009]. In their system, each junction is divided into grids that are managed by an intersection manager agent. A vehicle approaching the intersection sends a message to the intersection manager to request a space-time slot to cross the intersection. The intersection manager then simulates the vehicle traveling through the junction, calculates the resources that it requires, reserves the required space-time if it is not being used and informs the vehicle of the outcome [Dresner and Stone, 2004]. The protocol ensures that vehicles are safe despite message losses as a vehicle will not cross the intersection until it receives the outcome of the reservation [Dresner and Stone, 2005]. An advantage of the system is that it can be programmed with policies that allow human driven and autonomous vehicles to use the road at the same time [Dresner and Stone, 2006b], moreover such policies can be switched at runtime [Dresner and Stone, 2007]. Dresner's system can mitigate catastrophic failure in the intersection when a vehicle breaks down in the junction by i) reducing the number of crashes by stopping other vehicles from crossing and ii) reducing the impact of vehicle collision by engaging the on-board collision routine [Dresner and Stone, 2008a]. In their system, a broken vehicle sends out a distress message to the intersection manager and the intersection manager floods an emergency channel with breakdown messages. Any vehicle that hears the breakdown message and does not have a reservation stops and vehicles with a reservation switch on their on-board collision avoidance routine. In addition, Dresner's PhD thesis [Dresner, 2009] provides a protocol to address vehicles just after crossing the junction and entering the connecting road. This is done by having vehicles reserve driving space on the connected road from the intersection manager. A vehicle informs the intersection manager that it is leaving the road and in the event that a vehicle does not inform the intersection manager, the intersection manager polls the vehicle and the absence of a reply implies that the vehicle has left. The method of polling and time-outs to detect vehicles leaving the system implies that the system is not completely immune to imperfect communication. Consider the situation where polls from the intersection manager are dropped due to communication errors, and a vehicle is still in the system after the time-out period, the intersection manager may then allow other vehicles to cross the junction and enter a road that is occupied. Dresner and Stone's work is the only previous work that handles vehicles exiting a junction. In contrast, this thesis models this problem as an enter-road constraint for reusability (See Section 3.4.5.1).

Schemmer et al. [2001] presented a method for autonomous vehicles to schedule their crossing an intersection. The method is built on top of their real-time reliable group communication protocol [Schemmer and Nett, 2003b]. The group communication protocol uses a centralized access point (AP) for coordination and assumes that there is a constant number, called the omission degree (OD), that bounds the number of message losses affecting any information transfer between the vehicles and the AP (i.e., at least one message will be delivered when OD+1 messages are sent).

An extension to Dresner's system allows autonomous vehicles to request a crossing slot without a centralized intersection manager [VanMiddlesworth et al., 2008]. The method assumes that at least one message is delivered when a vehicle sends its request multiple times is similar to the OD assumption, which might not always hold.

Bouroche et al. [2006] describe the intersection crossing problem using a safety constraint to ensure that there is at most one vehicle crossing the intersection at a time. The protocol is distributed and handles communication failures. It does not allow multiple vehicles to cross the intersection at the same time, and no proof of deadlock freedom and starvation freedom are available.

A common observation in these papers is that the intersection collision avoidance scenario is modeled as a resource-reservation problem. Naumann et al. [2002] represent the whole junction as a single resource and only the token holding vehicle may proceed to cross the junction. Yared et al. [2007] design their robots to communicate and reserve convex-hull shapes based on a continuous geometrical plane. Dresner [2009] divides the junction into grid and each vehicle specifies its arrival time, speed and desired destination to the intersection manager to reserve a crossing slot. Schemmer et al. [2001] schedule the intersection access and Bouroche et al. [2006] solve the intersection problem as a distributed mutual exclusion problem. Learning from these designs, this thesis describes the intersection collision problem as a shared resource problem and provides distributed scheduling and mutual exclusion protocols for vehicles to cross an intersection safely.

2.2.2 Robotic soccer

RoboCup is an international initiative to promote artificial intelligence and robotics research by providing a common task for its evaluation [Kitano et al., 1997]. The range of technologies involved in the competition includes autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning and planning, intelligent robotics, and sensor fusion. RoboCup is designed to exhibit real-world complexities: a dynamic environment, incomplete sensor information, distributed computation and the need for real-time reaction. Similarly, Comheolaíocht targets dynamic environments, incomplete sensor information, distributed computation and requires real-time reaction. However, robotic soccer is different from Comheolaíocht in three aspects: firstly, robotic soccer involves a known and fixed number of robots while Comheolaíocht supports dynamic participation. Secondly, the cost of a robotics soccer team's failure is benign, whereas Comheolaíocht deals with safety critical-systems. Thirdly, Comheolaíocht's entities are selfish while robotic soccer demonstrates both cooperation among team mates and competition with another team.

Simulated robots in the UvA Trilearn team [Kok et al., 2005], who won the Padova, Italy 2003 Simulated Robot RoboCup, coordinate only amongst robots that are close together using a coordination graph representation [Guestrin et al., 2001]. Robots in the team use wireless communication whenever possible, however, when communication fails, the robots perform role (e.g., forward, defender) assignments using local context with approximations. Similar to UvA Trilearn, Comheolaíocht also coordinates only with robots that are physically close, and has a strategy for handling communication failures.

The CAMBADA team [Azevedo et al., 2007] was ranked first and third in the middle-size RoboCup competition in 2008 and 2009 respectively. Robots in CAMBADA coordinate by sharing their world model with other players [Lau et al., 2009]; the world represented in the real-time database is updated and replicated on all robots in real-time. Robots then deliberate on their role and behavior based on the world model. Commonly used in many team sports such as soccer, rugby and basketball, a set play defines some predetermined steps for each player to follow in specific situations. The use of set plays is one of CAMBADA's winning strategies [Lau et al., 2009, Mota et al., 2011]. CAMBADA implements set play as hard-coded series of steps and alternative paths amongst multiple robot roles. During the execution of a set play, roles are assigned to the robots to act out the predefined steps, in the process, robots may act on an alternative path if the original steps are no longer feasible (e.g., an opponent blocking a pass). Set plays requires messages to be exchanged between the players for synchronization. In Comheolaíocht, set plays can be modeled as sets of sequential constraints where an action is follow by another.

2.2.3 Exploration

Another area for multi-robot coordination is multi-robot exploration [MacKenzie, 2003, Lemaire et al., 2004, Schermerhorn and Scheutz, 2006, Elizondo-Leal et al., 2008]. In multi-robot exploration, robots are tasked to search a particular area and possibly respond to some findings. The exploration task usually involves a large area, imperfect communications and entities may join or leave the system. In contrast to Comheolaíocht, where different-time ordering constraint problems are mapped to shared-resource problems, exploration problems are usually mapped to task-allocation problems. In a shared-resource problem, many entities want to access the same resource, and a protocol must ensure that the entities access the resource at different times (the different-time ordering constraint). Conversely, the task-allocation problem involves many entities mapping to many tasks, and a protocol must ensure that each task is mapped to at least one entity while optimizing some parameters; the time at which an entity performs the allocated task is non-essential (the no-time relation constraint). Therefore, this section only surveys specific applications from multi-robot exploration that require more than task allocation.

Simultaneous auctions Lemaire et al. [2004] present an auction-based protocol [Smith, 1980, Dias et al., 2006] to allocate way points for UAVs performing surveillance. In their paper, identical entities are launched at the same place, have the same internal state and start off at the same time. The authors explain that this is a more probable case than multiple machines being launched at different places. Since all entities start off with the same internal state, the entities could initiate the contract net protocol at the same time for the same tasks, which results in chaos. The authors layer a token-based protocol (Section 2.4.3.1) on top of the auction protocol so that only the entity holding the token can initiate the protocol.

Exploration acquired tasks As mentioned, robots in multi-robot exploration search some area and may respond to some findings. Such findings become new tasks, and the operation on tasks acquired after the exploration phase might be different in each system. Examples of these systems includes search and track [Pavone et al., 2009, How et al., 2009], search and destroy [Sotzing et al., 2007, Sotzing and Lane, 2010, Sariel and Balch, 2006, Sariel et al., 2006b], and search and rescue [Jones et al., 2011]. While most existing work applies the same auction protocol for both allocating entities for exploration and response, such response-tasks might involve constraints that require other properties; some of which may be provided by Comheolaíocht.

An example of a search and destroy mission is mine counter measure (MCM) missions. In the U.S Navy unmanned underwater vehicle Master Plan [Fletcher, 2000], underwater mines are identified as one of the most challenging problems. For this reason, multiple papers describe the deployment of autonomous underwater vehicles (AUV) to detect and neutralize underwater mines [Sotzing et al., 2007, Sotzing and Lane, 2010, Sariel and Balch, 2006, Sariel et al., 2006b]. MCM missions using AUVs are complex because underwater acoustic communications are highly unreliable, limited in range and bandwidth [Sotzing and Lane, 2010]. These missions usually include two different types of AUVs - searchers and neutralizers [Sotzing et al., 2007, Sotzing and Lane, 2010, Sariel and Balch, 2006, Sariel et al., 2006b]. The searcher AUVs explore the sea-bed to find mine-like objects (MLO). When a searcher AUV finds a MLO, it must employ the help of a neutralizer AUV to identify the MLO, and neutralize it if the object is confirmed to be an underwater mine. The danger of mine neutralization coupled with the imperfect navigation and

	Failure Cost	Require DP	System ordering	Event ordering
Intersection	Safety critical	yes	Partial ordered	Different/Sequential
RoboCup	Benign	no	Partial ordered	Different/Sequential
Exploration	Depends on application	maybe	Non-ordered/ Partial ordered	No time relation
MCM	Safety critical	maybe	Partial ordered	Different/No time relation
Construction	Depends on application	no	Total ordered/ Partial ordered	Different/ Sequential/Same
Air traffic control	Safety critical	yes	Partial ordered	Different/Sequential

Table 2.2: Multi-entity system properties

communication results in ensuring only one neutralizer AUV is assigned to the identification and neutralization of each MLO. Such a system requires the different-time ordering event constraint (in ensuring that only one neutralizer AUV is allocated), and ensures that the task is carried out despite communication and entity failures.

2.2.4 Others

There are many other systems that require multi-entity coordination; these include the area of robotics construction where multiple robots coordinate to build a structure [Wawerla et al., 2002, Werfel and Nagpal, 2006], air traffic control where planes coordinate to avoid collision [Dolev et al., 2006, Brown, 2007], entity localization where ground vehicles use UAV [Chaimowicz et al., 2004], and underwater vehicles use surface craft [Fallon et al., 2010] for navigation. Each of these application areas could involve multiple constraints that can be classified as same-time, different-time, sequential and no-time. For instance, air traffic control requires that planes must not take the same route at the same time [Brown, 2007] - a different-time constraint.

2.2.5 Analysis

Table 2.2 shows the different types of multi-entity applications and their properties. Amongst the surveyed applications, intersection collision avoidance and air traffic control problems are safety critical, require dynamic participants, and must ensure different-time and sequential event ordering constraints; which are properties provided by Comheolaíocht.

The failure cost for exploration applications depends on the actual environment for which the protocol is used, for instance, a MCM mission is safety-critical because imperfect coverage could miss finding a mine, which may damages ships and lives. In another instance, robot exploration for vacuuming a floor is benign. Exploration tasks may require dynamic participants as the area to be explore could be large and entities' presence could be dynamic with respect to the local environment. While exploration tasks do not require different-time or sequence constraints, applications with exploration may face complications like simultaneous auctions or require responses to acquired tasks.

Similar to exploration applications, RoboCup and construction applications may seem unlikely candidate for Comheolaíocht where dynamic participants may not be required and the task is not safety critical. However, there may be complications or opportunities that require properties provided by Comheolaíocht, for instance - set play in RobuCup.

2.3 Generic Coordination Protocols - Scalability Focus

As mentioned in Section 2.1.4.1, many coordination methods organize agents into teams, hierarchies, and federations so as to support scalability. The organization of a multi-entity system is the collection of roles, relationships, and authority structures that govern entities' behaviors [Horling and Lesser, 2004]. Such an organization guides how members of the population interact with one another. Horling and Lesser provide a survey of interaction organizations like hierarchies, coalitions, teams, societies etc. The survey describes the benefits and drawbacks of each organization, provides example applications implementing the organization and discusses the formation/maintenance of these organizations. This section reviews the organizations classified by Horling and Lesser [2004] with respect to system scalability.

2.3.1 Hierarchy and Matrix

Hierarchical organization has been used in many early multi-entity systems [Agrawal and Abbadi, 1991, Ferber, 1999]. Hierarchical organizations are scalable because the number of interactions is small relative to the total population. However, the use of hierarchy implies a centralized protocol with a potentially central point of failure.

Matrix organizations relax the one entity/one manager restriction in hierarchical organizations; entities may have more than one parent, thereby forming a lattice graph interaction structure. Both hierarchical and matrix organizations have fixed interaction patterns that do not support dynamic participants.

2.3.2 Federation

Another commonly seen interaction organization is federation. In federations, a group of entities come together and have ceded some amount of autonomy to a single delegate that represents the group. An example of a federation implementation is Space Elastic Adaptive Routing [Hughes, 2006], the protocol divides the environment into cells and communication with entities across cells must be sent through an elected gateway. Another use of federation is the virtual stationary automata [Dolev et al., 2006] in which an operating area is divided into segments and a virtual leader is always present in the segment to perform centralized calculations.

In general, federations provide an excellent way of scaling a huge system as each group of entities only needs to coordinate amongst themselves, and only the representatives need to coordinate across groups. However, federation can be complex in a system with dynamic participants as entities may need to enter or leave a federation to join another. Another complexity is that entities cannot distinguish between deficient communication and the absence of other entities [Bouroche, 2007]. While Brown [2007] presented an air traffic control system using the virtual stationary automata [Dolev et al., 2006], the implementation assumes perfect communication; while the author showed that perfect communication is attainable in air traffic control, the same assumption may not be applicable in other systems.

2.3.3 Societies

Societies are inherently open systems; entities may come and go at will while the society persists [Horling and Lesser, 2004]. Entities within the society follow a set of constraints that are commonly known as social laws, norms or conventions - the set of rules or guidelines on entities' behaviors to facilitate coexistence. An example of a society organization is the use of contracts in Comhordú [Bouroche, 2007]. Another example are swarm-based systems where entities coordinate by following some common rules [Werfel et al., 2008, Parker and Zhang, 2009, Ioannidis et al., 2011].

Society organizations naturally support dynamic participants. However, there are no structure defined for entity interaction; communication between entities is based on the set of social rules, which might be different between applications. This means that the scalability of societies with regards to communication bandwidth, memory and computational resources cannot be determined unless the actual set of social rules are evaluated.

2.3.4 Market

Market-based organizations have entities that take up roles as sellers and buyers [Horling and Lesser, 2004]. Buying entities may request or place bids for items such as resources or tasks. Selling entities (auctioneers) are responsible for processing bids and determining winners [Gerkey and Matarić, 2004, Dias et al., 2006, Zlot and Stentz, 2006]. Market-based organizations have been applied to areas like robotic soccer [Mota et al., 2011], exploration [Lemaire et al., 2004] and mine countermeasures [Sariel et al., 2006b, Sotzing et al., 2007] to perform optimal role assignments [Kok et al., 2005, Mota et al., 2011], resource allocations [Huang et al., 2008] and task allocations [MacKenzie, 2003, Sariel and Balch, 2006]. Gerkey and Matarić [2004] and Dias et al. [2006] provided a good survey paper on market-based organizations.

It is common to have market-based organizations operate as open systems, allowing entities to take part as long as it respect the system's rules; as such market-based organizations can be quite similar to societies. Market-based organizations, like societies, are scalable. However, the existence of auctioneers are similar to federated systems where a distinguished entity is (or a group of entities are) responsible for coordinating other entities. Therefore, market-based organizations also have the same problems as federations in the management of the auctioneers (e.g., handing over when auctioneer wants to leave the system, or coordinating between multiple simultaneous auctioneers [Lemaire et al., 2004]).

2.3.5 Coalition, Team, Holarchy and Congregate

As mentioned in Section 2.1.2.3, Comheolaíocht does not support entity grouping, therefore, coalition, team, holarchy and congregate are not in the scope of this work.

2.3.6 Analysis

Not all system organizations fit into a single category; a system architecture may include characteristics of different styles of organization. For instance, Comhordú [Bouroche, 2007] implements a society-based architecture for entities coordination which is built on top of Space Elastic Adaptive Routing [Hughes, 2006] - a federation-based architecture for communication.

Of the organizations surveyed, societies and markets support open systems by having a set of rules governing entities' behaviors; markets and federations have a distinguished entity for directing other entities; and hierarchies and matrices has lesser interactions with respect to the number of entities. In view of the requirement for dynamic participants, hierarchies and matrices are not suitable in this work. Similarly, dynamic participants complicate the management of the distinguished entity in markets and federations.

This work chooses to use the contracts defined in Comhordú [Bouroche, 2007] for entities coordination; a society-based organization where entities are expected to follow the protocol. As mentioned, society inherently supports dynamic participants, but entity interaction structure are not defined. This work include social rules for entity interaction and provides scalability in entity numbers.

2.4 Generic Coordination Protocols - Reliability focus

A reliable protocol must ensure that safety constraints are satisfied at all times despite failures. Comheolaíocht supports two of the four classes of constraints for specifying safety (i.e., sequential and different-time), this section surveys the coordination protocols supporting these two classes of constraints. As most coordination protocols use the consensus property, this section first surveys the consensus problem, and shows why a consensus-based protocol is not suitable in Comheolaíocht. Following this, the section reviews coordination protocols supporting the sequential and different-time event ordering constraints.

2.4.1 Consensus

Consensus cannot be solved deterministically in asynchronous distributed systems in the presence of failures [Fischer et al., 1985, Lynch, 1996]. In the synchronous case, a solution referred to as the 'Byzantine Generals' [Lamport et al., 1982] is known. Under a synchronous system, Chockler et al. [2005] show that consensus cannot be solved even with the assumption of eventual collision free messages without knowledge of the total number of nodes. The authors show that under a network partition, if the total number of nodes is unknown, each part may arrive at their own consensus result.

Charron-Bost and Schiper [2007] claim that most consensus protocols exhibit three harmful dogmas, and introduce the heard-Of (HO) model, which is free from these dogmas. The HO model's computation consists of rounds, and assumes that a message that is sent and not received within the same round is lost; message delays are considered faults. This handles the first dogma: most papers distinguish between message delays and faults. In each round, the HO model only records the nodes from which a node received messages; there is no differentiation in the HO model between process and link failures, which addresses the second dogma. Instead of having "every correct/good process eventually decides", the HO model does not label a single process as 'incorrect' or 'not good' but instead requires two-thirds of the entities to agree on the same value for consensus; this addresses the third dogma. The Paxos/LastVoting protocol [Lamport, 2001] is extended with the HO model to provide consensus for wireless ad hoc networks [Borran et al., 2008a]. There are two reasons why these protocols are not suitable for an application with dynamic participants. Firstly, both the HO model and the LastVoting protocol extension require multiple rounds to reach consensus, and neither of these protocols address the problem of entities entering or leaving when the protocol is in progress. Secondly, the HO model terminates when two-thirds of the entities agree and the Paxos/LastVoting decides when half of the entities agree, implying that both models need to know the total number of nodes.

Ren et al. [2005] and Olfati-Saber et al. [2007] surveyed the consensus problem in multi-agent cooperative control systems. In the surveys, Ren et al. [2005] model the world as a graph $G = (V, E)$ and Olfati-Saber et al. [2007] model the world as a dynamic graph $G(t) = (V, E(t))$ in which V is the set of vertices representing entities/nodes and E is the set of edges representing entities who can communicate with each other. In the Olfati-Saber et al. model, the set of edges, $E(t)$ are time varying; entities' communication ability with each other changes with time. The models, described in both papers are only best-effort, use a fixed V and therefore do not support dynamic participants.

Parker and Zhang [2009] propose a solution to the decentralized best-of-N decision problem in which a decentralized multi-robot system must unanimously select one of N alternatives. Unlike the other consensus algorithms, this swarm-based approach to consensus does not require an entity to know the number of entities. However, the protocol does not consider the case of a network partition and is based on best-effort. Moreover, the protocol requires many communication rounds to reach consensus.

The protocols reviewed take multiple rounds to reach consensus and do not consider changes in entity participation during execution. Moreover, these protocols either assume that the total number of entities is fixed [Ren et al., 2005, Olfati-Saber et al., 2007] or that there is an upper-bound on the number of entities [Charron-Bost and Schiper, 2007, Borran et al., 2008a]. Such protocols are not applicable in the dynamic participant environment where entities could enter or leave the system at any time (i.e., when a consensus protocol is on-going).

In contrast, Comheolaíocht does not rely on consensus, however, it requires a multicast protocol that provides ordered delivery, bounded message latency and real-time feedback (Section 5.1 provides more details on these requirements and shows that the requirements can be achieved in a system with dynamic participants).

2.4.2 Sequential Ordering

Problems with the sequential ordering constraint require that events happen one before another. There are two types of sequence problems: *dependency* constraints and *priority* constraints. Two events, x and y, have the *dependency* constraint, when event x must happen before event y can be executed. The same two events have the *priority* constraint, such that if events x and y are to happen, then x should happen before y, in this case, event y can be executed without event x. An example of a dependency constraint problem is the set of producer-consumer problems [Ben-Ari, 2006]. Other examples of dependencies can be seen in applications like construction [Wawerla et al., 2002, Werfel and Nagpal, 2006] and robot soccer's set play [Lau et al., 2009, Mota et al., 2011]. An example of an application with priority constraints is a transportation system where normal vehicles are to give way to higher-priority emergency vehicles [Dresner and Stone, 2006b, Senart et al., 2008]. Most solutions providing sequence ordering involve applying special rules for some specific situations, which means that they cannot be easily reused.

To our knowledge, only Alami et al. [1998] and Werfel and Nagpal [2006] consider the possibility of a deadlock that may result from the dependency constraints. However, Werfel and Nagpals' protocol uses specific rules to prevent deadlocks, and Alami et al. proposed coordination protocol is based on plan sharing, which uses the consensus property and involves only a fixed number of robots.

2.4.3 Different-time Ordering

Problems with the different-time ordering constraint ensure that events do not happen at the same time; entities must ensure exclusive execution of the events. Existing solutions for different-time ordering constraints can be grouped under two categories: mutual exclusion or scheduling.

2.4.3.1 Mutual exclusion

Current implementations of mutual exclusion use either token-based or permission-based methods.

Token-based approaches In token-based mutual exclusion, a token is passed among the participants and the entity holding the token can have exclusive access to the critical section. While token-based mutual exclusion has been implemented in robots crossing an intersection [Naumann et al., 2002], and may tolerate communication failures [Chang et al., 1990], to our knowledge, there are no token-based mutual exclusion protocols that support dynamic participation; for example Naumann et al.s' demonstration consists of three robots.

Permission-based approaches Permission-based approaches involve an entity sending a request to someone (or everyone), which will reply with their agreement on whether the entity can access the critical section. Mutual exclusion based on permission can be sub-categorized into quorum-based or consensus-based. In quorum-based approaches [Maekawa, 1985, Agrawal and Abbadi, 1991] a node acquires permission from a predetermined sub-set of all nodes to enter a critical section. Maekawa [1985] describes an approach whereby a node only needs $O(\sqrt{n})$ (where n is the number of nodes) messages to gain entry into a critical section. Their method, however, is not tolerant to nodes leaving the system (e.g., node failures) without proper handing over of responsibilities. Agrawal and Abbadi [1991] describe a tree-based method to obtain mutual exclusion that is tolerant of node failures; this algorithm, however, assumes a fixed number of nodes. Solutions based on quorum assume that nodes are static so as to pre-calculate the particular sub-set from which to request permission. The static-nodes assumption makes quorum-based approaches unsuitable for scenarios with dynamic participation.

In consensus-based approaches, a node requests permission from all nodes to enter the critical section [Ricart and Agrawala, 1981, Wu et al., 2008, Attiya et al., 2010]. Properties of consensus-based mutual exclusion are derived from the consensus problem in distributed systems [Lynch, 1996] and inherit the same impossibility results [Fischer et al., 1985, Chockler et al., 2005]. Due to these impossibilities, work on consensus-based mutual exclusion assumes synchronous systems with knowledge of the total number of nodes. Ricart and Agrawala [1981] were the first to provide mutual exclusion using the consensus approach, their algorithm uses only $2 * (n - 1)$ messages (where n is the number of nodes). In the algorithm, a node attempting to enter the critical section sends a request to all other nodes. A node receiving a request replies to the request if it does not want to enter the critical section or has a lower priority. In the case where the receiver wants to enter the critical section and has higher priority, it delays replying until it leaves the critical section. The requester enters the critical section only when it receives a reply from every node. Wu et al. [2008] revisited Ricart and Agrawalas' algorithm and extended it to mobile ad hoc networks by handling links and nodes' transient failure; a node could be moving to another part of the environment and becomes disconnected for a while. The authors assume that the node or link will recover within a time-out period, therefore the requester has to resend its request after the time-out period. Ricart and Agrawala's, and Wu et al. assumption of a known total number of nodes and eventual recovery of node and link failures are not suitable for dynamic participation scenarios. Borran et al. [2008a] extended the LastVoting algorithm [Lamport, 2001] to provide consensus in wireless ad hoc networks, where the algorithm works by gathering a majority of votes. In the LastVoting algorithm, the number of nodes must be between an upper bound, n, and a lower bound, $\frac{n}{2}$. Moreover, the LastVoting algorithm takes four rounds to reach a consensus and changes in participants within the four rounds are not considered. Attiya et al. [2010] provide mutual exclusion in mobile ad hoc networks by first using a set of 'doors' to ensure that the current set of nodes participating becomes static before using an implementation of the Ricart and Agrawala algorithm on the static group to obtain mutual exclusion. The set of 'doors' works by excluding nodes that are not present when the protocol starts from the 'current set' and nodes within the 'current set' must announce their departure. Their algorithm does not cater for network partitions.

2.4.3.2 Scheduling

There is a lot of work done onr centralized scheduling [Liu and Layland, 1973, Manimaran and Murthy, 1998, Kutanoglu and Wu, 1999, Pinedo, 2008], especially in processors where there is a job queue. Pinedo [2008] provides a taxonomy of scheduling problems, Section 3.4.3.2 in this thesis uses Pinedo's taxonomy to specify resource scheduling constraints in Comheolaíocht.

Current distributed scheduling protocols are usually used for the allocation of resources for wireless networks [Zhou et al., 2010, Sharma et al., 2010, Liu et al., 2008, Gupta et al., 2009]. These papers either do not consider imperfect communication or do not ensure exclusive access to resources. Zhou et al. [2010] formulate the scheduling of video streams over multi-channel, multi-radio, multi-hop wireless networks as a convex optimization problem and propose a distributed solution by jointly considering channel assignment, rate allocation and routing. Sharma et al. [2010] propose two randomized distributed algorithms for maximal scheduling policy to control congestion under the 1-hop and 2-hop interference models. Liu et al. [2008] propose a distributed scheduling algorithm to organize the resources in the physical and MAC layers to increase network goodput, decrease end-to-end packet delay, and achieve less QoS outage probability. Gupta et al. [2009] propose two algorithms for scheduling resources in wireless ad hoc networks.

To our knowledge, the only distributed scheduling protocol that considers imperfect communication and ensures exclusive access to resources is by Schemmer et al. [2001]. Their protocol schedules autonomous vehicles access to intersections by building on top of their real-time reliable group communication protocol [Schemmer and Nett, 2003b]. However, their group communication protocol uses a centralized access point (AP) and provides consensus on the assumption that at least one message will be delivered when a given number, OD + 1, of messages are sent. The algorithm is therefore not suitable in situations where there may be an extended period of communication breakdown.

Scheduling in real-time environments by using a worst case execution time analysis was first described by Liu and Layland [1973]. Fault-tolerant real-time scheduling has also been implemented for processes (e.g., sensor fusion) within a robot [Becker et al., 2005].

2.4.4 Analysis

This section surveyed coordination protocols based on their event ordering constraints. Current consensus protocols take multiple rounds to reach consensus and assume either fixed [Ren et al., 2005, Olfati-Saber et al., 2007] or an upper-bound on the number of entities [Charron-Bost and Schiper, 2007, Borran et al., 2008a]. Such protocols are not applicable in the dynamic participation environment where entities can enter or leave the system at any time (i.e., when a consensus protocol is on-going).

Most solutions for sequence-ordering problems apply specific rules to a specific situation, therefore such implementations are not easily reusable. Moreover, problems with dependency constraints might encounter deadlock. While Alami et al. [1998] handle both the generic case, and detect and resolve deadlocks, their method uses consensus and is not suitable in a dynamic participants scenario. While Senart et al.s' [2008] solution is distributed, handles dynamic participants and safety-critical requirements, their solution is specific to emergency vehicles on the road and is difficult to reuse in other problems involving dependency and priority. In contrast, Comheolaíocht allows the specification of sequence constraints and priorities, and ensures that the developed protocol does not have live-locks or deadlocks.

Different-time ordering constraints can be modeled as exclusive access to shared resources. Access to shared resources can be controlled by schedules or mutual exclusion. Current distributed scheduling methods usually do not consider communication failures or do not ensure exclusive allocation; these methods cannot be applied to a situation with safety-critical requirements. Distributed mutual exclusion is implemented using either token-based or permission-based methods. Dynamic participation amplifies the lost token problem in token-based approaches, while limited knowledge of the number of nodes makes obtaining quora and consensus in permission-based approaches impossible, rendering both mutual exclusion implementations impractical. In contrast, Comheolaíocht provides distributed real-time scheduling and mutual exclusion without relying on consensus. Our protocol is able to ensure exclusive allocations even with dynamic participation and imperfect communication.

2.5 Real-time Coordination Middleware

The main function of a middleware system is to facilitate the communication and coordination of components distributed across several networked hosts, it should enable application engineers to abstract from the low-level details of network communication, coordination, reliability, scalability and heterogeneity [Emmerich, 2000]. The challenges of middleware for mobile ad hoc networks (MANET) are heterogeneity, scalability, limited resources, context awareness, mobility and dynamic network topology [Hadim et al., 2006]. Hadim et al. [2006] survey nine middleware systems, and only two: LIME [Murphy et al., 2001] and STEAM [Meier and Cahill, 2002] support scalability, mobility and heterogeneity.

This section presents a survey of middleware that supports the coordination of real-time mobile entities. The section evaluates each of the presented middleware by its ability to support dynamic participants, ability to specify and ensure ordering constraints, and provision for communication and entity failure.

2.5.1 LIME

Linda in Mobile Environment (LIME) [Murphy et al., 2001], is built on top of Linda [Gelernter, 1985]. Where Linda shares information through a central tuple storage, LIME supports the physical mobility of hosts and logical mobility of entities by having each entity maintain its own interface tuple space.

Connected entities share their tuple spaces as a single federated tuple space. When an entity arrives into communication range of a group, a message is sent to all entities in the group to freeze their operations and synchronize tuples in the federated tuple space. The freeze is lifted only after all the tuples are synchronized. A similar freeze and un-freeze step happens when an entity leaves the communication area, where the tuples of the leaving entity are removed from the federated tuple space. Reactions in LIME allow developers to program entities to run specific code when a certain tuple appears. Strong reactions are executed atomically whenever the tuple appears but they only match tuples that are on the same host as the reaction code. Weak reactions do not execute atomically, but react to tuples matched in the federate tuple space. LIME's assumptions that the network is not highly dynamic, can sustain a connection during a transaction, and having to freeze every entity whenever there is an entry or an exit is not practical in an environment with dynamic participants. While strong reactions allow an entity to react in real-time to local events, LIME provides no real-time guarantees with respect to events on a remote host. In addition, the virtual federated space is implemented by replication on every hosts, which could cause problems in a host with limited storage space especially when the number of connected entities are large.

Limone [Fok et al., 2004] extends LIME with context management thereby allowing entities to discover neighbors and to selectively duplicate tuples depending on the application requirements and network settings. The selective duplication conserves bandwidth and memory, which are limited resources in multi-entity systems. In addition, Limone does not assume connectedness throughout a transaction and uses timeouts to prevent deadlock due to packet loss or disconnection.

TinyLIME [Curino et al., 2005] extends LIME for sensor networks by focusing on the synchronization and reaction in LIME. It defines 'Time-Epochs', defined as the minimum unit of time, to synchronize the distributed sensors. Sensor data is defined in terms of n-epochs freshness. When some process requests some data, fresh data (epochs $\leq n$) is returned and stale data (epochs $> n$) is replaced. Reactions are redefined so that they can be fired every n-epoch. TeenyLIME [Costa et al., 2006, 2007] also extends LIME for sensor networks, targeting nodes with limited resources. TeenyLIME is designed for sensor nodes that do not have lots of intelligence or memory; sensor nodes must be connected to a master agent. Instead of replicating everything, resources can be saved by limiting replication in the federated tuple space through a replication profile. Although TinyLIME allows better definition of reactions and TeenyLIME conserves memory and bandwidth resources, both extensions are targeted for sensor networks and are still unable to address dynamic participants or provide real-time guarantees for remote events.

The CAST coordination model extends LIME to provide primitives that allow entities to coordinate even when the entities are not connected in space and time [Roman et al., 2006]. CAST achieves this functionality by allowing mobile entities to store a message when the destination is not connected; when an entity's movement brings it to the message's destination, the stored message is then forwarded. In addition, spatio-temporal operations are defined for operations to execute at specific locations in space and at certain points in time. TNM extends CAST to support real-time coordination [Hackmann et al., 2005]. TNM offers primitives that allow agents to negotiate timed and untimed sequences of actions. However, neither CAST nor TNM redefines the procedure for an entity entering and leaving a federation in LIME; implying that all other entities need to freeze their actions when the federated tuple space is synchronized. The freeze operation is not practical in a dynamic participation environment where entities may enter and leave at a high rate. Since an entity knows the total number of entities in the federation, entity negotiation in TNM may use consensus-based coordination for handling event-ordering constraints. Assuming that Limone [Fok et al., 2004] is integrated with CAST or TNM, then communication errors are addressed using timeouts on remote transactions. However, nothing is mentioned about handling entity failure.

2.5.2 STEAM, TBMAC, SEAR and Comhordú

STEAM [Meier and Cahill, 2002] is an event-based middleware service designed specifically for mobile environments. The middleware builds on the observation that entities in a mobile environment are typically interested in events produced by other entities within a certain geographical area. For this reason, STEAM supports proximity filters in addition to the normal subject and content filters. Subject and proximity filters help address the scalability problem; the filters are applied at the publisher side to limit forwarding of event messages thereby reducing consumption of communication and computation resource.

A real-time version of STEAM is proposed by Hughes et al. [2004] and is implemented as space elastic adaptive routing, SEAR [Hughes, 2006, Hughes and Cahill, 2009]. SEAR provides real-time routing and feedback based on the TBMAC protocol [Cunningham and Cahill, 2002]. TBMAC proposes to divide the application environment into cells such that adjacent cells use different transmission frequencies. A node joining a cell has to observe one TDMA cycle before making a request for an empty slot during the TDMA contention period in the following cycle. Messages sent during the contention period may collide, resulting in the entity waiting for another cycle - providing no guarantee of the time with which an entity may join a cell. In addition, there is no mention of real-time hand-off when an entity is about to move to another cell (e.g., by reserving a channel for the target cell in advance). A node leaving the cell either broadcasts its intention to leave during its allocated sending slot, or the slot can be reclaimed when other entities have not heard any information during the allocated period.

SEAR is a real-time routing protocol that guarantees transmission and adaptation notification latency supporting space-elastic applications. The set of space-elastic applications are defined as programs that require real-time communication to a certain area (desired coverage); in cases where the desired coverage cannot be achieved, the routing protocol will adapt the desired coverage to a smaller area (actual coverage) and feedback to the program within a fixed time (i.e., the adaptation notification latency). The space-elastic application can then decide on its reaction to the smaller actual coverage.

Comhordú [Bouroche, 2007] provides real-time coordination of mobile entities by using the real-time feedback in SEAR [Hughes, 2006, Hughes and Cahill, 2009]. Comhordú's protocol coordinates by first allocating responsibilities to entities; such that to each specified system-level safety-constraint, a responsible entity is specified. When a request for other entities to change behavior fails (i.e., the actual coverage is less than the desired coverage), the responsible entity must ensure that the safety constraint is not violated either by changing its behavior, delaying its action or requesting other entities to change their behavior. Comhordú defines a parameter, *present*, measuring the time for which an entity listens on the communication channel before starting coordinating (this step is termed *lurking*). Since the required time for an entity to join a cell and start sending messages may not be bounded in TBMAC, it might be difficult to choose a value for *present*. In Comhordú, however since an entity who is unable to send a message acts responsibly, so entities' safety is still ensured.

The Comhordú-SEAR-TBMAC combination supports dynamic participants in that a joining entity may receive messages and a leaving entity is not required to announce its intention. However, support for dynamic participation is limited by the possible break in communication whenever a node moves across cells. Comhordú addresses different-time event ordering through its cardinality constraint, however it only provides a mutual exclusion solution. The protocol does not support the same-time and sequential ordering constraints. Comhordú can guarantee entities' safety even when communication fails, however, entity failure is not supported.

2.5.3 TicTac

TicTac [Allouche and Daigle, 2006] is a real-time coordination framework designed for the coordination of multiple entities in command and control (C2) operations. In TicTac, a temporal plan is built using a set of temporal constraints for specifying events that have the same time, before/after, and independence ordering. The framework allows a developer to perform offline planning and check the consistency of temporal plans. TicTac can be used to define three types of coordination: *dependent*, *compete* and *help*. The *dependent* relation exists when an entity cannot execute an action but another entity can. TicTac coordinates *dependent* relation by having the entity who can perform the action send a message to the other entity when the action is completed. *Compete* relations exist when either of two entities is able to execute an action. TicTac handles this type of relations by having the entities negotiate and the entity with the lightest workload is to perform the action. The *help* relation exists when two events cannot happen at the same time; *help* relations are coordinated by having an entity send a message to the other entities both when it starts and stops executing the action.

TicTac does not support dynamic participation; the number of entities and their respective actions are specified offline using the temporal plans. The framework supports events ordering. Sequence and different time ordering events are supported through TicTac's *dependent* and *help* relation, however, coordination is only among the fixed entities specified in the temporal plan, and there is no mention of how the protocol addresses race conditions. There is no mention of communication or entity failure in the framework.

2.5.4 A middleware for cooperating mobile embedded systems

Schemmer et al. [2001] and Schemmer [2004] presented a middleware for developing mobile autonomous systems with real-time cooperation. The middleware provided three main contributions: local real-time task scheduling, real-time group communication and distributed scheduling.

Schemmer and Nett [2003a] presented a method for local real-time task scheduling (e.g., sensor fusion) in the CPU and Becker et al. [2005] provided an analysis of the algorithm. They argued that the usage of worse case execution time to schedule processes with large execution time variances is wasteful on CPU resources, and proposed to schedule CPU resources using expected case execution time. In order to work with expected-case execution times and still achieve predictable timing behavior, their algorithm defines an exception part to be executed when the main part is about to miss its deadline.

Their second contribution, a real-time group communication protocol, provides real-time atomic broadcast [Nett et al., 2001] and a real-time membership service [Nett and Schemmer, 2003, Schemmer and Nett, 2003b]. The protocols exhibit properties of validity, agreement, integrity, total order and timeliness based on three assumptions: the access point (AP) does not have crash failures, there is an upper bound on team-size, and there exists an omission degree (OD) defining an upper-bound on the number of message losses affecting any information transferred between an AP and the station.

Their third contribution, the distributed scheduling protocol uses the real-time group communication protocol, to maintain a globally consistent view among distributed vehicles [Schemmer et al., 2001]. The global view is kept consistent by informing every newcomer of the current view, and having each entity calculate the new view based on the messages received. While the protocol allows nodes to be subjected to crash failures, the crashed nodes are only detected and are removed from the membership after OD+1 polling messages.

The middleware can handle dynamic participants (up to an upper-bound), perform event ordering by scheduling, allow message failures (at most OD messages lost in OD+1 attempts) and entity failures (by removing them from the system). However, the middleware relies on centralized points in the network (i.e., the AP) and there is no mention of other entities' reaction to entity failure.

2.5.5 Timely computing base and wormhole

Veríssimo [2006] presented the use of hybrid (v.s. homogeneous) models, where there are different synchrony levels in a system, to overcome some of the difficulties faced when asynchronous models meet timing specifications. The author describes the Wormhole model which can maintain some strong properties (e.g., synchrony) whilst preserving the model's weak abstractions. The idea of a Wormhole is similar to the Timely Computing Base (TCB) [Veríssimo and Casimiro, 2002]. The TCB is a component within the system that is assumed to have known upper bounds on processing delays, rate of local clock drifts, and message delivery delays. These assumptions allow the TCB to provide other components (which might be asynchronous) with timely execution, duration measurement and timing failure detection. A payload system of any degree of asynchrony can be transformed into a synchronous subsystem using the TCB [Verissimo et al., 2000]. The authors also discuss the implementation of the TCB in the class of fail-safe applications which exhibit correct behavior or else stop in a fail-safe state. Casimiro and Verissimo [2001] showed the usage of TCB in implementing a class of time-elastic applications; where time bounds can increase or decrease dynamically. Casimiro and Verissimo [2002] introduce a paradigm for generic timing fault tolerance with replicated state machines based on the existence of services provided by the TCB. More recently, Casimiro et al. [2009] discuss the use of hybrid models to deal with the intrinsic uncertainties of wireless and mobile environments in the design of distributed embedded systems. Their system applies the wormhole hybrid model to provide fault handling operations for safety purposes. Generic asynchronous payloads use a wormhole which provides a Timely Timing Failure Detection (TTFD) service, to detect timing failures in the payload part or in the payload-wormhole interactions.

The TCB model relies on the assumption that synchronous properties, such as known bounds on processing and message delivery delays are achievable and maintained. Although Casimiro et al. [2009] discuss the usage of the wormhole in the mobile wireless environment, the paper target an environment with a fixed number of entities where it might be possible to satisfy these assumptions. Therefore, it is not clear whether these synchrony properties can be assumed in an environment with dynamic participants using wireless communication.

	Entities joining	Entities leaving	Ordering	Communication failure	Entity failure
CAST & TNM	freeze transactions	freeze transactions	no information	handled	no support
SEAR & Comhordú	wait & listen (lurking)	announce/ time-out	different time	handled	no support
TicTac	no support	no support	same time, different time, sequential	no support	no support
Schemmer [2004]	leader update	announce/ time-out	different time, no information	$\leq$ omission degree (OD)	remove from team
TCB & Wormhole	no information	no information	no information	detected by the TCB	detected by the TCB

Table 2.3: Summary of the evaluation of real-time coordination middleware

2.5.6 Analysis

Table 2.3 shows a summary of real time coordination middleware systems and how they support dynamic participants joining (**DP - Join**) and leaving (**DP - Leave**), event **ordering** constraints, communication failure and entity failure.

Middleware that supports dynamic participants require that entities to go through a joining and leaving step. The joining step involves allowing the newcomer to obtain updated information either by freezing every transaction [Murphy et al., 2001], waiting and listening for a certain period [Cunningham and Cahill, 2002] or having a fixed entity to send the update information [Schemmer and Nett, 2003b]. Comheolaíocht uses the same idea as Cunningham and Cahill [2002] where entities joining the system have to wait for a certain period. The method is chosen because the waiting only affects the entity joining.

Support for different-time event ordering is supported by Comhordú, TicTac and Schemmer [2004]. However, Comhordú only provides mutual exclusion and TicTac assumes that the constraints are pre-computed. Only Schemmer provides exclusive access through scheduling thereby allowing entities to plan ahead. Support for same-time ordering events can be found in TicTac, whereas Schemmer and TCB might be able to support same-time ordering because these middleware systems rely on consensus. In contrast, Comheolaíocht only supports different-time and sequence ordering constraints, this is because Comheolaíocht's coordination model (similar to Comhordú) is not based on consensus.

Except for TicTac, all other middleware systems surveyed support communication failures. However, Schemmer [2004] only supports up to OD consecutive failures. CAST & TNM (assuming Limone is integrated) supports communication failures by providing roll-back of transactions. Only Wormhole & TCB and SEAR supports real-time mobile entities by providing detection of communication failures using time-outs, and allows entities to adapt their behaviors to the failure. Like Comhordú, Comheolaíocht also assumes a protocol that provides real-time feedback on communication failures, and defines entity behavior adaption in the event of such a failure.

Wormhole is able to detect entity failure and Schemmer's middleware can remove a failed entity from the system. Comheolaíocht brings the handling of entity failure a step further and considers the effects of an entity failure on other entities in the vicinity; it provides a protocol that ensures enough allowance is made for reaction to an entity failure.

2.6 Analysis

This section first presents a summary of the comparisons made in this chapter. The section then concludes the chapter by listing some of the work that influences this work.

2.6.1 Comparison Summary

This chapter evaluates related work based on two parameters: reliability and scalability (Figure 2.1 provides a representation of the evaluated properties).

2.6.1.1 Reliability

In the evaluation of system reliability, support for safety and liveness properties are evaluated. With regards to safety properties, each work is evaluated on its support for the different time and sequential event ordering constraints to support safety constraints, and if these constraints can be ensured despite communication and entity failures. With regards to liveness properties, systems are evaluated for any guarantees of a system's goal being eventually achieved, in particular, whether the system handles deadlocks and live-locks, which might be present due to interleaving constraints.

Safety Most existing systems provide different-time and sequential-ordering constraints based on consensus. However, current consensus protocols do not support environments with dynamic participants. In addition, most current solutions supporting sequential-ordering constraints are only applicable to a specific application. Generic support for different-time ordering constraints can be modeled as scheduling or mutual exclusion problems. However, current distributed scheduling methods usually do not consider communication failures or do not ensure exclusive allocation. Distributed mutual exclusion is implemented using either token-based or permission-based methods. Token-based approaches do not support dynamic participants, while permission-based approaches require knowledge on the number of nodes, rendering both mutual exclusion implementations impractical.

Comheolaíocht provides an approach for developing distributed real-time scheduling and mutual exclusion protocols using a permission-based approach without relying on consensus. In place of consensus, it is based on a communication protocol that provides ordered delivery, bounded message latency and real-time feedback. Ordered delivery provides the necessary conditions for ensuring that the any entities that make a decision, decide on the same things, and bounded message latency and real-time feedback ensures termination; that entities eventually makes a decision. Chapter 5 describes the derivation of the coordination protocol and Section 5.1 provides details on the communication protocol requirements, the same section also shows that the requirements can be achieved in a system with dynamic participants.

Liveness While many mutual exclusion protocols are proven to be free from deadlock and live-lock, those protocols are based on centralized systems. To our knowledge, distributed protocols that are shown to be free from deadlock and live-lock only support specific applications (e.g., [Yared et al., 2007]) or do not support dynamic participants [Allouche and Daigle, 2006].

In contrast, the protocols that Comheolaíocht generates are free from both live-locks and deadlocks.

2.6.1.2 Scalability

In this work, the evaluation of a system's support for scalability focuses on its capability to handle dynamic participants. A system's organization defines the entities' roles, relationships, and authority structures that govern entities' behavior. Systems based on federation, market or social organizations support dynamic participants by maintaining information about entity membership. Federation and market organizations implement membership with a centralized distinguished entity which can either be statically specified or dynamically elected at runtime. The static centralized entity might be costly to install and maintain and the dynamic elected leader is more complex due to election, and the handing and taking over of responsibilities whenever the elected leader leaves the system. Moreover, both methods involve a central entity which might introduce a bottleneck and a centralized point of failure. Social organizations do not have distinguished entities; membership in social organizations is maintained by having entities go through joining and leaving steps. The joining step is implemented using two methods: freezing transactions or lurking. The freezing transactions method sacrifices the efficiency of every entity in the system whenever an entity enters, while the lurking method may result in an unbounded time before the entity may enter the system.

Comheolaíocht uses social organization with entities lurking before coordination. It assumes that membership is supplied by the underlying communication protocol (e.g., STEAM [Meier and Cahill, 2002], Vertigo [Slot et al., 2010]), allowing the developer to choose/implement different membership methods based on the system's requirements. In order to keep the connection between the coordination protocol (developed by Comheolaíocht) and the chosen membership protocol simple, this work only requires to know the identities of participating entities at a fixed instant instead of requiring membership information for the period of the transaction.

2.6.1.3 Applications and middleware

The previous sections describe the parameters and complexities in a real-time multi-entity coordination system. A summary of this classification is shown in Table 2.4. This table lists only the most significant systems in each community. In addition, these systems are ranked only according to the following criteria: support for dynamic participants, local coordination, safety constraints, liveness, and event ordering constraints. The first four criteria are rated using a number of stars, a system suitable for the problem that we are tackling needs to have a rating of two stars in each criteria; this corresponds to supporting dynamic participation, supporting local real-time coordination, and ensuring safety and goals constraints despite failures. The last criteria, event ordering constraints, represents support for different-time (dt), same-time (st), before (bf) and no-time (nr) event ordering. In addition, different-time event ordering can have a superscript dt^s to denote support for scheduling. This work supports the set of $\{dt^s, bf\}$ ordering constraints.

2.6.2 Influential work

Comheolaíocht is largely motivated by Comhordú [Bouroche, 2007]. Therefore, most of the concepts like responsibility, lurking time, sending area, safe mode and contract without feedback are either taken straight from Comhordú or adapted to this work. These concepts and their modifications will be introduced when used.

Besides Comhordú, some systems that influenced Comheolaíocht include intersection manager [Dresner, 2009], scheduling [Pinedo, 2008], auctions [Gerkey and Matarić, 2004, Dias et al., 2006, Zlot and Stentz, 2006] and fully-observerable systems [Schermerhorn and Scheutz, 2006, Elizondo-Leal et al., 2008].

The problem of autonomous vehicles crossing an intersection and its various requirements (e.g., vehicle breakdown, exiting intersection) are adapted from Dresner's work. In this work, especially during the implementation of specific requirements, Dresner's concept is used as a baseline for comparison.

Comheolaíocht models the different-time event ordering constraints as a mutual exclusion and scheduling problem. This work uses the characterization of scheduling problems by Pinedo [2008] for the specification and modeling of a multi-entity coordination problem.

System	Dyanmic partici-pants	Local co-ordination	Safety	Liveness	Event ordering constraints
Intelligent Transportation Systems					
Collision prevention platform for a dynamic group of asynchronous cooperative mobile robots. [Yared et al., 2007]	★★	★	★	★	*dt*
Autonomous Intersection Management [Dresner and Stone, 2006b] ... [Dresner, 2009]	★★	★	★★	–	*dt*
Other applications (RoboCup, exploration, MCM)					
CAMBADA [Azevedo et al., 2007, Lau et al., 2009]	–	★	-	-	dt, bf, nr
Multi-robot exploration and mapping using self-biddings [Elizondo-Leal et al., 2008]	★★	★	-	-	*nr*
DEMiR-CF [Sariel et al., 2006b]	-	-	★★	★★	bf, nr
DELPHÍS [Sotzing et al., 2007]	-	-	★★	★★	bf, nr
Coordination protocols					
A fault tolerant mutual exclusion algorithm for mobile ad hoc networks [Wu et al., 2008]	★	-	★★	★★	*dt*
Extending Paxos/LastVoting [Borran et al., 2008a]	-	-	★★	★★	*dt*
Local Mutual Exclusion [Attiya et al., 2010]	★	-	★★	★★	*dt*
Real-time coordination middleware					
LIME & extensions [Roman et al., 2006, Hackmann et al., 2005]	★	-	★	-	-
TBMAC, SEAR & Comhordú [Hughes, 2006, Bouroche, 2007]	★★	★★	★	-	dt, bf, nr
A middleware for cooperating mobile autonomous systems [Schemmer et al., 2001]	★★	★★	★	-	dt^s
TCB & Wormhole [Veríssimo and Casimiro, 2002, Veríssimo, 2006]	-	★★	★★	-	-

Legend:
DP: – not supported; ★ supported affect others (e.g., freeze everyone);
★★ supported do not affect others
Local coordination: – not supported; ★ supported non-realtime;★★ supported real-time
Safety, liveness: – not supported; ★ supported, assumes perfect communications or entity;
★★ supported, do not assumes perfect communication and entity
Event ordering constraints: - no information; *st* same time; *dt* different time with mutual exclusion, dt^s different time with scheduling; *bf* sequential; *nr* no time relation

Table 2.4: Comparison Summary

The idea of market-based organization [Horling and Lesser, 2004] and auctions [Gerkey and Matarić, 2004, Dias et al., 2006, Zlot and Stentz, 2006] is to use an utility value for determining the optimal allocation of tasks or resources. Comheolaíocht's CwoRIS pattern is adapted from Bouroche's [2007] contract without feedback protocol which allocates shared resources based on a first-come first-served basis. In addition, CwoRIS's usage of a score (utility) for breaking deadlocks, determining race winner, and preforming preemptions is inspired by auctions.

Various multi-robot coordination techniques assume that entities have access to every attributes in the system [Schermerhorn and Scheutz, 2006, Elizondo-Leal et al., 2008]. With some similarity, this work builds on the fact that entities can listen in on wireless communication and gather some understanding about the system's current state. In contrast to these work, Comheolaíocht does not assume that the information thus gathered is perfect due to imperfect communication.

2.7 Summary

This chapter first presented a taxonomy of multi-entity coordination and characterizes the complex problems, in particular, it investigates the difficulties in developing coordination protocols for scalable and reliable multi-entity systems. The chapter then reviewed work in multi-robot applications, generic coordination protocols and middleware. A number of criteria with regards to scalability and reliability are used to classify the work mentioned. This demonstrated that none of the existing work provides a generic protocol that addresses the challenges addressed in this thesis. Finally, we have shown how the design of Comheolaíocht was influenced by some of the work presented.

Chapter 3

System Modeling and Specification

System modeling is the first of Comheolaíocht's three steps in designing a solution for a coordination problem. This stage involves step-by-step modeling and specification of the system by describing the different participants and their behaviors, the environment, and the constraints between them. This step is designed to capture the relevant parameters and hide the unnecessary details.

A multi-entity system can be described by parameters along five axes: *entity*, *environment*, *relationship*, *objective* and *communication* (Section 2.1.2.1). Comheolaíocht supports the development of selfish, rational and honest entities that do not work in a team, are aware of other entities and have neutral intentions, which fixes all the parameters in the *relationship* axis. This system modeling and specification step captures parameters from the *entity*, *environment* and *objective* axes, while Comheolaíocht outputs a protocol for the *communication* axis.

The following section describes the distinction between entities and elements, in particular, it defines a developer's *sphere of influence* and further scopes this work to handle only entities that are within a developer sphere of influence. Section 3.2 then defines entities' behaviors and describes similar behaviors modeled as behavior types. Similarly, Section 3.3 describes similar entities modeled as entity types. Behavior types and entity types can be use to promote modularity by encapsulating the differences between similar entities and behaviors. This modeling captures parameters from the *entity* axis.

Section 3.4 then model a system environment by partitioning the physical space of the environment into non-overlapping scenarios and grouping similar scenarios into scenario types. For each scenario-type, entity coordination constraints and objectives are defined. This modeling captures parameters from the *environment* and the *objective* axes. The partitioned system provides a separation of concerns across the parts thereby allowing sub-systems to be developed in parallel. Such a partition makes the system design process less complex by considering only sub-problems with less interleaving constraints.

3.1 Entities and Passive Elements

An environment can be modeled as a collection of *elements* such that *passive elements* refer to elements that are not within the application developer's sphere of control and *entities* refer to elements within an application developer's sphere of control [Bouroche, 2007]. An application developer's sphere of control over an element can be defined by whether the developer can modify the element's behaviour. For instance a traffic light can perform actions (change lights) and has an actuator (displaying the lights). If a developer cannot make the traffic lights change color, the traffic light is modeled as a passive element. Since passive elements are not within the control of application developers, safety between passive elements will not be considered in this work. Coordination between entities and passive elements may be performed using a contract without transfer whereby an entity senses the presence of passive elements by indirect communication and gives way to the passive elements [Bouroche, 2007]. Using the contract without transfer implies that entities must give way to passive elements, putting the entities in a lower-priority position.

Consider a situation where application developers know an element's deterministic reaction to some stimulus, then the element's actions can be indirectly influenced by the generation of such a stimulus; this means that such an entity could be indirectly included in a developer's sphere of influence. For instance, if pedestrians will not cross a junction during a red light, an emergency vehicle entity could ensure that no pedestrians are crossing the junction by communicating with the traffic lights in advance, thereby changing all pedestrian crossing lights to red, and influencing all pedestrians not to cross the road. The knowledge of pedestrians' deterministic reactions allow the emergency vehicle to influence their behaviors (through a traffic light proxy and indirect communication). In this example, pedestrians are considered to be under the 'sphere of control' of application developers and are entities.

As such this work redefines a developer's *sphere of influence* to include those elements that: a developer can either i) directly control via their actions (which are not required to be deterministic) or ii) indirectly control via their deterministic reactions by providing the required stimulus for some reaction. In the event that the developer cannot be entirely sure whether an element's reaction is deterministic, it should be assumed that it is not. Continuing the pedestrian example, some pedestrians (e.g., in Ireland) may cross the road while the pedestrian light is red, meaning that not all entities of the pedestrian type (cf., Section 3.3) have deterministic reactions and therefore pedestrians should be modeled as passive elements.

The first step for the modeling a coordination system is to identify the passive elements and entities.

3.2 Behavior & Behavior Types

The second step in modeling a coordination system is to identify the entities' behaviors. Entities' behaviors and behavior types are subsequently analyzed (see Chapter 4: Model Analysis) for the entity's capability to act safely and achieve the system's goal. We defined a *behavior* as an *action/stimulus-reaction* or a series of actions/stimuli-reactions that an entity may perform to generate some *observable effects*. Two concepts need to be clarified in this definition, series of actions/stimulus-reactions and observable effects.

An entity may sense the environment to obtain some *stimuli* (i.e., sensor inputs, received messages) to which the entity *reacts* (i.e., via actuator or sending a message). A *stimulus-reaction* pair models an entity performing some action (i.e., reaction) when it detects some properties in the environment (i.e., stimulus). An example for a stimulus-reaction is when a vehicle senses an obstacle (stimulus), and changes it trajectory (reaction). A series of actions/stimuli-reactions can be used to model complex behaviors. For example, a vehicle performing an overtaking action has the stimuli of a leading slow vehicle and no vehicles on an adjacent lane, and the actions of vehicle changing lane and accelerating. Defining behavior using a series of actions/stimulus-reactions abstracts away the details of low-level actions.

Observable effects of a behavior refer to the outputs of actions that can be detected by other entities. Such observable effects may be in the form of receiving a message, changing the entity's physical state or changing the environment. The requirement for behavior to produce observable effects means that all actions that cannot be detected by other entities (e.g., sensing, deliberating, learning) are hidden and not considered as behavior. Defining behavior only via observable effects abstracts away all other actions that are not visible to other entities.

A *behavior type* defines the parameters that describe a behavior. Behaviors belonging to the same behavior type have the same set of stimuli and observable effects; two behaviors with different series of actions/stimuli-reactions belong to the same behavior type as long as their set of stimuli and observable effects are the same. This definition of behavior type groups seemingly heterogeneous behaviors under a single type based on their inputs (stimuli) and outputs (observable effects). The following sub-section describes the input and output of a behavior-type in detail and section 3.2.2 provides a summary of the parameters for specifying an entity's behavior.

3.2.1 Behavior inputs/outputs

The parameters used to differentiate between two behaviors are their inputs (stimuli) and outputs (observable effects). This section describes the inputs and outputs of a behavior type.

- Inputs (stimuli)

  The inputs of a behavior, also called the stimuli of a behavior, refer to conditions that make an entity perform that behavior. Behaviors are further classified as

  - Controllable: an entity can choose to perform the behavior.

    An example of a controllable behavior is an autonomous vehicle gets impatient and decides to perform an overtaking behavior.

  - Uncontrollable: the entity cannot choose to perform the behavior or not.

    An example of an uncontrollable behavior is when the autonomous vehicle breaks down.

  An entity with uncontrollable behaviors may perform unsafe actions. Section 4.2.6 analyses whether a model can be solved by Comheolaíocht by checking whether an entity's behaviors allow it to achieve the system goals while respecting the safety constraints.

- Outputs (observable effects)

  The output of a behavior is differentiated by its observable effects. If other entities are not able (or are not required) to differentiate between two sequence of observable effects, then any observing entities may perceive that these effects are generated by the same behavior (the entities need not be concerned that the effects are produced by different behaviors). Consequently, behaviors with non-deterministic observable effects, may be split into multiple behaviors depending on whether other entities need to differentiate between the effects. For example, if a robot may avoid an obstacle by taking either a left or right side step and if other entities need to differentiate whether the robot performed a left or right maneuver, then the obstacle avoidance behavior can be modeled by two behaviors: 'avoid obstacle left' and 'avoid obstacle right'.

  In addition, an entity could have an 'inaction' behavior with observable effects. An example is an autonomous vehicle parked on the road, while parking does not involve performing any actions, such inaction is observable to other autonomous vehicles. Therefore, parking can be modeled as a behavior with the observable effect of a stationary vehicle.

3.2.2 Behavior parameters

In summary, a behavior's parameters are its inputs and outputs. As this work is only concerned about how entities influence each others' behaviors, only observable effects are required to be recorded. The parameters describing a behavior are:

- Name: unique name for the behavior.
- Controllability: whether the behavior is controllable or uncontrollable.
- Stimulus: the condition that makes the entity perform behavior.
  - A behavior can be controllable or uncontrollable.
- Observable effects: what other entities can detect and are concerned about.

3.3 Entity types

Entity types group entities with the same set of behaviors (inputs-outputs) and abstract away the non-relevant parameters (e.g., different physical characteristics that may result in different robot control). This abstraction allows more entities to be grouped under the same entity type, thereby resulting in fewer entity types and lowering the system complexity by considering fewer types of different entities. For example, different makes and models of vehicles (e.g., trucks, buses, cars) might not be differentiated because all vehicle entities have the same behaviors.

This section describes the design of entity types in detail. The following sub-section first describes an entity-type's knowledge. Next, the parameters needed to specify an entity-type are presented.

3.3.1 Knowledge

An entity's knowledge can be described by the subject, state and model (Section 2.1.2).

An entity type's knowledge of its own model (i.e., subject = self) is captured by its set of behaviors. The entity type's knowledge of its own state is recorded by its set of local variables. These variables include the inputs of the entities' behaviors.

In this work, we assume that an entity knows that other entities will adhere to the same protocol developed by Comheolaíocht.

3.3.2 Entity-type parameters

In summary, the parameters to specify an entity-type are:

- Name: an unique name for an entity-type
- Set of behaviors
- Set of local state variables

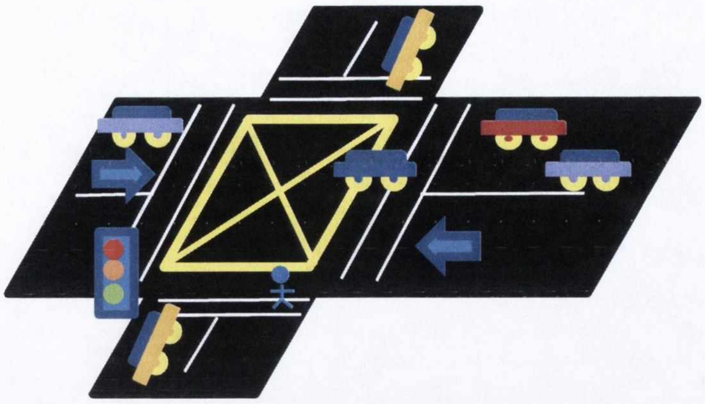


Figure 3.1: Intersection crossing

3.4 Environment, Scenario and Scenario Types

An *environment* is partitioned into multiple *scenarios* in order to provide separation of concerns and allow developers to work independently on the scenarios. It is expected that entities' constraints within each scenario are less complex when compared to tackling the system as a whole due to less interleaving of constraints.

A *scenario* is a partition (non-overlapping and non-empty) of the physical space in the environment, such that each scenario involves a minimal (non-zero) number of entity types and each entity type has a minimal (non-zero) number of applicable behaviors in the scenario. In addition, the set of scenario partitions are collectively exhaustive; the union of all partitions defines the whole environment. This section uses the intersection collision avoidance environment as an example (Figure 3.1). A scenario breakdown would first identify each of the pedestrian crossings as a stand alone scenario; firstly, because pedestrian entities and the vehicle entity's avoid-pedestrian behavior do not occur anywhere else in the environment and secondly, because a pedestrian crossing scenario does not have a smaller physical partition where a different set of entity types and behavior types is observed. The two other scenarios in this example are road and junction (see Section 6.3 for the modeling of the intersection collision avoidance scenario).

Scenario types group scenarios with the same set of entity types and behavior types together. For instance the four pedestrian crossings in the above example are of the same pedestrian-crossing scenario type. Each scenario type can be analyzed independently so as to achieve modularity and reusablility across the environment; interactions between scenarios are specified in the constraints (Section 3.4.5), and will be examined in the scenario composition step (Section 4.4).

The following sub-section provides an overview of the constraints for a scenario type.

3.4.1 Constraint overview

This thesis defines five types of constraints: scenario abstraction, scenario setting, entrance, safety and goal. Each of these constraints is related to some aspects of a scenario.

- Scenario abstraction

  Coordination problems with different time event ordering can be represented as a resource sharing problem. The scenario abstraction step maps scenario parameters (e.g., location) into resources. Section 3.4.2 presents the scenario abstraction step.

- Scenario setting

  Coordination problems with sequence event ordering (before/after) can be modeled in the scenario setting step. The scenario setting step records two properties: i) the order in which resources are used by the entities, and ii) the priorities between the entities. Section 3.4.3 presents the scenario setting step.

- Entrance (pre-condition)

  The entrance constraint may also be referred to as the pre-condition required to be satisfied when an entity joins a scenario. This constraint records two properties: i) the location where the entity enters the scenario, and ii) a constraint to ensure that an entity who has just entered a scenario is not in a behavior that will inevitably violate the safety constraint. For example, a vehicle entity that is about to enter a road scenario must have enough breaking space between itself and the last vehicle on the road.

- Safety (invariant)

  Safety constraints are introduced in Bouroche's thesis [Bouroche, 2007]. The safety constraint may also be referred to as the invariant of the scenario. It describes conditions that must be satisfied at all times; violation of the safety constraint may lead to the failure of the system. Section 3.4.4 presents the modeling of safety constraints.

- Goal (post-condition)

  The goal constraint may also be referred to as the post-condition of the scenario. It is the condition that must be satisfied just before an entity leaves the scenario. Section 3.4.5 presents the modeling of a scenario's entrance and goal constraints.

3.4.2 Scenario abstraction; different time constraints

This section presents the steps to abstract and specify coordination problems with different-time event ordering constraints (Section 1.3). The first subsection shows that such coordination problems can be modeled as resource sharing problems. This modeling provides two advantages: firstly, established algorithms for coordinating access to shared resources can be reused, and, secondly, the modeling provides a standard way to approach problems exhibiting similar constraints. Section 3.4.2.2 then defines the syntax for specifying an entity's use and reservation of resources and Section 3.4.2.3 presents the syntax and semantics for representing resource-usage conflicts; i.e., resource-usage violating the different-time property. Section 3.4.2.4 summarizes this section and presents the syntax for defining the different-time events ordering constraints.

3.4.2.1 Modeling different-time event ordering problems as resource sharing problems

Different-time event ordering constraints describe situations where events must not happen at the same time (Section 1.3). Problems with such constraints can be modeled as resource-sharing problems. In resource-sharing problems, there is a pool of shared resources that entities may use, and the entities must be ensured exclusive resource usage at any time [Ricart and Agrawala, 1981, Agrawal and Abbadi, 1991, Pinedo, 2008]. This exclusive resource usage property is similar to the different-time event ordering constraint - the only difference is the absence of shared resources in some problems with different-time event ordering constraints. For example, a commonly seen different time event ordering constraint is where robots must not be in the same place at the same time; i.e., space is a shared resource.

In essence, there are many ways to define a physical area. Similarly, problems with different-time event ordering could be mapped to a virtual space. For instance, in the robot-soccer scenario, in order to maintain a formation, different robots are required to take up different roles (e.g., left-striker, right-defender); these unique roles can be mapped as resources. The main idea is to map different-time event ordering constraints into resource sharing problems by identifying the 'space' for which entities must be ensured exclusive access, such 'space' could either be physical or a virtual abstraction with a unique name.

Note: Specific steps for modeling constraints into shared-resources in not in the scope of this work.

Multiple instances of resources One relaxation of the different-time event ordering constraint is the not-more-than-n condition specifying that not more than n events may happen at the same time. The corresponding resource specification is:

- Single instance of resource with identifier i, r_i

  Defines and specifies a single resource with its unique identifier.

- n multiple identical instances of resources sharing the identifier i, r_i^n
  - To specify a specific instance $r_{i \odot j}$
  - To specify more than one, specific instances $r_{i \odot [j,k,\ldots]}$
  - To specify any instance $r_{i \odot ?}$
  - To specify m (non-specific) instances $r_{i \odot ?m}$

  The definition of multiple instances of identical resources allows multiple entities to use some resources at the same time, or an entity to use more than one instance of the resource at the same time. For example, multiple vehicles may board a barge to cross a river, since the barge can carry a maximum capacity. The problem can be abstracted as the barge having multiple identical instances of resources and each vehicle must obtain such a resource in order to board the barge.

- n multiple near-identical instances sharing identifier i and differentiated by a parameter, $r_i^{n(\text{parameter})}$

  For specialized treatment of events, resources may carry some parameters to allow entities to reason about the characteristics of the resources they require.

  For instance, a road can be modeled as lanes, where each lane is nearly identical to another lane except for its location - which is used to define neighboring lanes. An abstraction for resources on a road can be specified as $r_i^{lanes(\text{laneNumber})}$, where *'i'* refers to the identity of the road, *lanes* specifies the number of lanes (resource units) on the road, and each resource has a parameter specified by laneNumber. When an autonomous vehicle travels on the road, it reserves and uses the road resources model. Using the laneNumber parameter, an autonomous vehicle's lane changing behavior could be specified that it requires a laneNumber that is to the left (-1) or to the right (+1) of the current lane.

3.4.2.2 Resource usage

The previous section shows that problems with different time event ordering constraints can be abstracted as shared resource problems. This section shows that event (i.e., entity performing an action, time passing, something happened in the environment) occurrence can also be abstracted as resource use.

- Resource use:$< r_i, t_s, t_e >$

  The use of a resource, or the occurrence of an event, can be described by the resource identifier (r_i), and the start (t_s) and end (t_e) time. The CwoRIS pattern (Section 5) can implement resource use as scheduling or mutual exclusion. Scheduling of resources requires a start and a end time to be specified whereas resource mutual exclusion just requires the resource use start time; the end time (t_e) for mutually exclusive resource use is not predefined.

- Basket of resources usage: $R = < r_i, t_s, t_e >, ...$

  The basket of resources abstraction has been used, for example in the literature for combinatorial auctions [Kutanoglu and Wu, 1999] where entities require a whole set of resources or none. Our work represents resources basket use as a set of resource use tuples. The resource basket specifies that the entity needs all the resources in the basket or none.

An entity may use a resource only if it holds exclusive rights to it. As mentioned in Section 2.1.2, Comheolaíocht coordinates by having entities send requests, therefore, an entity may request the right to use the resources it requires. There is no ambiguity when an entity requests or holds a resource of which there is only a single instance of the resource. In the case where there are multiple instances of a resource, an entity might want to request for any/many instances of the resource and to hold exclusive rights to some specific instances of the resource. The syntax for describing an entity requesting and holding exclusive rights to resources are:

- Resource requests:
  - Single instance of the resource: r_i
  - Multiple instances of resources, requesting for any single instance: $r_{i\odot?}$
  - Multiple instances of resources, requesting for m instances: $r_{i\odot?m}$
  - Multiple instances of resource differentiated by a parameter: $r_{i\odot?m}^{condition}$

  As an illustration, in the previous example where a road is specified as $r_i^{lanes(\text{LaneNumber})}$, a request for an adjacent lane could be $r_{i\odot?}^{\text{laneNumber=currentLane+1}\vee\text{currentLane}-1}$, where currentLane is the entity's current lane number.

- Resource usage (held):
  - Single instance of the resource: r_i
  - Multiple instances of resources, holding exclusive rights to a single instance identified by k: $r_{i\odot k}$
  - Multiple instances of resources, holding exclusive rights to multiple instances: $r_{i\odot[j,k,\ldots]}$

  Note that an entity only can hold specific instances of a resource, holding unspecified instances (i.e., $r_{i\odot ?m}$) is not defined.

3.4.2.3 Resource comparison

This section presents the syntax for comparing two baskets of resources. In particular, the conflict relation is used to check whether two resource sets violate the different-time condition by comparing whether they require the same resources at the same time. These definitions will be used in Chapter 5.

Two sets of resources, R_a and R_b, are in conflict when some resources exist in both sets and their usage time overlaps.

$$\begin{aligned} R_a \cap R_b \neq \emptyset := \quad & \exists \langle r_i, t_{s1}, t_{e1}\rangle \in R_a, \langle r_i, t_{s2}, t_{e2}\rangle \in R_b \bullet \\ & t_{s1} < t_{e2} \wedge \\ & t_{s2} < t_{e1} \end{aligned}$$

We further define the intersection between two sets of resources by: a resource reservation is in the intersection set, $R_a \cap R_b$, if the resource appears in both sets such that their usage times overlap. The resource reservation/use time of the resource in the intersection set is defined as the period for which their times overlap.

$$\begin{aligned} R_a \cap R_b = \quad & \{\langle r_i, \max(t_{s1}, t_{s2}), \min(t_{e1}, t_{e2})\rangle : \\ & \langle r_i, t_{s1}, t_{e1}\rangle \in R_a \wedge \\ & \langle r_i, t_{s2}, t_{e2}\rangle \in R_b \wedge \\ & t_{s1} < t_{e2} \wedge t_{s2} < t_{e1}\} \end{aligned}$$

Note that Chapter 5 only requires the definition for conflict comparisons between two resources ($R_a \cap R_b \neq \emptyset$); the intersection set ($R_a \cap R_b$) is presented because it will be used for defining other relations.

Multiple instances When there are multiple instances of resources, the comparison between two resource sets for conflicts is straight forward if both sets use specific instance. The conflict definition is extended to handle multiple instances of resources:

- Specific instance

  There is a conflict only if both sets refer to the same instance (i.e., $x.r_{i\odot j} = y.r_{i\odot j}$)

$$\begin{aligned} R_a \cap R_b \neq \emptyset := \quad & \exists \langle r_{i\odot j}, t_{s1}, t_{e1} \rangle \in R_a, \langle r_{i\odot j}, t_{s2}, t_{e2} \rangle \in R_b \bullet \\ & t_{s1} < t_{e2} \wedge \\ & t_{s2} < t_{e1} \end{aligned}$$

  Comparisons between two resource baskets to detect conflicts when either set specifies *any* resource instance ($r_{i\odot ?}$ and $r_{i\odot ?m}$) is not defined.

- Any one

  In order to define the conflict relation for resources involving non-specific instances, it is necessary to define some supporting constructs:

  - Intersection between two sets of resource tuples

$$\begin{aligned} R_a \cap R_b = \quad & \{\langle r_{i\odot j}, \max(t_{s1}, t_{s2}), \min(t_{e1}, t_{e2}) \rangle : \\ & (\langle r_{i\odot ?}, t_{s1}, t_{e1} \rangle \in R_a \vee \langle r_{i\odot j}, t_{s1}, t_{e1} \rangle \in R_a) \wedge \\ & (\langle r_{i\odot ?}, t_{s2}, t_{e2} \rangle \in R_b \vee \langle r_{i\odot j}, t_{s2}, t_{e2} \rangle \in R_b) \wedge \\ & t_{s1} < t_{e2} \wedge t_{s2} < t_{e1} \end{aligned}$$

    The above definition extends the previous definition of the intersection between two sets of resources, R_a and R_b. In this definition, both R_a and R_b use the non-specific (any-one) resource specification ($r_{i\odot ?}$). Note: in the above definition, when both resources are specified as 'any one' instances, ($r_{i\odot ?}$), the result of the intersection is also an 'any-one' specification.

  - Intersection with the whole system

    The set of all reservations/uses of resources in the whole system is denoted by the universe ($\mathcal{U}$). The intersection of a resource set with the universe is defined as the union of all pair-wise intersections of R_a with every reservation/use in the universe. $R_a \cap \mathcal{U} = \cup_{(\forall R_b \in \mathcal{U}, R_a \cap R_b)}$

  - Union between two sets

    The union between two sets of resource reservations, R_a and R_b, is defined by: if a resource, $r_{i\odot j}$, is in one set but not in the other set, then resource $r_{i\odot j}$ with its original reservation time is in the union, moreover, if resource $r_{i\odot j}$ is in both sets, then the union uses the earlier start time and later end time (effectively filling any gaps between the use times).

Let $a = \langle r_{i\odot j}, t_{s1}, t_{e1} \rangle, b = \langle r_{i\odot j}, t_{s2}, t_{e2} \rangle, a \oplus b = \langle r_{i\odot j}, \min(t_{s1}, t_{s2}), \max(t_{e1}, t_{e2}) \rangle$ in

$$\begin{aligned} R_a \cup R_b = \quad & \{c : \\ & (a \in R_a \wedge b \notin R_b \implies c = a) \wedge \\ & (a \notin R_a \wedge b \in R_b \implies c = b) \wedge \\ & (a \in R_a \wedge b \in R_b \implies c = a \oplus b)\} \end{aligned}$$

- Resource request conflict (any single copy)

  Conflict: $\exists r_{i\odot ?} \in R_a, |\{r_{i\odot 1}, ..., r_{i\odot n}\} \cap \mathcal{U}| \geq n$

  The definition can be read as: if there exist a resource, $r_{i\odot ?}$ in request R_a, that requires any instance of the resource to be available, such that every copy of the resource has already been taken up, $|\{r_{i\odot 1}, ..., r_{i\odot n}\} \cap \mathcal{U}| \geq n$, then the request is in conflict.

- Resource request conflict (any multiple copies)

  Conflict: $\exists r_{i\odot ?m} \in R_a, |\{r_{i\odot 1}, ..., r_{i\odot n}\} \cap \mathcal{U}| > n - m$

  When more than one instance of a resource is required, the universe must have at least the required number of resources available; a conflict happens when there are not enough resources available.

• Parametrized instance

The request for a parametrized instance, $r_{i\odot ?m}^{\text{condition}}$, includes a condition to be matched against the parameter in the specification, $r_i^{n(\text{parameter})}$. Requests for a parametrized instance are required only when requesting *any* instances; request for specific instances do not need to check the condition parameter.

Conflict: Let $R_i = \{r : r^{\text{para}_r} \in \{r_{i\odot 1}^{\text{para}_1}, ..., r_{i\odot n}^{\text{para}_n}\} \wedge \text{condition}(\text{para}_r)\}$ in

$$\exists r_{i\odot ?}^{\text{condition}} \in R_a, |R_i \cap \mathcal{U}| \geq |R_i|$$

Where *condition(para)* refers to the request's condition applied to the parameter, which evaluates to true when a resource with the parameter can be a match for the request. R_i represents the set of resources in the universe that satisfy the condition. $|R_i|$ counts the number of possible matches and $|R_i \cap \mathcal{U}|$ counts the number of matched resources reserved in the system. The definition can be read as: if there exist a request R_a for *any* resource with a condition, $r_{i\odot ?}^{\text{condition}}$, such that every instance of the resource that satisfies the condition (R_i) has already been taken up, $|R_i \cap \mathcal{U}| \geq |R_i|$, then the request is in conflict.

This definition is an adaption from the resource request conflict (any single copy) definition. The parametrized copy definition checks whether there is an available copy in the universe. A conflict is registered when all the resources that satisfy the condition are reserved or when there are no resources that satisfy the condition (for which case $|R_i \cap \mathcal{U}| = 0$ and $|R_i| = 0$).

3.4.2.4 Syntax

This section presents the syntax for scenario abstraction using EBNF [ISO, 1996].

A scenario abstraction is a representation for the shared resources and any constraints on the use of the resources.

$$\text{scenario\_abstraction} := \text{resource\_definition\_list}, ";", \text{resource\_constraint\_list}$$

$$\text{resource\_definition\_list} := \text{resource\_definition}, ";", \text{resource\_definition\_list} | "NULL"$$

$$\begin{aligned}\text{resource\_definition} := \quad & \text{resource\_name}, " = ", \text{some\_resource} \\ & | \ \text{resource\_name}^{n}, " = ", \text{some\_resources} \\ & | \ \text{resource\_name}^{n(\text{parameter})}, " = ", \text{some\_resources} \\ & | \ "NULL"\end{aligned}$$

$$\text{resource\_constraint\_list} = \text{resource\_constraint}, ";", \text{resource\_constraint\_list} | "NULL"$$

$$\text{resource\_constraint} := \quad \text{resource\_name}, " \cap ", \text{resource\_name}$$

The resource constraint list is used to define static conflicts among the resources. Static resource conflicts are used to specify that two resource names refer to the same physical resource. Such specifications are useful for hierarchical resource abstraction; for instance, a junction can be modeled as a grid, set of lanes or as a single resource (Figure 3.2). A hierarchical abstraction includes all three abstractions and the static resource conflict records that lane, grid and the single junction resources refer to the same resource. An entity may use any of the lanes or grid-square abstractions to specify a resource, since some grid square and lanes refer to the same space (resource), a vehicle using a lane must have exclusive use of both the lane and the corresponding grid square.

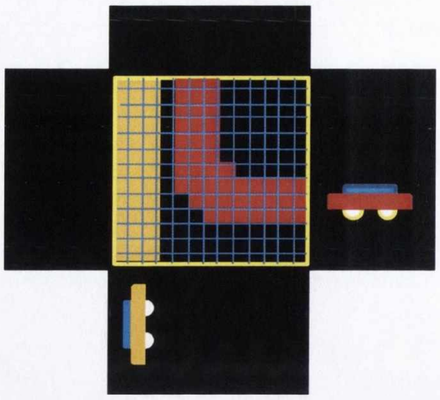


Figure 3.2: Intersection resource abstraction: grid and lanes

3.4.3 Scenario setting; sequence constraint

While the previous section presented the modeling and specification of different-time event ordering constraints, this section presents the modeling and specification of sequence event ordering constraints.

The following subsection describes the modeling of the sequence coordination problems and defines the basic constructs for specifying sequence constraints. Section 3.4.3.2 describes constraints that limit the use of resources in particular sequences and links these sequence definitions to the traditional scheduling problem classification [Pinedo, 2008]. Section 3.4.3.3 then presents the syntax for defining the scenario setting and summarizes this section.

3.4.3.1 Sequence constraints

Coordination problems with sequence event ordering constraints have constraints on events happening one after another. This section presents a method for modeling the problem, and describes the scope of this work.

Section 3.4.2 presented a model that uses resources to represent different-time constraints, this section reuses the same resource abstraction; an event is abstracted as an entity request use of resources. Using this abstraction, sequence constraints can be classified under three categories:

1. Same entity, different resources

   This category groups problems where an entity must perform some actions (use different resources) in sequence. An example problem mapping to this abstraction is the specification for an autonomous vehicle to drive in a single direction on the road; each end of the road is represented by a resource and the resource usage sequence constraint limits vehicles to use the resources in sequence (i.e., they must drive from one end to the other end).

$r_i \blacktriangleright r_j \implies (\langle r_j, t_{sj}, t_{ej} \rangle \in R \implies \langle r_i, t_{si}, t_{ei} \rangle \in R \wedge t_{si} < t_{sj})$

Using the $\blacktriangleright$ operator, the relation above defines the sequence constraint "same entity different resources". In the relation, the resource usage of a single entity is captured by the resource basket R. It specifies that: if the resource, r_j, is being used by an entity, then the entity must also use r_i and it must start using r_i before r_j (as denoted by $t_{si} < t_{sj}$). Note: the sequence constraint only limits the start time; the times for resources use may overlap.

- Chain; $r_i \blacktriangleright r_j \blacktriangleright r_k$

  The sequence operator may be chained; in this case event r_k cannot start unless both r_i and r_j have started, and, r_j cannot start unless r_i has started.

2. Different entities, same resource

   This category groups problems where different entities must perform the same action in sequence; that is, an entity must perform an action first before another entity may perform the same action.

   Dependency Comheolaíocht defines that two entites, e_1 and e_2, has a *dependency* constraint (i.e., e_1 *depends* on e_2), when e_1 can only use some resources r_1 after e_2 completed using its resources r_2.

   - These two entities is said to have a *natural dependency* when $r_1 = r_2$.
   - An entity is said to have *no dependency constraint* if it is not dependent on any entity.

   An example for this constraint is where a vehicle e_1 must allow another vehicle e_2 in front of it, on the same lane, to move away first, (i.e., e_1 is dependent on e_2).

   Dependency may be handled in Comheolaíocht using priority and a preemption constraint. An entity's priority is defined for each entity based on a developer's understanding of which events are more important. Priority in Comheolaíocht can be use for preemption, handling races (Chapter 5) and the prevention of starvation (Chapter 6). Note: defining which entity should have higher priority is not in the scope of this work.

   Comheolaíocht maps the scheduling of entities and resources to the scheduling of jobs and machines [Pinedo, 2008]; (i.e., jobs as entities and resources as machines). In machine-job scheduling, preemption implies that the scheduler is allowed to interrupt the processing of a job (preempt) at any point in time and put a different job on the machine instead. There are two main differences in resource-entity scheduling: the definition of when an entity starts using a resource and the absence of the scheduler. We define preemption in resource-entity scheduling as:

Preemption implies that it is not necessary that an entity completes its resource use even though it has acquired exclusive access to it. A higher-priority entity may request (preempt) the resources, and the lower priority entity must give up the resource.

In the definition, an entity *completing* using its resources means that it does not require to use the resource anymore; being preempted means that it has to give way to the higher-priority entity by releasing the resources before the end of its usage.

Since traditional scheduling systems are centralized, preemption was denoted as a system constraint; that is, if the system allows preemption, it is assumed that every machine is capable of handling the preemption of a job by another. Comheolaíocht uses resources as an abstraction for roles, tasks and actions. Resources may be different and distributed, as such, an entity may only preempt some resources (i.e., other resources cannot be preempted). Following traditional scheduling, this methodology uses the term *prmp* to means preemption; the syntax is extended to allow the specification of the subset of resources supporting preemption;

prmp$(r_i, ..., r_k)$ denotes that the resources in the set $r_i, ..., r_k$ allow entities with higher priority to preempt usage by another entity.

3.4.3.2 Resource usage order

A scheduling problem can be described using three parameters: $\alpha|\beta|\gamma$ [Pinedo, 2008]. The α parameter describes the resource usage environment and contains just one entry. This section describes Comheolaíocht's adaption to the resource usage (α parameter) and links our representation of resource use ordering with traditional scheduling. The β field provides details of processing characteristics and constraints and may contain zero to multiple entries. This section also describes the constraints (β parameter) that can be used in distributed entity coordination and provides some examples of usage. The γ field describes the objective to be minimized; γ optimizations are not supported in this work.

α-constraint In traditional scheduling, the α constraint is used to specify the resources and their usage sequence [Pinedo, 2008]. This section shows the modeling of three of the α constraints: the flow shop, flexible flow shop and job shop. Although the other α constraints (e.g., flexible job-shop, machines in parallel with different speeds) can also be modeled, however, they are removed from the scope of this thesis as there are no ready examples to demonstrate their application.

- Flow shop: F^m

  There are m resources and each entity has to use the m resources in sequence. Let's assume that the identity of the resources are 1...m; the sequence of resource usage for each entity is $r_1 \blacktriangleright r_2 \blacktriangleright \cdots \blacktriangleright r_m$

  An example of the flow-shop abstraction is autonomous vehicles traveling on a single-lane road. A single-lane road can be modeled as a flow-shop by dividing it into segments such that each segment is represented by a resource, an autonomous vehicle must travel on every segment (use every resource) in sequence.

  Entities in the flow-shop has a dependency constraint such that an entity holding resource r_i is dependent on an entity holding r_j if $i < j$.

- Flexible flow shop: $FF^c :: [r_1^{n1}, r_2^{n2}, \cdots, r_c^{nc}]$

  The flexible flow shop is a generalization of the flow shop. There are c stages in sequence, each stage has n_i identical resources; each entity follows the usage sequence of $r_{1\odot?} \blacktriangleright r_{2\odot?} \blacktriangleright \cdots \blacktriangleright r_{c\odot?}$ (i.e., each entity uses one of the identical instances of resources in each stage in sequence).

  Similar to the single-lane road's flow-shop abstraction, a multiple-lane road can be modeled as a flexible flow-shop. On the multiple-lane road, each lane is divided into segments where segments on adjacent lanes belong to resources in the same stage. An autonomous vehicle needs to travel from one stage to another in sequence in order to traverse the road. In addition, the vehicle may change lane (use an adjacent segment of the lane) while traveling on the road.

- Job-shop; J^m

  There are m unique resources in the scenario and each entity has its own predetermined sequence of resources to use. This work defines the job-class sequence constraint whereby entities performing the same job-class use the same predetermined resource sequence.

  - job-class sequence constraint: $J_i = (r_a \blacktriangleright r_b \blacktriangleright \cdots)$; $1 \leq a, b \leq m$
  - Set of all job-class sequence constraints: $J = \{J_p(r_a \blacktriangleright r_b \blacktriangleright \cdots), \cdots, J_q(r_c \blacktriangleright r_d \blacktriangleright \cdots)\}$,$1 \leq a, b, c, d \leq m$

  An example of a scenario that can be modeled with the job-shop is the intersection collision avoidance scenario. In the scenario, vehicles crossing from the same (direction, lane) source to the same (direction, lane) destination belong to the same job-class and use the same set of resources in sequence (Section 6.3 illustrates the modeling of the junction scenario as a job-shop).

β-constraints The β-constraints on a scenario are used to describe its resource usage. This section describes these constraints and their representations.

- Sequence dependent and job-family setup time; $R_a \cap_{\text{setup}} R_b$

  There are times where a buffer must be introduced between entities accessing same resource. In traditional machine-job scheduling, the setup time provides the necessary time for preparing a machine to accept the next job; in the resource abstraction, setup time may also be used to provide buffers between resource access. For instance, inter-vehicular distance must be kept between traveling vehicles so that vehicles can react in time in the event of breakdowns and errors in positioning and control (Section 5.4.3.6 illustrates the use of setup times for collision prevention in the event of breakdowns). In an abstraction, the inter-vehicular distance can be converted into time (i.e., using the vehicle's speed) and setup time constraints can be utilized for describing inter-vehicular distance.

  The **sequence dependent setup time**, s_{xy}, is the time required to prepare a resource between the processing of entities x and y. If the setup time between entities x and y depends on the resource r, then the subscript is included, i.e., s_{rxy}.

  s_{rxy} := setup time required for entity y to access resource r, after entity x has accessed r.

  In **job-family setup time** constraints, entities in the system belong to one of the F different job families. Entities from the same family may use a resource one after another without requiring any setup time in between. However, if the entity switches over from one family to another, say from family g to family h, then a setup, s_{gh}, is required. If the setup time depends on the resource, then the subscript is included, i.e., s_{igh}.

  $$\begin{aligned} R_a \cap_{\text{setup}} R_b \neq \emptyset := \quad & \exists x \in R_a, y \in R_b \bullet \\ & x.r = y.r \wedge \\ & (x.t_s - s_{ryx}) < y.t_e \wedge \\ & (y.t_s - s_{rxy}) < x.t_e \end{aligned}$$

  Conflict constraints (Section 3.4.2.3) describe the condition that entities cannot access resources at the same time. Setup-time conflict constraints extend the conflict constraint by including the required setup time in the calculation (Section 5.4.3.6 presents the use of job-family setup times to allocate allowances for vehicle breakdowns).

- Preemption; $prmp(r_i, ..., r_k)$, $prmp(R)$

  Preemption allows an entity with higher priority to replace another lower priority entity in the use of some resource. This β-constraint was presented in the different-entities-same-resource abstraction. (Section 3.4.3.1).

- Blocking; $block(r_i, ..., r_k)$, $block(R)$

  When the blocking constraint is present, an entity who has finished using the resources may still hold the resource, thereby preventing (i.e., blocking) another entity from using the same resource. A road leading into the junction is an example of a blocking resource: when a front vehicle cannot cross the junction, it has to wait on the road, thereby preventing (blocking) other vehicles behind it from using the resource it occupies (the segment of the road it is on) and also preventing other vehicles from accessing the junction.

- No-wait; *nowait*

  A schedule exhibits the no-wait requirement when an entity is not allowed to wait between access to two successive resources. Using the intersection crossing scenario as an example, a vehicle is not allowed to stop between the access to two grid squares (resources) within the junction (i.e., a vehicle must always be on at least on of the squares).

- Breakdown; *brkdown*

  Resource breakdowns imply that a resource may not be continuously available. The breakdown abstraction can be used to denote the fact that resources may become unavailable for some reason. For instance, in the intelligent transportation system, a broken vehicle may exceeds its allocated resource usage, resulting in the road not being accessible to other vehicles. The unavailable segment of the road can be abstracted as a resource breakdown.

This section presented the α and β constraints of traditional scheduling systems and adapted them to distributed entity coordination. The section also presented some examples of entity coordination using resource abstractions with various constraints. Since Comheolaíocht only provides access to resources on a first-come/first-served basis, this section did not present any γ-optimization parameters.

3.4.3.3 Syntax

This section presents the syntax to model the α and β constraints.

$$\text{setting\_constraint} := \alpha\text{usage}, \beta\text{constraint\_list}$$

The scenario setting constraint consists of two parts, the α usage sequence and the list of β usage constraints.

$$\begin{aligned}\alpha\text{usage} := \quad & F^m \\ & \mid FF^c :: [r_1^{n1}, r_2^{n2}, \cdots, r_c^{nc}] \\ & \mid J^m, \text{job\_constraint\_list}\end{aligned}$$

The αusage constraint could be flow shop, flexible flow shop or job shop. As mentioned, the syntax may also be used to specify other α constraints, but they are not in the scope of this thesis.

$$\begin{aligned}\text{job\_constraint\_list} := \quad & \text{job\_name} = (\text{resource\_sequence}), \text{job\_class\_constraint\_list} \\ & \mid \text{job\_name} = (\text{resource\_sequence})\end{aligned}$$

Note that the job-shop constraint list must have at least one job constraint.

$$\begin{aligned}\text{resource\_sequence} := \quad & \text{resource\_name} \blacktriangleright \text{resource\_sequence} \\ & \mid \text{resource\_name}\end{aligned}$$

The resource sequence can specify more than one resource in a chain, there must be at least one resource in a sequence.

$$\beta\text{constraint\_list} := \beta\text{constraint}, \beta\text{constraint\_list} \mid NULL$$

$$\begin{aligned}\beta\text{constraint} := \quad & \cap_{\text{setup}}[:: \text{family\_name}, ...] \mid \\ & prmp(r_i, ..., r_k) \mid \\ & block(r_i, ..., r_k) \mid \\ & nowait(r_i, ..., r_k) \mid \\ & brkdown(r_i, ..., r_k)\end{aligned}$$

The βconstraint list is a potentially empty list of constraints applicable to the scenario. The setup time βconstraint may include the list of family names needed to specify job-family setup time. The actual time allocated for the sequence dependent setup time may be implemented in different ways (e.g., a function based on the resource and entity parameter) and will not be recorded in the specification. The preemption, block, no wait and breakdown constraints are applied to some set of resources, (e.g., if the *nowait* constraint is specified, entities must release the resources after their use is completed).

3.4.4 Safety constraint

The safety constraint specifies a condition that must be satisfied at all times in the scenario; the safety constraint is the equivalent of a scenario's invariant. Some examples of safety constraints are: autonomous vehicles do not travel too close to each other on the road, vehicles do not cross an intersection on a collision path at the same time and (at a higher-level of description) vehicles do not collide.

The following sub-section presents Bouroche's definition of a safety constraint [Bouroche, 2007] and how these constraints can be specified in Comheolaíocht. The next sub-section then presents the syntax for Comheolaíocht's safety constraints.

3.4.4.1 Bouroche's safety constraint

In Bouroche's thesis, a safety constraint can be defined as [Bouroche, 2007]:

$$\begin{aligned}
\text{incompatibility} \;\; &:= (\text{incompatibility}, \text{"} \wedge \text{"}, \text{incompatibility}) \\
&\mid (\text{incompatibility}, \text{"} \vee \text{"incompatibility}) \\
&\mid (\text{entity-type}, \text{"."}, \text{state-variable}, \text{rel-operator}, \text{value}) \\
&\mid (\text{entity-type}, \text{"."}, \text{state-variable}, \text{rel-operator}, \text{entity-type}, \text{"."}, \text{state-variable}) \\
&\mid (\text{"distance(",position,",",position,")", rel-operator, value}) \\
&\mid (\text{"|", entity-type,".",state-variable, rel-operator, value,"|",rel-operator, value})
\end{aligned}$$

$$\text{rel-operator} = \text{"} < \text{"} \mid \text{"} \leq \text{"} \mid \text{"} > \text{"} \mid \text{"} \geq \text{"} \mid \text{"} = \text{"} \mid \text{"} \neq \text{"}$$

In Comheolaíocht, the distance() function and cardinality function, |...|, are replaced by other abstractions. For instance, maintaining a distance between two entities can be abstracted as the time difference between two entities accessing a resource (setup-time). The use of time in place of separation difference provides two advantages: firstly, distance can be distorted by non-straight roads and secondly, time incorporates the difference between vehicles' speed (i.e., faster vehicles with the same time-separation result in a larger distance separation, which is consistent with vehicles' reaction). The cardinality operator is replaced by the number of resources, in particular, 'value' in the cardinality constraint can be mapped directly to a resource quantity in r^{value}.

3.4.4.2 Comheolaíocht's safety constraint

A scenario's safety constraint is defined as a conjunction/disjunction of resource usage constraints and other entity conditions.

$$\begin{aligned}\text{safety\_constraint} := \; & (\text{safety\_constraint}, \text{"} \wedge \text{"}, \text{safety\_constraint}) \\ & \mid (\text{safety\_constraint}, \text{"} \vee \text{"}, \text{safety\_constraint}) \\ & \mid \text{resource\_usage} \\ & \mid \text{entity\_condition}\end{aligned}$$

The 'entity-condition' is either a limitation of the values in state variables or entity behaviors. Note that in the syntax, entity-type.entity-behavior_ID is short hand for (entity-type.behavior_ID=entity-behavior_ID) and ¬entity-type.entity-behavior_ID is short hand for (entity-type.behavior_ID≠entity-behavior_ID).

$$\begin{aligned}\text{entity\_condition} := \; & \text{entity-type}, \text{"."}, \text{state-variable}, \text{rel-operator}, \text{value} \\ & \mid \text{entity-type}, \text{"."}, \text{entity-behavior\_ID} \\ & \mid \neg\text{entity-type}, \text{"."}, \text{entity-behavior\_ID}\end{aligned}$$

The resource usage constraint links entity's state and behavior with resource usage.

$$\text{resource\_usage} := \text{entity\_condition} \Rightarrow holds(\text{resource\_name}_{id[\odot copy]})$$

While the scenario abstraction and scenario setting constraints describe the constraints regarding resources usage, the resource usage syntax above specifies that entities are responsible for acquiring the relevant resources in order to perform some conditions. Note that acquiring a resource does not imply that the entity must fulfill the specific set of conditions; therefore, the imply ($\Rightarrow$) operator is used instead of the equality ($=$) operator. For example, a vehicle performing the crossing junction behavior implies that it is holding the resources for crossing the junction, but a vehicle holding on to the resources is not required to be crossing the junction.

For brevity, time is omitted in the specification; the actual, longer form for the resource usage syntax should be:

$$\begin{aligned}\text{resource\_usage\_long} := \; & \text{entity\_condition} \wedge time = t_0 \Rightarrow \\ & holds(<\text{resource\_name}_{id[\odot copy]}, t_s, t_e >) \wedge t_s \leq t_0 \leq t_e\end{aligned}$$

The longer form of the constraint captures the fact that an entity can only execute some behavior (at time t_0) only if it is holding the relevant resource and the resource has been allocated to the entity for a period that includes t_0.

An example of a safety constraint definition for the intersection collision avoidance scenario is:

$$(\text{vehicle.crossing} \implies holds(r_{grid})) \lor (\text{vehicle.breakdown} \land (\text{vehicle.location is in intersection}))$$

The above safety constraint states that vehicles can execute the behavior crossing only if it holds the relevant resources, a vehicle may however be in the intersection without holding the resources if it is broken down. The condition to specify vehicle's location is in the junction can be expressed as a conjunction of inequalities on the x and y position values.

3.4.5 Entrance & goal constraints

The entrance and goal constraints form the pre- and post-conditions of a scenario respectively. The entrance constraints must be satisfied when an entity physically enters a scenario; the goal constraint must be satisfied just before the entity physically leaves the scenario.

Note that since the safety constraint (the invariant) must be true at all times; the safety constraint is naturally included in both the entrance and goal constraints; therefore, there is no need to reiterate conditions of the safety constraint in the entrance and goal constraints. Entrance and goal constraints are specified in order to expose the requirements (entrance constraints) and commitments (goal constraints) during the scenario composition step (see Section 4.4).

This section presents an example of exiting a junction as a motivation for entrance and goal constraints (Section 3.4.5.1) before presenting the syntax for the two constraints (Section 3.4.5.2 and Section 3.4.5.3).

3.4.5.1 Exiting a junction example

A vehicle exiting an intersection and driving on to a road may involve a number of potential threats:

1. If there is no space on the adjoining road for the crossing vehicle to leave the intersection, a crossing vehicle has to stop in the intersection, thereby violating the no-wait constraint (as introduced in Section 3.4.3.2). This situation is commonly seen on busy intersections where a driver either disregards the yellow box or miscalculates the speed of vehicles on the exiting road. Vehicles forced to stopped in the intersection lower the intersection's throughput as it prevents other vehicles from crossing. In the worst case, vehicles stopped in the intersection could be involved in accidents.

2. When a vehicle exits the junction too fast, its speed when entering the adjoining road could be much higher than the speed of the previous vehicle on the road. A collision may occur if the newly arrived vehicle cannot decelerate in time to maintain a safe inter-vehicular distance.

3. A vehicle that changes lane just after it crosses the junction (e.g., it needs to turn a corner on the next junction) could crash into another vehicle exiting the junction.

Dresner addresses the problem of vehicles exiting the junction with an admission control zone (ACZ), managed by the intersection manager installed at each intersection [Dresner, 2009]. Vehicles exiting the junction (and entering the ACZ) are required to reserve a space proportional to their length on the lane in the ACZ. Each lane in the ACZ has a limited fixed capacity and admissions are stopped when the capacity is exhausted. A vehicle changing lanes or leaving is required to make its request to the intersection manager.

Comheolaíocht views the exit-intersection problem as a requirement on vehicles entering a road; not as a problem of vehicles exiting a junction. It is observed that the three problems stated above are common to vehicles entering a road (i.e., whether from exiting a highway or a round-about). Using this view, the road scenario must have a pre-condition that specifies:

1. Vehicles must ensure that there is enough space before they are committed to entering the road.
2. Vehicles must ensure that they do not enter the road (exit the junction) too fast - the speed at which vehicles enter the road must not be greater than a proportion of the speed and distance to the vehicle in front.

This constraint on vehicles entering a road describes the same constraints that vehicles exiting a junction (and thus entering the road); that both the no-space (first problem) and the exit-too-fast (second problem) problems do not occur. The third problem of vehicles changing lanes is a problem intrinsic to the road scenario, and should not be exposed to other scenarios. This distribution of responsibilities provides the advantage that scenarios (in this case the road scenario) can be reused in other situations (e.g., exiting a roundabout), thereby providing better modularity.

3.4.5.2 Entrance (pre-condition)

Syntactically,

entrance_constraint := safety_constraint

Syntax wise, the entrance constraint is the same as the safety constraint; the difference is only in the semantics. An entrance constraint should record two pieces of information. Firstly, the possible locations and states of an entity when it physically enters the scenario should be specified; for example, the entrance location constraint for a road might capture the fact that vehicles only enter the road from the incoming end. Secondly, the entrance constraint should specify the conditions for entities entering the scenario to be compatible with i) entities that are already in the scenario and ii) entities who might be entering the scenario at the same time.

3.4.5.3 Goal (post-condition)

Syntactically,

goal_constraint := entity_condition

Using the 'entity_condition' syntax, the goal constraint records the location/state of entities just before leaving the scenario. No other conditions are recorded since entities are not required to coordinate to leave a scenario. Note that, entities may be required to coordinate before entering the next scenario and therefore may not leave the current scenario until they have done so. Section 4.4 presents these cross scenario issues in more detail.

3.5 Summary

This section presented a step-by-step guide to specifying a system of distributed entities. System modeling is designed to handle heterogeneous entities in a coordination system: the behavior-type, and entity-type grouping focuses on recording only relevant parameters, thereby lowering the number of heterogeneous types. The scenario-type partition of physical space provides a separation of concerns between the physical spaces. In addition, each scenario-type abstracts entity coordination constraints into resources. The separation of concerns and the common abstraction on entity coordination constraints provides less intertwined constraints during the development of a heterogeneous entity system.

As discussed, the steps for the modeling and specification of a coordination system are

1. Identifying the passive elements and entities.
2. Identifying the entities' behaviors. Only behaviors that produce observable effects are recorded.
3. Grouping behaviors into behavior types.
4. Grouping entities into entity types.
5. Partitioning the environment into scenarios and grouping similar scenarios into scenario types. For each scenario type:

(a) Scenario abstraction models the different-time event ordering constraints as resources, and to specify that entities do not access some resources at the same time.

(b) Scenario setting models the sequence event ordering constraint. Specification of the sequence event ordering constraint reuses specifications from traditional scheduling; the α constraint is used to specify a resource usage sequence and the β constraint is used to specify characteristics in resource usage like preemption, setup time and nowait.

(c) The safety constraint specifies the invariant of a scenario; a condition that must be satisfied at all times.

(d) The entrance constraint specifies the condition of an entity just before it physically enters the scenario.

(e) The goal constraint specifies the condition of an entity just before it physically leaves the scenario.

Chapter 4

System Analysis

Comheolaíocht's second stage, *system analysis*, analyses the specifications captured in the first step to obtain two results: firstly, it determines whether Comheolaíocht can provide a reliable solution to the specified multi-entity coordination system, and secondly, if a reliable solution exists, the step provides a coordination strategy that ensures the safety in the system.

The following section provides an overview of the steps in system analysis. Section 4.2 introduces the concept of modes, and provides some guidelines for the design of modes. Section 4.3 then presents some methods for reasoning about the evolution of entities' states, analyzing whether the scenario is solvable in Comheolaíocht, and deriving a coordination strategy. Finally, Section 4.4 presents methods for combining individual scenarios into the application environment by defining the entities behavior for transiting between scenarios.

4.1 Overview

Modes were first introduced by Bouroche [2007]. In Bouroche's thesis, modes serve the purpose of reasoning about the evolution of entities' states, and are used to analyse entities' safety throughout the evolution. While Bouroche defines modes as representing an entity's possible actions (actuation, sensing, sending messages or signals, processing), there is no mention of how to represent different entity's actions as modes.

The following section formally defines modes in Comheolaíocht; it provides a step-by-step guide for i) defining modes in a multi-entity system, and ii) defining mode transitions for analysing entities' behaviors. The section extends Bouroche's modes to handle entities' inactions that might lead to safety constraints violation and errors in entities' perception. Bouroche's mode transitions are also extended to handle non-deterministic transitions.

Modes and mode transitions can be represented using a graph (see Figure 4.1 on page 109 for an example), named a mode transition diagram [Bouroche, 2007]. Comheolaíocht uses the mode transition diagram to analyse each scenario for solvability; a scenario is solvable only if the system's safety is maintained and progress can be achieved. The mode transition diagram is analyzed for the safety property by making sure that entities have control over their transitions into modes that might violate the safety constraints. System progress can be achieved if every entities' mode transition diagrams have a path that allows entities to achieve their goals. When the system is solvable, the analysis outlines a coordination strategy that defines what entities can/cannot do in various situations so as to ensure safety in the system.

As defined in Section 3.4, a scenario is a partition (non-overlapping and non-empty) of the physical space in the environment, therefore, entities moving in the physical space may cross scenarios. Comheolaíocht terms scenarios that are physically connected (i.e., between which entities may move) simply as connected scenarios.

In addition to analyzing a single scenario for solvability, this chapter presents the composition of connected scenarios. This composition step analyzes the scenarios' pre- and post-conditions and any safety constraints that cannot be ensured within a scenario; these constraints are said to be exposed; exposed constraints must be handled before entities enter the scenario. The composite scenario can then be analyzed for solvability by applying the same analysis steps as for analyzing a single scenario; the analysis checks that entities moving between the scenarios respect the exposed constraints.

4.2 Modes

In Bouroche's thesis, modes represent an element's possible actions (actuation, sensing, sending messages or signals, processing, see Table 4.1 on page 106 for an example). In Comheolaíocht, the set of entities' possible actions is captured by entities' behavior (Section 3.2) which model entities' actions based on the developer's perspective. In contrast, modes are designed with the sole purpose of modeling entities' safety in a scenario, Thus:

> An entity's *mode* is a disjoint subset of the entity's states. An entity's *state* defines the situation of an entity at a given time; the entity's state can be described by the values of its set of variables.
>
> A *mode transition* from a mode x, to another mode, y, happens when an entity's state changes states from s_0 to s_1, such that $s_0 \in x$ and $s_1 \in y$. An entity's *mode transition diagram* describes all modes and possible mode transitions.

This section defines the concepts for mode design, and presents a step-by-step guide to designing modes and identifying mode transitions, which will be used in the next section for analyzing a scenario's solvability. The following sub-section introduces the concept of participation variables to be used in mode design. Section 4.2.2 then presents the definitions of fail-safe and non-fail-safe modes. Section 4.2.3 then presents long-lasting fail-safe modes. Section 4.2.4 provides an overview of error handling in Comheolaíocht; in particular the section shows how modes can be used to handle some of these errors. Finally, Section 4.2.5 and Section 4.2.6 provide a step-by-step guide to designing modes and identifying mode transitions.

4.2.1 Participation variables

Two properties of modes are defined by Bouroche [2007]: i) an entity is always in one of its mode, and ii) the transitions between modes are instantaneous. This section introduces *participation variables* and describes the basic construction of modes. It then demonstrates that these two properties hold. We defines:

> The set of *participation variables* is a subset of an entity's state variables such that a mode is a partition (non-overlapping and non-empty) of the range of these variables. The union of all participating variables' values over all the modes is the range of all possible values (i.e., the partitioned modes are collectively exhaustive).

Consider an example using a vehicle entity with three modes: stopped, decelerating and accelerating. The stopped mode groups all the states with a velocity, $v = 0$, the accelerating mode groups all the states with a positive acceleration, $a > 0$, and the decelerating mode groups all the states with a negative acceleration, $a < 0$. In this case, the set of participationg variables are $\{a, v\}$. However, not all values in the participating variables' range are defined in this example; the values, $v > 0$ and $a = 0$ are not mapped to any modes; this is not acceptable. For simplicity sake, this example assumes that the range of velocity is positive, $v \geq 0$.

Modifying the example, a developer could define another mode called 'constant speed' with the values: $v > 0$ and $a = 0$; exactly the ones that are not mapped. Alternatively, the developer may group 'constant speed' and 'accelerating' under one mode - 'traveling' with the partition having $v > 0$ and $a \geq 0$. The two possible partitions for vehicle's modes where $PV = \{a, v\}$ are:

$$\text{i)}\quad \begin{array}{rcl} \textit{constant speed} & : & a = 0, v > 0 \\ \textit{stopped} & : & a \in \textit{range}(a), v = 0 \\ \textit{accelerating} & : & a > 0, v \in \textit{range}(v) \\ \textit{decelerating} & : & a < 0, v \in \textit{range}(v) \end{array} \quad \text{and ii)} \quad \begin{array}{rcl} \textit{travelling} & : & a \geq 0, v > 0 \\ \textit{stopped} & : & a \in \textit{range}(a), v = 0 \\ \textit{decelerating} & : & a < 0, v \in \textit{range}(v) \end{array}$$

Since all values of the participation variables must be mapped to a mode, it follows that an entity is always in only one of its modes. In addition, transitions between modes are instantaneous (see Section 4.2.6).

4.2.2 Fail-safe mode

In Bouroche's thesis [2007], safety constraints are defined to involve two entities (see Section 3.4.4). Therefore, Bouroche's definition of fail-safe modes are defined in relation to the modes of other entities.

In contrast, Comheolaíocht's safety constraint are defined with respect to an entity's use of shared resources. The definition of fail-safe mode reflects this difference:

> A mode m of an entity x is said to be a *fail-safe mode* if and only if all of the states in m respect the safety constraints.

A mode that is not fail-safe is refered to as a *non-fail-safe* mode.

4.2.3 Long-lasting fail-safe mode

The definition of fail-safe mode reflects a snapshot of time and does not consider the implication of what may happen in the future. Consider the case where the developer defines a mode **MovingBeforeJunction**. In this mode a vehicle is approaching the junction at some positive speed. Since the safety constraint is only defined for vehicles in the junction, an entity is not required to hold any resource to be in the **MovingBeforeJunction** mode at some instant in time, therefore, the **MovingBeforeJunction** mode is a 'fail-safe' mode. However, the positive speed constraint in **MovingBeforeJunction** mode implies that the vehicle may only remain in this mode for some finite period before it transitions into another mode, for example, entering the junction and possibly becoming unsafe. While it is sufficient for an entity to remain in a fail-safe mode so as to ensure that it does not violates the safty constraints, it may not be possible for the entity to remain in a fail-safe mode forever.

> A fail-safe mode is a *long-lasting fail-safe mode* (LLFSM) if the entity can remain in that mode for a long enough period such that the entity is guaranteed successful coordination.

In the definition, successful coordination means that the entity has obtained exclusive access to the shared resources that it requires. Successful coordination is required to ensure that the entity does not violate the safety constraint when it leaves the LLFSM and enters a non-fail-safe mode. The duration required to achieve successful coordination is dependent on various parameters such as the coordination protocol, number of entities involved and the lower-level communication guarantees available. An alternative interpretation of 'long enough' is that the entity must take some deliberate action to transition out of a LLFSM; that is, an entity is able to stay in a LLFSM for an infinitely long period and will not automatically transition out of the mode when time passes.

4.2.4 Errors

Errors can happen at the entity level (e.g., breakdown) or at the component level (e.g., sensor failure). Errors in actuator components and entity-level errors can be represented as uncontrollable behavior (see Section 4.2.6). Errors in communication components can be handled at the communication protocol level (Section 5). Sensor component errors are reflected in the state variables. In particular, Comheolaíocht's modes support two types of errors in variables: inaccuracy and omission. If a participation variable exhibits:

Inaccuracy the values might not reflect the actual information that the variable represents.

When such inaccuracy can be bounded (e.g., $x \pm \delta$), the error can be handled by allocating allowances. There are two methods for handling inaccuracy errors in Comheolaíocht: encapsulate at mode design level or expose to resource reservation. In order to encapsulate inaccuracy in a participation variable at mode design, the boundaries of modes that are relativly more dangerous are increased by the error bounds. Section 4.2.5.2 presents the steps for encapsulating inaccuracies in modes. Alternatively, inaccuracies can be exposed to the entities by having each entity reserve the resources they require with allowance for inaccuracies.

In general, exposing inaccuracies to entities is more suitable in a system with heterogeneous entities because each entity may have different accuracy parameters.

Omission there may not be a value available during execution.

When a participating variable exhibits an omission error, the value 'unknown' is included in the variable's range. Since all participating variable values must be mapped to some modes, this implies that the modes containing the value 'unknown' must be associated with the corresponding behavior for handling the situation. Section 4.2.5.2 presents the steps for encoding omission errors into modes.

4.2.5 Mode design

This section provides a step-by-step guide for designing modes. This design considers both safety and sensor component errors in the selection of participation variables and the partition of physical environment.

The first sub-section focuses on the choice of participation variables and partitions' boundaries based on the safety constraints and the second sub-section shows how to modify the partitions to handle errors in the participation variables.

4.2.5.1 Safety

Modes can be designed by analyzing the variables associated with the safety constraints and events that affect those variables. The steps for designing modes are:

1. Constraint violation

   This step looks for participation variables specified directly in the safety constraints. The state space is then partitioned to differentiate those that might violate the safety constraints and those states that would not violate the safety constraints. The modes that might violate the safety constraint are non-fail-safe modes.

   Illustrating using the intersection collision avoidance scenario as an example, the safety constraint was specified in Section 3.4.4.2 and repeated here for convenience:

   (vehicle.crossing $\implies holds(r_{grid})) \vee$ (vehicle.breakdown $\wedge$ (vehicle.location is in intersection))

   The only state variable specified in the safety constraint is vehicle.location; vehicle.crossing and vehicle.breakdown are entity's actions. Therefore this first step creates partitions based on whether the vehicle's location is in the intersection. The modes are:

Modes	location
in_junction	in the intersection
not_in_junction	not in the intersection

   Safety constraints in Comheolaíocht are only defined in terms of resources and contraints on entities' state variables. For simplicity, the intersection scenario illustration in this section includes the roads leading into the intersection. They are actually two different scenarios, they will be shown in detail in Chapter 6.

2. Time passing leading to constraint violation

   The second step is to partition the state space to differentiate between the states that might and the states that will not automatically transition into states in a non-fail-safe mode when time passes. This step chooses participation variables from the set of variables that are both i) affected by changes in time and ii) might affects the values of variables in the safety constraint. A variable that is affected by change in time may changes values without the entity performing any action as time passes.

Continuing with the intersection collision avoidance scenario illustration:

(a) Assuming that the vehicle has variables of {*acceleration*, *velocity*, *steering*, *heading*, *location*}. As shown previously, *location* is in the safety constraint, therefore it directly affects the values of variables in the safety contraint. In addition, the variables *heading* and *velocity* affect the vehicle's location. Therefore, both variables also indirectly affect the safety constraint. Furthermore, *acceleration* affects *velocity* and steering affects *heading*, which ultimately changes the vehicle's *position*. Therefore all these variables might affect the values of variables in the safety constraint.

(b) Assuming that vehicles are controlled by changing their *acceleration* and *steering*, these two variables are not affected when time passes; since the passing of time does not change their values. *Velocity* is a function of the vehicle's *acceleration* and time, it is affected by time. Similarly, *heading* is a function of vehicle's *steering* and time, and vehicle's *location* is a function of *heading*, *velocity* and time. Therefore, the variables affected by time are {*velocity*, *heading*, *location*}.

(c) Variables that are both affected by time and could affect the values of the safety constraint are thus {*velocity*, *heading*, *location*}.

When a vehicle's *velocity* is positive and time passes, the vehicle travels forward, this might cause the vehicle to transition from the **not_in_junction** mode to the **in_junction** mode. Although a vehicle's *heading* affects *position*, it cannot be used to determine transitions from **not_in_junction** to **in_junction** because it is assumed that the vehicle will follow driving rules and they stay on the road (i.e., the vehicle cannot change steering to avoid driving into the junction). In addition, a **not_in_junction** vehicle could have already crossed the junction. Unlike a vehicle traveling before the junction, a vehicle that has crossed the junction would not travel into the junction as time passes. Therefore, a vehicle's *location* is partitioned once more, the modes are:

Modes	location	velocity
in_junction	in the intersection	any
after_junction	after the junction	any
stopped_before_junction	before the intersection	0
traveling_before_junction	before the intersection	> 0

3. Reactions that may cause safety constraint violation

   An entity's outputs may cause other entities to perform some actions that may violate the safety constraints. For example, in a pedestrian crossing scenario, the pedestrians are passive elements who are sensed by traffic lights (i.e., pedestrians push a button to signal they want to cross). As usual, traffic lights with the color of green, amber and red are used to control vehicles crossing the pedestrian crossing, where vehicles may cross the pedestrian crossing on green, must stop if possible on amber and must not cross on red. Therefore, green traffic lights cause vehicle entities to cross the pedestrian crossing, which may violate the safety constraint.

   If (i) an entity's behavior may violate the safety constraints, and (ii) the inputs of this behavior (the conditions that causes the entity to perform the behavior) are caused by another entity's output, then both entities state variables must be partitioned:

   (a) Outputs that may cause other entities to violate the safety constraints

      The first part of third step is to create mode partitions to differentiate between outputs that may and that will not cause other entities to react and possibly violate the safety contraint. The step looks at an entity, x, that modifies some variables in the environment, E, such that another entity, y, might sense E, resulting in y performing some actions that may result in the violation of the safety constraints.

      In the illustration above, the traffic lights entity is the entity x who modifies the environment, the modes of the traffic light entity are:

Modes	color
red	red
green	amber or green

   (b) Actions that may violate the safety constraint due to other entities' output

      The second part of the third step is to create partitions to differentiate between entity's reactions that might and that will not lead to the violation of the safety constraint. The step looks at the entities' sensor inputs and its behaviors.

      For illustration, assuming that vehicle entities in the pedestrian crossing scenario have modes {**after_crossing**, **in_crossing**, **traveling_before_crossing**, **stopped_before_crossing**} that are similar to the intersection crossing scenario's illustration in step 1 and 2. Therefore the vehicle entity's modes in the pedestrian crossing scenario are:

Modes	location	velocity	traffic lights
after_crossing	after junction	any	any
in_crossing	in junction	any	any
traveling_before_crossing_red	before junction	> 0	red
traveling_before_crossing_green	before junction	> 0	amber or green
stopped_before_crossing_red	before junction	0	red
stopped_before_crossing_green	before junction	0	amber or green

These three steps create partitions which can be categorised into non-fail-safe/fail-safe (step 1), possibly LLFSM and fail-safe (step 2) and partitions that either affect or are affected by the environment and other entities (step 3).

4.2.5.2 Errors

As presented in the overview of error handling (Section 4.2.4), some errors can be handled by modes. These include some entity-level and actuator component errors, sensor-component inaccuracies and sensor-component omission errors.

1. Entity-level and actuator component errors.

   When the error is at the entity level or there is an actuator error, the entity may lose control and perform some unplanned actions (e.g., a vehicle entity may break down). When such unplanned actions may violate the safety contraint, an additional variable and corresponding mode partitions should be created to represent the entity's loss of control. Unplanned actions are not unexpected actions, that is, errors are expected to occur but they were not in the entity's plan.

   Illustrating using the intersection collision avoidance scenario, in normal circumstances, a vehicle may only use the resources it has reserved. However, a vehicle may breakdown halfway through the crossing and use resources for longer than allocated. Partitioning the breakdown case, the final set of modes for the intersection collision avoidance scenario with their participating variable's ranges are shown in table 4.1.

   A vehicle breakdown in the intersection violates the safety constraint, therefore an additional partition is created. In contrast, vehicle breakdown on a road (before or after the intersection) may stop the vehicle on the road, which does not violate the safety constraint, therefore vehicle breakdown on the road can be encapsulated in the **stopped_before_junction** mode.

Modes	location	velocity	breakdown
in_junction_traveling	in the intersection	any	no
in_junction_breakdown	in the intersection	any	yes
after_junction	after the intersection	any	any
stopped_before_junction	before the intersection	0	any
traveling_before_junction	before the intersection	> 0	any

Table 4.1: A vehicle entity mode in the intersection collision avoidance scenario

2. Sensor components inaccuracies.

   When the inaccuracy is bounded (e.g., $x \pm \delta$), the error can be handled by extending the boundaries of modes that might violate the safty constraints. The extension causes entities that might experience errors to always act conservatively. As a guide, an entity's non-fail-safe modes (Step 1 from Section 4.2.5.1) are extended first, followed by the modes affected by time passing and other entities' outputs. For instance, if the vehicle's position is inaccurate (i.e., it has a bounded error of $\pm\delta$) then the bounds of the location variable 'in the intersection' is extended by $\pm\delta$ and the values 'before the intersection' and 'after the intersection' are reduced by $\pm\delta$. If such an extension results in a modes' range being undefined (i.e., there are no possible values in range), then the mode is removed. For example, if the vehicles' velocity is inaccurate, then the **traveling_before_junction** mode is extended and the **stopped_before_junction** mode is removed (because the entity cannot be sure that its velocity is zero). Note that, in such cases, there may not be a solution to this scenario.

3. Sensor components omission errors.

   As mentioned, when a sensor component cannot return a value due to an omission error, the value 'unknown' is returned. This 'unknown' value must be mapped to some mode(s) and the entity encoded with the corresponding behavior for handling the situation.

   Extending the vehicle entity in the pedestrian crossing illustration, assume that there are instances where the vehicles cannot observe the traffic lights because another entity blocks their view. In this case, the vehicle entity's representation of the traffic light variable exhibits omission error and the value 'unknown' is included to its range. As vehicle entities act in a safe behavior by not crossing the pedestrian crossing in the '**traveling_before_crossing_red**' and '**stopped_before_crossing_red**' modes, these modes are extended to include the 'unknown' value of traffic lights color.

   The set of modes for the pedestrian crossing scenario with their participating variables range are shown in Table 4.2:

Modes	location	velocity	traffic lights
after_crossing	after junction	any	any
in_crossing	in junction	any	any
traveling_before_crossing_red	before junction	> 0	red/unknown
traveling_before_crossing_green	before junction	> 0	amber/green
stopped_before_crossing_red	before junction	0	red/unknown
stopped_before_crossing_green	before junction	0	amber/green

Table 4.2: A vehicle entity mode in the pedestrian crossing scenario

4.2.6 Mode transition

This section describes a manual method for identifying mode transitions. Using a graph representation, a mode is represented by a vertex, m (see Figure 4.1 on page 109 for an example). A transition from m_1 to m_2 describes that some events (e.g., an entity performing a behavior, time passing) that change the values of its participating variables from some states in m_1 to some states in m_2. The transition from mode m_1 to m_2 is represented by a directed edge, (m_1, m_2). In addition, the events that cause the transition are labelled on the edge.

Comheolaíocht uses a straight-forward method that examines every pair of modes; for each pair of modes,

1. Check whether any behaviors or external factors may result in an entity transition between the modes

2. If there is a transition, label the pair of adjacent modes with an edge and the transition *cause*; otherwise, there is no edge between the pair. *Causes* for mode transitions are categorised into three groups based on the entity's ability to control the cause, the groups are:

Controlled mode transitions are caused by the entity's deliberate actions; that is, the entity can choose not to perform the actions. For example, a vehicle can decide to apply the breaks to transition from the mode '**traveling_before_junction**' to '**stopped_before_junction**' (Table 4.1), and to apply the accelerator to transit from '**stopped_before_junction**' back to '**traveling_before_junction**'.

Although controllable mode transitions are caused by entities performing deliberate actions and mode transitions are instantanous, an entity's actions may not result in instantanous mode transition. In fact, an entity may not be able to decide when the transition will take place. For example, a vehicle applying the breaks in the '**traveling_before_junction**' mode may only result in a transition '**stopped_before_junction**' mode some time later.

Category	Transition causes	Entity knows when transition happens
Controlled	deliberative actions	Yes (with clock)
Uncontrolled	events the entity cannot control	No
Timed	time passing	Yes (with clock)

Table 4.3: Category of mode transition causes

Uncontrolled mode transitions may also be initiated by the entity, however, some events may happen that cause the transition without check. For example, in the pedestrian crossing scenario, the vehicle-entity's mode transitions from '**traveling_before_crossing_green**' to '**traveling_before_crossing_red**' (Table 4.2), the vehicle does not have control over the color of the traffic light. Another example is vehicle breakdown causing the entity to transit from '**in_junction_traveling**' to '**in_junction_breakdown**' (Table 4.1).

Timed mode transitions are a special subset of uncontrolled mode transitions that are caused by time passing. Mode transitions in this sub-group are special because although an entity may predict when the transition will occur (assuming that the entity has an internal clock), the entity cannot prevent the transition unless it can take some action to prevent it. For example, the mode transition from '**traveling_before_junction**' to '**in_junction_traveling**' (Table 4.1) happens with the passing of time if the entity maintains its traveling action.

Table 4.3 shows a summary of the categories of mode transition causes.

Example Figure 4.1 shows the completed mode transition diagram of the intersection collision avoidance scenario with the causes (edges) entered. The diagram shows that not every pair of modes have transitions. In the diagram, most transitions are unidirectional. For example, the transitions from **before-junction** to **in-junction** to **after-junction** is unidirectional because the scenario does not allow the vehicle to reverse out of the junction. An example of bi-directional transitions can be seen between the traveling and stopped modes, as a vehicle that is traveling may apply the breaks to stop and it can accelerate to resume traveling.

In addition, the mode transition diagram highlights the entity's modes for entering and leaving the scenario (entrance and exit modes). In this scenario, the entity must be moving and located 'before the junction' to enter the scenario (velocity > 0), therefore its entrance mode is '**travelling_before_junction**' (i.e., could not have entered the scenario when it is stopped nor can it enters the scenario 'after the junction'). Similarily, a vehicle entity leaves the intersection collision avoidance scenario when it is after junction or after it has broken down in the junction (we assume some engineers clear the vehicle off the road after some time). These value limitations on entities entering and leaving a scenario are recorded

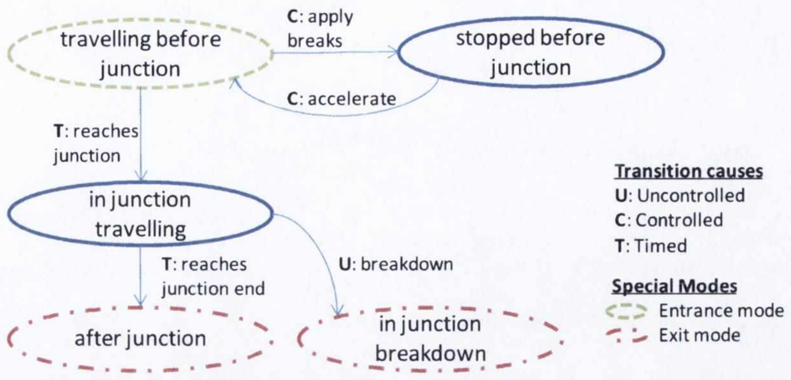


Figure 4.1: Mode transition diagram of the Intesection Collision Avoidance scenario

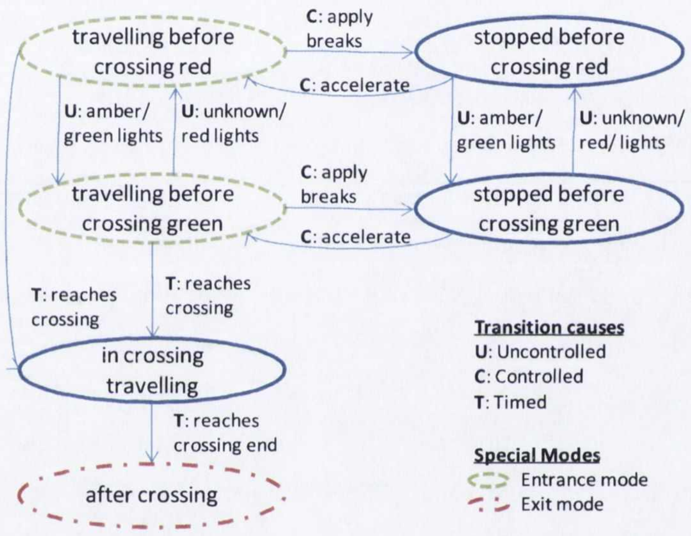


Figure 4.2: Mode transition diagram of the Pedestrian Crossing scenario

under the scenario's entrance and goal conditions (Section 3.4.5.2 and Section 3.4.5.3).

Figure 4.2 shows the mode transition diagram for the pedestrian crossing scenario, the scenario is more complex due to the addition of the traffic lights. In this scenario, the transitions between the traffic lights are not within the control of the entity (uncontrolled).

4.3 Solvability in Comheolaíocht

This section presents a method of analysis of a mode transitions diagram to evaluate whether a scenario is solvable in Comheolaíocht; by following the Comheolaíocht approach, a developer can define coordination protocols ensuring safety and allowing progress. In particular, the following section analyses which modes are LLFSMs and Section 4.3.2 presents an analysis of whether an entity may transition into a mode deterministically. Finally, Section 4.3.3 presents an analysis for calculating the coordination strategy: what entities can/cannot do in various situations so as to ensure that they do not violate a safety constraint.

4.3.1 Identifying long-lasting fail-safe modes

As previously defined, a LLFSM is a fail-safe mode in which an entity can stay long enough to guarantee successful coordination (Section 4.2.3). LLFSMs can be identified using the mode-transition diagram as follows:

4.3.1.1 Simple long-lasting fail-safe modes

A fail-safe mode is a simple LLFSM if it has no out edges with uncontrollable (including timed) causes; that is, either the fail-safe mode has no out edge or all its out edges are controllable. A mode without an out edge is a sink such that an entity who reaches that mode never leaves it. Since the entity may stay in that mode forever, it can indeed stay long enough to guarantee successful coordination. Therefore, a fail-safe mode that is a sink is a LLFSM. If all the out edges of a mode are controllable, then the entity can choose when to transition out of the mode, therefore the entity can also choose to stay within the mode for as long as required to perform successful coordination - the condition required for the definition of long-lasting.

Illustrating using the intersection collision avoidance scenario (Figure 4.1): the vehicle entity has fail-safe modes of {**after_junction**, **traveling_before_junction**, **stopped_before_junction**}. In this scenario, '**after_junction**' is a sink LLFSM as it has no out edges, and '**stopped_before_junction**' mode is another LLFSM because its only out edge has a controllable behavior (i.e., accelerate).

4.3.1.2 Composite long-lasting fail-safe modes

Two or more fail-safe modes can be combined to achieve the effects of a LLFSM. Such composite LLFSMs are formed by a group of fail-safe modes such that all out edges from the group have controllable causes; there can be any number and type of transitions between members within the group. An example of a composite LLFSM can be seen from the pedestrian crossing scenario example (Figure 4.2). The set of fail-safe modes in the pedestrian crossing scenario are {**after_crossing**, **traveling_before_crossing_red**, **traveling_before_crossing_green**, **stopped_before_crossing_red**, **stopped_before_crossing_green**}. In this scenario, the '**after_crossing**' mode is a simple LLFSM as it is a fail-safe mode without an out edge. The modes '**stopped_before_crossing_red**' and '**stopped_before_crossing_gre**
form a composite LLFSM; all out edges of the set are controlled (the two accelerate edges), even though there are uncontrolled edges (lights color changes) between the two modes.

4.3.1.3 Time required for successful coordination

Both simple and composite LLFSMs are identified based on the assumption that successful coordination may require an infinite period of time. However, some protocols may require a less than infinite period of time based on some assumptions (e.g., Schemmer [2004] assumes that there are at most OD consecutive message losses). When using such coordination protocols, the period of time that ensures successful coordination, t_{success}, can be defined. Taking such finite periods into consideration, simple and composite LLFSM can be relaxed to include modes with timed out edges as long as the entity can remain in those mode(s) to perform coordination for at least t_{success}.

Illustrating using the intersection collision avoidance example: if the entity can perform the coordination protocol for t_{success} ensuring successful coordination while in the '**traveling_before_junction**' mode, then the '**traveling_before_junction**' mode is a simple LLFSM.

As mentioned, successful coordination means that the entity has obtained exclusive access to the shared resources it requires. It does not mean successful communication to convey some message to other entities.

4.3.2 Non-determinism

Entities' actuators may be inaccurate or faulty. Inaccuracies and faults in the actuators can be modelled as uncontrollable transitions in Comheolaíocht. This section presents a method of analysis to determine whether an entity can perform a mode transition deterministically, the result of this analysis is applied in analysing a systems' progress and safety in Section 4.3.3.

The following section presents the motivation for analyzing whether entities' may perform transitions deterministically, the next section then presents the analysis.

4.3.2.1 Motivation

By definition, the system's safety in ensured if the safety constraints are not violated. The two tools that Comheolaíocht uses to ensure that no violation of the safety constraints occur are:

1. A coordination pattern (Chapter 5) for the entities to coordinate and ensure that the safety constraints are respected when they transition into a non-fail-safe mode. However, the entities must have *enough time* to engage in the pattern, i.e., it requires a long-enough period to ensure successful coordination.

2. LLFSMs, which are designed for the entity to be in a safe state long-enough in order to ensure successful coordination.

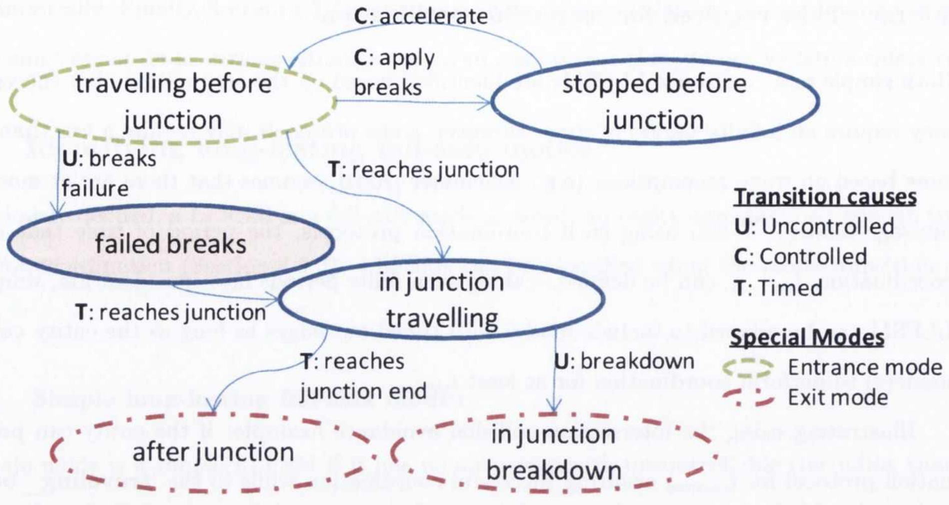


Figure 4.3: Intersection collision avoidance scenario that includes a break-failure mode

Therefore, in order to ensure successful coordination, the entity must be able to transition into a LLFSM deterministically so as to engage the coordination protocol before its transition into a non-fail-safe mode. Besides the two obvious cases where there are no LLFSMs or that the entity must access a non-fail-safe mode to reach a LLFSM, another reason for entities not being able to access a LLFSM is the possibility of uncontrollable transitions. Uncontrollable transitions could make an entity perform some unintended behaviors which make its transition into a LLFSM non-deterministic.

For example, assume that the intersection collision scenario includes a possible error where the entity's breaks fail and it cannot decelerate. In order to represent this condition, a mode '**failed_breaks**' is created, the resultant mode transition diagram is shown in Figure 4.3. In the diagram, an uncontrollable transition from '**traveling_before_junction**' leads into the '**failed_breaks**' mode representing the situation of a breaking failure being discovered when the vehicle tries to decelerate. A timed transition from the '**failed-breaks**' mode leads into the '**in_junction_traveling**' non-fail-safe mode, which shows that the vehicle is still traveling at some velocity and inevitably enters the junction. In this example, due to the possibility of a failure of the breaks the entity cannot deterministically access its LLFSM despite having one; Comheolaíocht cannot ensure a safe solution for such systems.

Therefore, the following sub-section presents an analysis of the mode transition graph to check whether an entity can deterministically transition along an edge. The next sub-section then presents an analysis of whether an entity can take an out edge safely.

4.3.2.2 Non-deterministic transitions

As illustrated in the intersection collision avoidance with faulty breaks scenario, uncontrollable transitions could prevent an entity from accessing LLFSMs. This section presents the analysis for examining whether an entity can perform a transition (along an edge) deterministically. With respect to the analysis of a mode transition diagram's solvability, every edge in the mode transition diagram is to be examined and marked as either deterministic or non-deterministic; the marked mode transition diagram will be used in Section 4.3.3 in the calculation for the coordination strategy.

Note that while timed transitions are a subset of uncontrolled transitions, in this section, uncontrolled is used to refer to the untimed uncontrolled transitions. The analysis is as follows,

For each edge, (m_1, m_2) (representing a transition from m_1 to m_2), in the mode transition diagram, if the edge is a:

Uncontrollable/Timed transition, the transition may be taken deterministically only if:

- There are no other uncontrollable/timed edges out of m_1 OR
- Every other out edge of m_1 is controllable, in which case, the entity can control itself not to transition using those edges.

In addition, when (m_1, m_2) is an uncontrollable transition, the transition may require an unknown period of time (which could be infinite).

Controllable transitions may be taken deterministically only if there are no uncontrollable out edges in m_1; every out edges on m_1 is either controllable or timed.

This is on the assumption that the entity can take the deliberative action to transition along the controllable out edge before the timed action is activated (e.g., that a vehicle can reach the **stopped_before_junction** mode before it has reached the **in_junction_traveling** mode, Figure 4.1).

Note: the existence of uncontrollable out edges from m_1 makes the controllable (m_1, m_2) transition non-deterministic because the entity may transition on a uncontrollable out edge before it can decide to transition along (m_1, m_2).

Table 4.4 summarizes the analysis above. The first column in the table specifies the edge under examiniation, (m_1, m_2) and the second column categorises the other out-edges from m_1. The third column in the table specifies whether the entity can deterministically transition along the considered edge and the fourth column specifies whether the entity can calculate the transition time deterministically.

Edge under examination	Other edges from the same vertex	Can entity transition along edge under examination deterministically?	Do the transition has deterministic transition time?
uncontrolled	some timed/uncontrolled	No	-
	all controlled	Yes	No (possibly infinite)
timed	some timed/uncontrolled	No	-
	all controlled	Yes	Yes
controlled	all controlled/timed	Yes	Yes
	some uncontrolled	No	-

Table 4.4: Entity's transition alone an edge.

4.3.3 Coordination strategy

In Comheolaíocht, a *coordination strategy* defines what entities can/cannot do in various situations so as to ensure that it does not violates a safety constraint. This section presents an analysis to derive such a coordination strategy from a mode transition diagram. The mode transition diagram can be from a single scenario or a composite scenario (see Section 4.4).

The follow sub-section describes the output of this analysis: the coordination strategy. Section 4.3.3.2 then presents an overview of the analysis, in particular, the section describes the analysis flow, and the sub-modules in the analysis. The other sub-sections then present these different sub-modules. Finally, Section 4.3.3.7 summarizes this section.

4.3.3.1 Semantics

This section presents the syntax and semantics of a coordination strategy; it presents the coordination strategy's parameters and their possible values in a coordination strategy, and how an entity may follow such a strategy to ensure that it does not violates a safety constraint.

An example of a coordination strategy for the intersection collision avoidance scenario (the mode transition diagram in Figure 4.1) is shown below, the meaning of the entries and their possible values are detailed next:

Index	Mode	Result	Condition	Comments
1	traveling before junction	Safe	O:3 | 2	Entrance
2	stopped before junction	Safe	-	LLFSM
3	in junction traveling	Safe	I:1 | 2	non-fail-safe
4	after junction	Safe	-	Exit, LLFSM
5	in junction breakdown	Safe	-	Exit

In the table above, the two columns, *Mode* and *Comments*, are not part of the coordination strategy; these columns only serve to label some information to aid the discussion below. The other columns are:

Index The *Index* column labels each mode with an unique identifier, the unique identifier is defined for two purposes:

1. To simplify referencing in the *condition* column and when refering to an edge (e.g., (m_1, m_2) refers to the transition from the **traveling_before_junction** mode to the **stopped_before _junction** mode) in the analysis.
2. To uniquely identify a mode; when two scenarios of the same type are combined (Section 4.4), there will be modes with the same names - therefore, the name of the modes are not used in the analysis.

Result The *Result* column specifies whether an entity can ensures that it does not violate a safety constraint. The possible values of the result column and their meanings are:

Safe An entity in a mode marked *Safe* can ensure that it does not violates a safety constraint.

- An entity should only transition to a mode marked *Safe*.
- When all entrance modes of a scenario are marked *Safe*, the scenario is *self-contained*; that is, an entity entering the scenario can ensures that it does not violate a safety constraint.
- An LLFSM is *Safe* by definition. An entity can stay in the mode for a long enough period to guarantee successful coordination to ensure the entity's safety.
- An exit mode that is not a non-fail-safe mode is *Safe*.

Unsafe An entity in a mode marked *Unsafe* cannot ensure the system's safety; it may not be able to control its actions in order not to violate a safety constraint.

- An entity should not transition to a mode marked *Unsafe*.
- If any entrance mode of the analyzed mode transition diagram is marked *Unsafe*, an entity can only ensures the system's safety if it can deterministically transition into a LLFSM before it enters the scenario.
  - In this case, the scenario's safety constraints are said to be *exposed* to other scenarios.
  - The scenario composition step checks for the LLFSM required to coordinate the exposed constraints.

Deadend An entity in a mode marked *Deadend* can only ensure the system's safety while it remains in that mode. The mode is termed *Deadend* because the entity cannot safely transition to an exit mode.

- There are two cases for *Deadends*:

  Deadend-1 The mode does not have an out-edge transition and it is not an exit mode, the entity will not be able to reach an exit mode.

  Deadend-2 All possible destination modes from the mode are marked *Unsafe*, the entity will not be able to reach an exit mode and ensures that it will not violate a safety constraint.

- An entity should not transition into a mode marked *Deadend*.
- If any of the entrance modes of scenario is marked *Deadend*, an entity can ensures the system's safety only if
  - All entrance modes marked *Deadend* are of case 2, and
  - The entity can deterministically transition into a LLFSM before it enters the scenario.

Condition The *Condition* column describes how an entity should act in order to ensures its safety. This condition is only applicable to a mode marked *Safe*.

There are two entries shown in the example are O: 3 | 2 and I: 1 | 2,

- The first entry O: 3 | 2 can be read as: an entity can only safely transition into mode 3 after is has successfully coordinated with other entities to give way, otherwise, the entity must transition into mode 2. This information will be used in the coordination scheme which specifies an entities' behavior (Section 5.3).
- The second entry I: 1 | 2 can be read as: for an entity transition from mode 1 it must obtain the required shared resources in mode 1 or an earlier mode, otherwise, it must transition to mode 2. This information is required for the calculation of the set of entities to coordinate with (Section 4.4).

Note: There can be more than one entries in the condition column. A condition is not specified when the *Result* is clear, (i.e., *Safe* without requiring the entity to perform anything).

4.3.3.2 Analysis overview

This section presents an overview of the analysis to derive a coordination strategy; the section first presents the overall flow, and then presents an overview of the sub-components required.

Comheolaíocht uses a dynamic programming [Cormen et al., 2001] algorithm for deriving the coordination strategy, this is because the analysis steps exhibit the properties of overlapping subproblems and optimal substructure:

Optimal substructure Whether an entity can ensures its safety in a mode, m_1, is dependent on:

1. Whether it can deterministically access a LLFSM, and
2. Whether the entity can ensure safety in m_1's destination modes

Overlapping subproblems m_1's destination modes may include another mode's destination modes.

Therefore, the analysis for the coordination strategy has the following structure:

- Let
  - *current* represent the current mode under examination,
  - *seenLLFSM* stores the identifiers of the last deterministically accessible LLFSM modes,
  - *tab* represents the strategy (the dynamic programming table),
  - *prev* represents the mode that the entity transitioned from,
  - *Analyze* represents the recursive function to generate the coordination strategy,
  - *G* represents the mode transition diagram (graph),
  - $\oplus$ represents the function to aggregate a set of strategies
- In
  - $$\begin{aligned} Analyze(current, seenLLFSM, tab, prev) = \quad & \forall(current, m_{dest}) \in G, \\ & \oplus\{Analyze(m_{dest}, seenLLFSM, tab, current)\} \end{aligned}$$

The above function can be read as, in order to analyze the strategy for the current mode, $Analyze(current, ...)$, all the destination modes (as defined by the out edges, $\forall(current, m_{dest}) \in G$) are analyzed, $Analyze(m_{dest}, ...)$, and their results aggregated ($\oplus$).

In this analysis, the term *current mode*, m_1, is used to refer to the mode under analysis and the term *destination mode* to refer to a mode to which m_1 has an out edge to. The subsequent subsections presents various calculations for the analysis, in particular

- Section 4.3.3.3 presents the termination case for the *Analyze* function above.
- Section 4.3.3.4 presents the initialization and intermediate results for the strategy table.
- Section 4.3.3.5 presents the recursive case for the *Analyze* function above.
- Section 4.3.3.6 presents a special recursive case for the *Analyze* function - loops.
- Finally, Section 4.3.3.7 summarizes the analysis by presenting the *Analyze* function.

Exit mode	$seenLLFSM = \emptyset$	$seenLLFSM \neq \emptyset$
LLFSM/Fail-safe mode	safe	safe
Non-fail-safe mode	safe, $I\,prev \mid seenLLFSM$	unsafe

Table 4.5: Decision in exit-mode

4.3.3.3 Termination

The Analyze function terminates when the current mode under examination is an exit mode. An exit mode is marked $Unsafe$ in the strategy table only if

- The exit mode is a non-fail-safe mode, and
- The entity cannot deterministically access a LLFSM prior to the transition into the exit mode. ($seenLLFSM = \emptyset$)

Otherwise, the strategy table entry for the exit mode is marked $Safe$.

In the case where the exit mode is a non-fail-safe mode, the condition is marked "$I\,prev \mid seenLLFSM$" to represent that an entity transitioning from mode *prev* must perform the required coordination in mode *prev* or in one of its last deterministically accessible LLFSM modes. Table 4.5 summarizes this termination case. Exit modes that are fail-safe are inherently safe and do not require a condition to be safe.

4.3.3.4 Intermediate coordination strategy table

This section presents an extension of the coordination strategy table to represent its state during initialization and intermediate analysis. Values of the coordination strategy table for finalized results were presented in Section 4.3.3.1.

The table below shows the coordination strategy table just after initialization:

Index	Mode	Result	Condition	Unexamined outedges
1	traveling before junction	Null	-	2, 3
2	stopped before junction	Null	-	1
3	in junction traveling	Null	-	4, 5
4	after junction	Null	-	
5	in junction breakdown	Null	-	

In the strategy table, there is an addition column, *Unexamined outedges*, that record the transitions that have not been analyzed. The table just after initialization has all results set to $NULL$ to represent the fact that a mode has not been analyzed.

The table below shows the table with some intermediate results, in particular, the table shows a snapshot of the table whereby the analysis of modes 4 and 5 is complete, and the analysis is about to mark mode 1.

Index	Mode	Result	Condition	Unexamined outedge
1	traveling before junction	Pending	-	2
2	stopped before junction	Null	-	1
3	in junction traveling	Unsafe	-	
4	after junction	Safe	-	
5	in junction breakdown	Safe	-	

In the intermediate results table, the value of the *Result* column can be *Pending*; it signifies that analysis has started on the mode, but there is no conclusion yet. In this example, mode 3 is marked *Unsafe* because the analysis has not examined mode 2, the required LLFSM. Section 4.3.3.6 presents the analysis for loops (mode 2) and describes revisiting mode 3 to mark it *Safe* (with a condition) for the final results.

4.3.3.5 Recursive step

Assuming that all the destination modes of the current mode, m_1, have been marked with a strategy, this section presents the recursive step: whether an entity can ensures that it does not violate a safety constraint in m_1.

In the analysis below, if m_1 is a LLFSM, then $m_1 \in seenLLFSM$; therefore, the entity can deterministically transition into a LLFSM ($seenLLFSM \neq \emptyset$). The analysis is as follows (a summary of this analysis is shown in Table 4.6):

- If m_1 is a non-fail-safe mode
  - and $seenLLFSM = \emptyset$, then m_1 is marked *Unsafe*.
  - otherwise, the mode is marked based on its destination modes, as specified in the following steps.

- An entity in m_1 can ensure it is safe if:
    1. It can deterministically transition into a destination mode that is mared *Safe*. (Case 1 and 2 in Table 4.6).
    2. It can deterministically transition into a LLFSM ($seenLLFSM \neq \emptyset$) before its transition to the destination mode, and that it can deterministically transition into a destination mode that may be unsafe (Case 5 in Table 4.6).
        (a) Such a destination mode is marked either with:
            i. *Unsafe*
            ii. *Deadend-2*
            iii. *Safe* with a condition in the strategy table (This case happens when another path has evaluated the destination mode previously)
        (b) In this case, m_1 is marked as *Safe*, $O : m_{\text{dest}}|seenLLFSM$ and the destination mode is marked as *Safe*, $I : m_1|seenLLFSM$. (Note: If m_1 is a LLFSM, then it is simply marked as *Safe* and the destination mode is marked as *Safe*, $I : m_1$ respectively)

        Note: In this situation (Case 5 in Table 4.6), m_{dest} must be reanalyzed due to its change in status.

    Although m_1 can be marked as *Safe* as long as one destination mode belongs to the category above, the developer may want to continue to analyze other destination modes of m_1 so that the entity's deliberation can have more choices for transitions.

- An entity in m_1 cannot ensure the system's safety (m_1 is marked *Unsafe*) if:
    1. There are no out-edges in m_1 with case 1, 2, or 5, and
    2. The entity cannot deterministically transition into a LLFSM before its transition to m_1 ($seenLLFSM = \emptyset$), and it may (non-deterministically) transition into a destination mode that is unsafe. (Case 8 in Table 4.6)

- An entity in m_1 can ensure that it does not violates a safety constraint (requires examination of all its destination modes) (marked *Safe*) if:
    1. There are no out-edges in m_1 with cases 1, 2, 5, or 8, and
    2. All transition edges out of m_1 are non-deterministic (with the exception of Case 6; which will be explained shortly) and that every destination mode is either:

(a) Marked as *Safe* (without a condition): (Case 3 and 4 in Table 4.6), or

(b) *Unsafe*, *Deadend-2* or *Safe* (with a condition), and the entity can deterministically transition into a LLFSM before its transition along the examined edges ($seenLLFSM \neq \emptyset$): (Case 7 in Table 4.6)

Note: In this case, the analysis is the same as Case 5 such that:

- m_1 is marked as *Safe*, $O : m_{\text{dest}}|seenLLFSM$ and the destination mode is marked as *Safe*, $I : m_1|seenLLFSM$.
- The destination mode must be re-analyzed because of the new information.

- An entity in m_1 cannot safely transition out of the mode if:

1. m_1 is not an exit mode and there are no out-edges out of m_1 (marked *Deadend-1*) (Case 9 in Table 4.6), or
2. m_1 is not an exit mode and all destination modes are marked as *Deadend-1*, m_1 is also marked as *Deadend-1* (Case 10 in Table 4.6).
3. There are no out-edges in m_1 with cases 1, 2, 3, 4, 5, 7 or 8, and

(a) The entity can deterministically transition into a destination mode, however it cannot ensure safety in this destination mode; the destination mode is marked *Unsafe* or *Deadend-2*, moreover

(b) The entity cannot deterministically transition into a LLFSM before its transition along the examined edge ($seenLLFSM = \emptyset$). (Case 6 in Table 4.6)

In all cases above, the entity has reached a deadend, it could stay in the mode for an infinite period, but it cannot safely transition out of the mode. However, the analysis differentiate between *Deadend-1* and *Deadend-2*.

Modes marked as *Deadend-2* could be remarked as Safe (with a condition) if further analysis revealed that the entity may deterministically access a LLFSM.

4.3.3.6 Loops and unmarked destination modes

On initialization, the result column in the strategy table has the value $NULL$ to represent that all modes have not been analyzed. If an analysis of a mode, m_1, requires the result of a destination mode, m_2, (e.g., the recursive steps described in the previous section requires the result of destination modes to be marked) which is not available ($NULL$), then m_2 should be analyzed in order to analyze m_1. In this case, m_1 is marked as *Pending*.

Cases	m_2 is marked	(m_1, m_2) is deterministic	$seenLLFSM \neq \emptyset$	Analysis result: m_1 is marked
1	Safe	Yes	Yes	Safe
2	Safe	Yes	No	Safe
3	Safe	No	Yes	Depends on all destinations
4	Safe	No	No	Depends on all destinations
5	Unsafe/Deadend-2	Yes	Yes	Safe, O: $m_2 \mid seenLLFSM$
6	Unsafe/Deadend-2	Yes	No	Deadend-2
7	Unsafe/Deadend-2	No	Yes	Depends on all destinations
8	Unsafe/Deadend-2	No	No	Unsafe
9	No out-edges in m_1			Safe if m_1 is exit mode, Otherwise Deadend-1
10	Deadend-1	Yes	-	Deadend-1

Table 4.6: Ensuring safety at a mode (m_1) - No loops

A loop is discovered when the analysis of a mode, m_x, requires the result of another mode, m_{loop}, which has been marked *Pending*. If the *seenLLFSM* in m_{loop} is a subset of the *seenLLFSM* in m_x, it implies that the entity can transition through some LLFSM by passing through the loop, therefore, the *seenLLFSM* in m_{loop} is updated to reflect the larger set of deterministically accessibe LLFSMs.

Leaving the modes in the loop marked as *Pending*, the analysis continues by analyzing other unmarked destination modes of m_{loop}. When all destination modes of m_{loop} are marked (not *Null*):

1. If there was an update on *seenLLFSM* due to any of the loops, then re-analyze all destination modes marked *Unsafe* or *Safe* (with a condition).

2. After the re-analysis of m_{loop}'s destination modes, m_{loop} is marked as presented in the recursive step (Section 4.3.3.5) by ignoring all loops (transitions marked *Pending*).

3. After marking m_{loop}, re-analyze all the loops (destination modes of m_{loop} marked as *Pending*).

4.3.3.7 Summary

This section presented an analysis to derive a strategy that defines whether an entity can ensures that it does not violates a safety constraint when it transitions to a mode. Given this strategy, an entity should only transition into modes marked as *Safe*, the condition specifies that an entity may be required to transit into the LLFSM in order to ensure the non-violation of safety constraints.

Appendix I shows the pseudo code for the Analyze function.

4.4 Scenario Composition

The modelling in Chapter 3 splits the application environment into scenarios and the first three sections of this chapter focus on analyzing the solvability of a single scenario. This section presents *Scenario Composition*, systematic steps for combining the partitioned scenarios back into the application environment. In the process, entities' coordination across scenarios is analyzed and the validity of compositions is checked.

The following sub-section presents the design and analysis steps applied to scenario composition. Section 4.4.2 presents the concepts underlying the composition of two scenarios. Section 4.4.3 then presents the checks for incompatible compositions (i.e., scenarios that cannot be composed). Finally, Section 4.4.4 presents the steps for composing scenarios.

4.4.1 Applying design and analysis to composite scenarios

Just as composite scenarios are also scenarios, the same set of design and analysis tools can be applied to analyzing composite scenarios. This section focuses on the re-design and analysis of modes for composite scenarios. The steps are:

1. Identify mode transitions between the entrance-modes and exit-modes of the scenario; this step is the same as Section 4.2.6.
2. Identify entrance modes with resource usage pre-conditions. A scenario may have pre-conditions that specify that entities must satisfy some constraints before they enter the scenarios. The pre-conditions supported by Comheolaíocht can be grouped under two categories: value-satisfying and resource usage:
   (a) *Value satisfying* pre-conditions specify that an entity's parameters should be within some range or in some modes. These pre-conditions are specified using the following syntax (see Section 3.4.5.2):

   $$\begin{aligned}\text{entity\_condition} := \;& \text{entity-type}, \text{"."}, \text{state-variable}, \text{rel-operator}, \text{value}\\ & |\,\text{entity-type}, \text{"."}, \text{entity-behavior\_ID}\\ & |\,\neg\text{entity-type}, \text{"."}, \text{entity-behavior\_ID}\end{aligned}$$

   (b) *Resource usage* pre-conditions specify that an entity must hold exclusive use of some resources in order to perform some actions. These pre-conditions are specified using the following syntax (see Section 3.4.5.2):

   $$\text{resource\_usage} := \text{entity\_condition} \Rightarrow holds(\text{resource\_name}_{id[\odot copy]})$$

Similar to a safety constraint that cannot be handled locally (Section 4.3.3.1), the resource usage pre-conditions are *exposed* to the other scenarios. As such, entrance modes with a resource usage pre-condition is represented as non-fail-safe modes in the composition; entities must coordinate to obtain the required resources in order to safely transition into the entrance modes.

3. Revisit the mode design step (Section 4.2.5) for the composite scenario, further partition any modes to differentiate between states that may eventually violate a pre-condition and states that will not violate a pre-conditions.

   An example is shown in Chapter 6 where a road scenario is composed with a junction scenario. In the example a road scenario has a **Driving** LLFSM, and the junction scenario has a pre-condition that requires entities to hold some junction resources before they enter the junction (i.e., cross the junction). As a vehicle in **Driving** mode may eventually reach the end of the road and enter the junction, the mode is split into **Driving_before_junction** and **Stopped_before_junction** modes (Figure 6.6). The road and junction composite scenarios together form the intersection collision scenario (Figure 4.1).

4. Apply the coordination strategy analysis (Section 4.3.3) to find a strategy for the composite scenario.

After applying these steps, the composite scenarios can be treated as a single scenario. For instance, composite scenarios can be further composed to form more complex composite scenarios (see Section 6.3 for an example of composing composite scenarios).

4.4.2 Composing two scenarios: concepts

Scenarios are non-overlapping partitions of the environment. This implies that as soon as an entity leaves a scenario, it enters another scenario. In order for an entity to enter a scenario safely, the scenario's pre-conditions must be satisfied. This section presents three questions an entity needs to answer in order to transit between scenarios:

1. Which scenario is the entity entering?
2. When should an entity start its preparation for satisfying the pre-condition?
3. Who should the entity coordinate with?

4.4.2.1 Which scenario is the entity entering?

When there are more than one scenario composed together, an entity leaving a scenario could be entering any of the other scenarios. Therefore, the entity must know which scenario it is joining so as to know which set of pre-conditions it must satisfy. This work assumes that the entity's higher-level intelligence (e.g., planning) knows which scenario an entity is joining; the higher-level intelligence is not within the scope of the coordination protocol (see Section 1.7).

4.4.2.2 When should an entity start its preparation to satisfy the pre-condition?

How early an entity starts its preparations depends on the pre-conditions. As mentioned, the pre-conditions supported by Comheolaíocht can be grouped under two categories:

Value satisfying pre-conditions that specify an entity's parameters should be within some range or in some modes.

The *reaction time*, δ, is the minimum time required for an entity to modify the constrained parameters into the acceptable range before its arrival at the new scenario. This implies that the entity's preparations for these pre-conditions must start at the latest δ before the entity enters the scenario. Although δ defines the latest time for preparation, the entity may start its preparations earlier based on some user preferences. For example, in the scenario composition of a vehicle entity exiting a highway and entering a road, the vehicle must observe the lower speed limit pre-condition of the road. The vehicle entity can either:

- perform maximum deceleration at the last minute so as to arrive at its destination earlier, or
- gradually decelerate at an earlier time so as to allow passengers a more comfortable journey.

Resource usage pre-conditions specify that an entity must obtain exclusive use of some resources in order to enter the scenario safely.

Similar to an entity entering a non-fail-safe mode, an entity entering a scenario with a resource usage pre-condition should start preparation for its entrance in the last deterministically accessible LLFSM. Such a LLFSM can be found in the coordination strategy of the composite scenario:

1. Let the entrance mode with the pre-condition be m_e,
2. The entry for m_e in the coordination strategy could be marked
   (a) *Unsafe*: there are no deterministically accessible LLFSM, and an entity may not safely transition to the entrance mode. In this case, some unhandled constraints are *exposed* to other scenarios.
   (b) *Safe* with a condition (e.g., $\{I : m_{y1}|\{m_{z1}\}, I : m_{y2}|\{m_{z2}\}, ...\}$): then the entity should start its coordination in one of the deterministically accessible LLFSM, (m_{z1} or m_{z2} in the example).

4.4.2.3 With who should the entity coordinate?

This section presents the minimal group with which an entity must coordinate to ensure the safety constraint (or the pre-condition) is not violated. In contrast, the protocol derivation step (Chapter 5) extends this group to enable an entity to reach its goal faster (i.e., by taking its planned path, which may not transit through a LLFSM).

Straight composition Let's start by explaining composition between two scenarios, s_1 and s_2, such that entities only travel from s_1 to s_2. When s_2 has a resource usage pre-condition, entities must coordinate before they enter the scenario. By definition, an entity can remain in the LLFSM for long enough so as to guarantee successful coordination. Further assume that s_1 has such a deterministically-accessible LLFSM that the entities can use for coordination before entering s_2. In the worst case where communication has failed for a prolonged period of time, every entity requesting entry into s_2 (i.e., requesting access to the shared resource) is waiting in its LLFSM in s_1. Therefore, in order to be safe, an entity must be able to coordinate with everyone in s_1's last deterministically-accessible LLFSM. However, utilizing the scenario's α-constraints, which defines the sequence of resource usage, an entity's dependencies can be inferred (Section 3.4.3). This minimal group for coordination can be redefined to be those entities with the highest priority (as defined by the developer) or no dependencies (as inferred from the sequence constraints) in the last deterministically accessible LLFSM.

Generalisation The argument for the minimal group with which to coordinate made two relaxations: i) that s_1 has a deterministically accessible LLFSM and ii) straight scenario composition.

- If s_1 does not have a deterministically-accessible LLFSM, then the pre-conditions are *exposed* to other scenarios.
- When there is more than one scenario leading into s_2, then by following the same argument as the straight scenario composition, the minimal group with which an entity must coordinate are the entities with the highest priorities or no dependencies in the last deterministically-accessible LLFSM *of all the scenarios leading into* s_2. The steps are:
    1. Apply the design and analysis steps to the composition (Section 4.4.1).
    2. In order for an entity to transition through an entrance mode m_e of s_2 in the composition, if m_e is marked
        (a) *Safe*: then an entity can transition into the scenario through m_e without coordinating with any other entity.

(b) *Safe* with a condition (e.g., $\{I : m_{y1}|\{m_{z1}\}, I : m_{y2}|\{m_{z2}\}, ...\}$):

i. If all the in-edges (i.e., m_{y1},m_{y2}, ...) are specified in the condition, then the entity needs to coordinate with all the entities (with the highest priorities or no dependencies) in the last deterministically-accessible LLFSM (i.e., m_{z1},m_{z2}, ...).

ii. If any of the in-edges are not specified in the condition:

A. If all the entrance modes of the composition are marked *Safe* or *Safe* (with condition), then all entities have a strategy not to use the in-edge to m_e without a condition (which may be marked *Unsafe*). In this case, the entity need to coordinate with all the entities in the last deterministically-accessible LLFSM (i.e., m_{z1},m_{z2}, ...).

B. Otherwise, there are no deterministically-accessible LLFSMs for the in-edge. The entity may not safely transition through m_e as it cannot determine the set of entities with which to coordinate.

(c) *Unsafe*: then there are no deterministically-accessible LLFSM to be found in the composite scenario. The entity may not safely transition through m_e as it cannot determine the set of entities to coordinate with.

4.4.3 Exposing incompatible compositions

Due to physical and technical limitations, not all scenarios can be composed. This section discusses these limitations in scenario composition and shows how to identify these incompatible compositions. The two reasons why scenario compositions are incompatible are: physical disjoints, and high-coordination requirements.

4.4.3.1 Disjoint incompatibility

Disjoint incompatibilities exist due to physical limitations. Disjoint incompatibility can be recognized by:

1. Mismatched physical boundaries

   Scenarios include physical boundaries like roads, lanes, and walls. It is a clear sign of incompatible scenarios when the specifications of these physical boundaries do not match.

2. No behavior for transition between the scenarios

   An entity leaving a scenario through its exit mode arrives at the entrance mode of its new scenario. A disjoint incompatibility may exist when an entity does not have a behavior that provides the transition from the previous scenario's exit mode to the new scenario's entrance mode.

3. Some of the entities in both scenarios are different.

   Entities come from somewhere and go to somewhere. If an entity appears in a scenario but not in another, then there is a possibility that the two scenarios are incompatible.

4.4.3.2 High coordination requirements

As discussed in Section 4.4.2.3, the minimal group with which an entity must coordinate are the entities with the highest priority or no dependencies in their last deterministically-accessible LLFSM of all scenarios leading into the entrance scenario. In addition, Section 4.4.2 shows the set of entities with which an entity must coordinate in order to enter a scenario. The combination of these two requirements may result in highly challenging coordination requirements:

1. Over huge distances

   Even when the set of deterministically accessible LLFSM can be calculated (based on the coordination stategy) their physical location could be very far apart. Such coordination may be impractical due to limitation in the hardware required for the coordination (e.g., communication, sensors) and inaccuracies in entities behavior.

2. Between a very large number of entities

   When there are a very large number of entities that may be concurrently in the set of deterministically-accessible LLFSM required for coordination, the physical limitation of the computational resources (i.e., computational power, bandwidth, memory) may make such coordination impractical. This challenge is presented in Section 2.1.4.1.

3. Errors

   Inaccuracies can be handled by allocating extra allowance in modes or requiring entities to reserve extra resources (Section 4.2.4). The effect of catering for these errors may adversely effect the system's efficiency. This inefficiency might mean that the system is impractical.

For illustration, assume a highway scenario composed with an intersection collision avoidance scenario (Figure 4.4). The intersection collision avoidance scenario's pre-condition states that vehicles must hold the relevant resources before crossing - a resource usage precondition. Further, assuming the highway scenario models vehicles that cannot stop, then vehicles in highways do not have the required LLFSM to coordinate entry into the junction. Even if the highway scenario is composed with other scenarios that have deterministically-accessible LLFSMs:

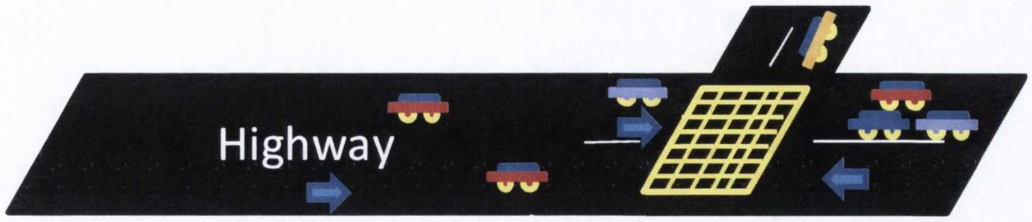


Figure 4.4: A non-compatible composition: highway connected to a intersection.

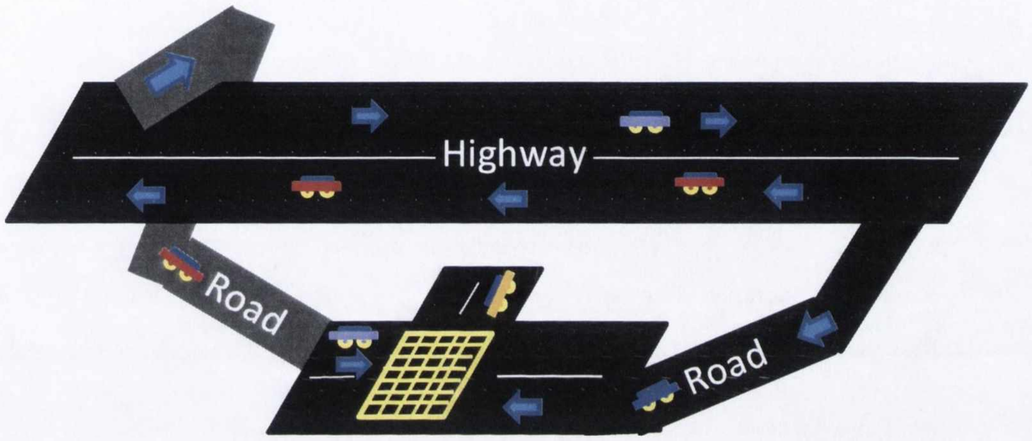


Figure 4.5: Current implementation: Highway-road-intersection composition

1. These deterministically-accessible LLFSM may be physically far apart (since the highway may be long).
2. There may be a very large number of entities if there are many scenarios connected to the highway scenario (many entrances into the highway).
3. The vehicles are required to plan and coordinate in their last deterministically-accessible LLFSM, which may be far away from the actual resource usage (i.e., the junction crossing). Furthermore, inaccuracies in vehicle's velocity implies that it cannot accurately predict the resource usage.

Together, the limitations make such a composition impractical for coordination while ensuring safety.

To complete the illustration: while it is too challenging to compose a highway to an intersection, a common solution is to insert a road scenario between the two scenarios (Figure 4.5), and do not assume that vehicles cannot stop in the highway scenario. This composition shows the current highway to intersection implementation - where highways are always connected to an intersection by a normal road which allows vehicle stopping and traveling at a slower speed.

4.4.4 Summary

This section presented the steps for performing scenario composition. The steps are:

1. The scenarios to be composed are checked for physical incompatibility.
2. The composite scenario's modes are re-designed and analyzed.
3. The information required by an entity is calculated.
4. High requirements that may render a composition impractical are check.

4.5 Summary

This chapter presented a step-by-step guide to analyze the system specified in Chapter 3. The analysis focuses on checking whether an entity can ensure that the safety constraints will not be violated. When the entities can ensure non-violation of the safety constraints, the analysis produces a coordination strategy ensuring this property. The coordination strategy specifies:

1. The set of entities with which an entity must coordinate with,
2. When to start preparing for the transition into a non-fail-safe mode,
3. Whether a transition is safe.

In the event where an entity cannot ensure its safety within a scenario, the safey constraints could be exposed to a connecting scenario. A entity entering a scenario with exposed constraints must start its coordination outside the scenario.

Chapter 5

Protocol Derivation

The final step in Comheolaíocht's uses our CwoRIS pattern to implement entities coordination. Using the environment model and constraints from step one, and the coordination strategy from step two, the CwoRIS pattern derives a protocol to provide distributed scheduling or mutual exclusion for dynamic participants that provides system safety. The derived protocol provides system safety by ensuring entities have exclusive access to shared resources despite imperfect communication.

The following subsection describes the communication and sensor requirements. Section 5.2 explains the concepts of the CwoRIS pattern which is implemented by three parts: coordination scheme (Section 5.3), request/response protocol (Section 5.4) and local scheduling/rescheduling (Section 5.5). Section 5.3.4 then presents an example of applying the CwoRIS pattern using the Intersection collision avoidance scenario.

5.1 Communication & Sensor Requirements

The CwoRIS pattern is designed to operate on top of a (1) multicast protocol that provides (2) ordered delivery of messages, (3) bounded message latency and (4) real-time feedback. Ordered delivery of messages ensures that messages that are delivered to some nodes arrive at these nodes in the same order [Kshemkalyani and Singhal, 1998]. Bounded message latency ensures that messages are either delivered within a certain period or discarded. Feedback on message delivery ensures that a sender receives result on the entities that a message is delivered. Real-time feedback ensures that feedback on message delivery is delivered to the sender within a bounded time.

The ordered delivery and real-time feedback properties replace the requirement for the consensus property (Section 2.4.1).

Ordered delivery is essential to CwoRIS because allocations are granted based on first-come/first-served; an unordered sequence may cause two entities with conflicting requests to believe that they have won the first-come/first-served race and access the shared resource at the same time, which is not safe. Ordered delivery can be implemented with time-stamped messages and buffers at reciever end. When a node receives a message, the message is stored in the buffer and sorted based on the message's time-stamp. These stored messages are delivered to the application in-order, delayed messages that may result in out-of-order delivery are discarded.

Real-time feedback is required because CwoRIS supports a real-time system where entities must make decisions based on the outcome of whether a message is delivered within a fixed period. Real-time feedback can be implemented at the senders' side with a timer. The timer is started after sending a message, when the time is up acknowledgements are aggregated and feedback to the sending application.

When applied to physical entities, CwoRIS further assumes that the lower-layer communication protocol provides geocast. Several protocols satisfy our requirements for geocast, ordered-delivery, bounded message latency and real-time feedback, (e.g., STEAM [Hughes, 2006](Section 2.5.2), Vertigo [Slot et al., 2010]). STEAM provides feedback on the area to which a message was successfully delivered. Vertigo uses a combination of sensors and routing protocol to provide both the set of nodes to which a message was successfully delivered (D), and the set of nodes that might be in the area (A) (a failure occurs when $A - D \neq \varnothing$).

Furthermore, the implementation of the mutual exclusion version of CwoRIS pattern requires the use of a reliable resource sensor for detecting entities using the shared resources. Entities may access the resource sensor to check whether there are any entities using the shared resources.

In addition to communication requirements, CwoRIS assumes that physical entities have i) a speed upperbound, and ii) sufficient control to use only the resources it requires.

5.2 Concept

In the CwoRIS pattern, an entity does not access the shared resources unless it can be sure that every other entity that may concurrently access the critical section has given way to it. Bouroche [2007] termed such behavior as being "responsible". Each entity is responsible for respecting the safety constraint (not entering the critical section), and an entity can only enter the critical section by transferring its responsibility to other entities which then give way to it. The CwoRIS pattern is an extension of the "contract without feedback" protocol, one of the contract protocols explored by Bouroche [2007]. In the contract without feedback protocol, an entity transfers its responsibility by delivering a request to everyone that may be interested in the entity's intention to enter the critical section, this results in granting allocations on a first-come/first-served basis. Any entity that receives such a proposal implicitly agrees to the proposal and defers to the sender; it must not access the shared resources at the same time as specified in the proposal message. The sender of the proposal establishes that it is safe to relinquish responsibility when the proposal has been delivered to everyone which can access the resources at the same time using communication feedback.

As mentioned CwoRIS is composed of three parts: the coordination scheme, the request/response protocol and scheduling/rescheduling. Based on the coordination strategy from step two (Section 4.3.3), a coordination scheme (Section 5.3) is derived that defines when and how should an entity act in various situations such as successful/unsuccessful resource reservation, race conditions, and entity breakdowns. The request/respond protocol (Section 5.4) implements the protocol where entities send requests and are granted exclusive access to the shared-resources based on a first-come/first-served policy. In addition, the protocol supports the various β-constraints captured in the system modeling step (e.g., preemption, setup time). Finally, local scheduling/rescheduling (Section 5.5) provides a method to schedule and reschedule a request that prevents a live-lock in the requests.

5.3 Coordination Scheme

The coordination scheme specifies the four steps an entity takes in order to guarantee exclusive access to the shared resources. In the definitions, $MsgLatency$ represents the latency within which a request must be delivered and feedback about message delivery returned to the sender. Figure 5.1(c) shows the example of a vehicle in the intersection collision avoidance scenario using CwoRIS's four steps, Figure 5.1(a) shows the vehicle entity mode transition diagram before entering the junction and Figure 5.1(b) shows the time line of the mode transitions with respect to the four steps. The four steps in the coordination scheme are:

1. Lurking: During this step, an entity listens and builds the situation picture, before arriving at its decision point (Section 5.3.1 describes the decision point). The lurking time is defined to be Δ.

   Note: the entity only stops listening after step 4 - after it had made a decision to access the shared resources.

2. Resource request sending: The second step uses the request/feedback protocol (Section 5.4) to send a request to the set of entities that might access the same resources at the same time. The request message is sent at least $MsgLatency$ before the entity's arrival at its decision point. Section 5.3.2 describes the calculation of the set of recipients and the lurking time.

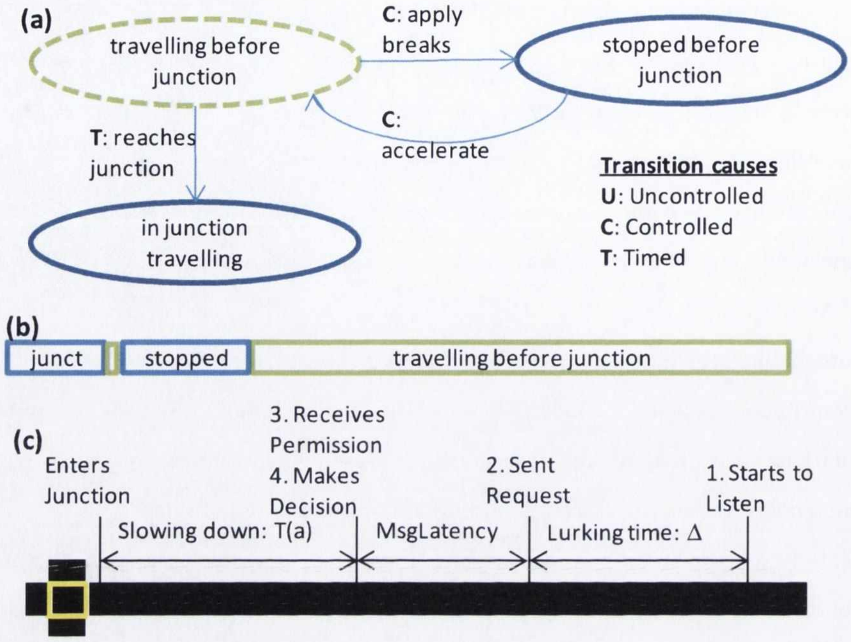


Figure 5.1: Steps for a vehicle using CwoRIS.

3. Request delivery and feedback receipt.

   The request should be delivered and the sender should receive feedback on the delivery results, at most $MsgLatency$ after the resource request is sent.

4. Decision making.

   Based on the result of request delivery, the entity may choose whether to access the shared resources or resend another request. There are two possibilities that can prevent an entity from acquiring access to its required resources: communication errors and the receipt of a conflicting request. The request/feedback protocol (Section 5.4) ensures exclusive access to shared resources despite race conditions and communication errors.

   In the event where an entity arrives at its decision point without having acquired exclusive access to resources it requires, it must start its transition into a long-lasting fail-safe mode (LLFSM; see Section 4.2.3). In this case, the entity sent requests to access the shared resources in its LLFSM.

5.3.1 Decision point

An entity may require time and space to prepare for mode transitions (e.g., vehicles require time to break to a stop). If i) an entity did not acquire exclusive access to the shared resources, and ii) by maintaining its current behavior, may result in it accessing the shared resources, then the entity must change its behavior to avoid accessing the shared resources. We define the last point in time where the entity must start changing its behavior in order to avoid accessing the shared resources as the *decision point.*

Definition: The *decision point* is the last point in time where an entity must start changing its behavior to avoid accessing the shared resources.

The location of a decision point can be derived from the coordination strategy: a decision point can be found at every mode specified in the coordination strategy with a condition specified as $O : x \mid y$ without a $I : z \mid y$ condition. Note: In the coordination strategy table for mode m, if the $I : z \mid y$ condition is present, it implies that a LLFSM, y, can only be found in the previous mode, z, or eariler; the mode m cannot transition into y.

5.3.2 Sending area and lurking period

Section 4.4.2.3 shows that the minimal group an entity must coordinate with to ensure the safety constraint (or the pre-condition) is not violated. While this minimal coordination group definition is sufficient to ensure safe coordination, it implies that every entity must transition into a LLFSM in order to coordinate; this might not be efficient. Illustrating using the intersection collision avoidance scenario: coordination only amongst this minimal group implies that each vehicle must stop (enter the LLFSM) before it coordinates and enters the junction. Although current intersections that have no traffic lights require the vehicles to stop, coordinating entities could be more efficient by coordinating earlier in order to cross the junction without breaking. Therefore, a system could be more efficient if the entities complete the coordination process before they reach their decision points.

An entity must coordinate with all the entities that might access the resources it require. Therefore, to be efficient, an entity must coordinate with a larger group than the minimal group defined in Section 4.4.2.3.

The CwoRIS pattern provides both scheduling and mutual exclusion versions of the protocol. The following sub-section presents the sending area and lurking time for the scheduling version, the sub-section after that presents the concepts applied to the mutual exclusion version.

5.3.2.1 Scheduling version

Lemma (Sending Area): Given a set of required resources $R_a = \{\langle r_{x,y}, t_s, t_e \rangle\}$, an entity must deliver its request to every entity that might use some of the shared resources before $\max_{x \in R_a}(x.t_e)$.

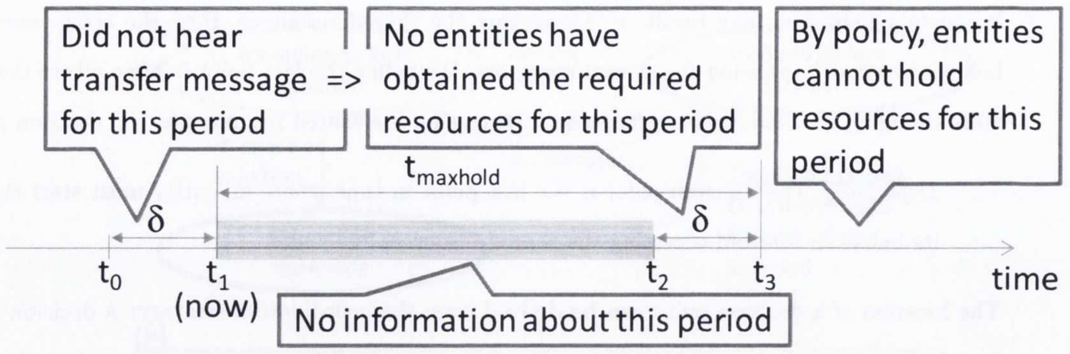

Did not hear transfer message for this period
=>
No entities have obtained the required resources for this period
By policy, entities cannot hold resources for this period
t_{maxhold}
δ
δ
t_0
t_1 (now)
t_2
t_3
time
No information about this period

Figure 5.2: Lurking time overview (scheduling version)

The calculation of the sending area is a direct translation from the definition that an entity must coordinate with all the entities that might access the resources it requires. This definition of the sending area is not exactly an area as it uses time ($\max_{x \in R_a}(x.t_e)$) as a measurement. A developer may need to define additional policies in order to translate this definition into an actual geographical area required for geocast, for example, the intersection collision avoidance (Section 5.3.4.1) uses the vehicles' maximum velocity on the road ($v_{\max}$) to translate the time-definition into the geographical area for the location of entities that may use the shared resources "before $\max_{x \in R_a}(x.t_e)$".

> *Lemma (Lurking Time)*: Given a policy that defines the longest period for which a resource can be reserved is t_{maxHold}. In order for an entity x to ascertain that no other entities is holding on to a resource γ after $t_{\text{maxhold}} - \delta$ from now, entity x must be able to access γ before $t_{\text{maxhold}} - \delta$ from now and has been lurking for δ without hearing a conflicting request.

Figure 5.2 shows an overview of the lurking time lemma. In the figure, t_1 shows the current time (now), entity x has lurked for a period of δ which is between t_0 and t_1. The policy states that entities cannot reserve resources for a period longer than t_{maxhold}, therefore, no entity could book a resource now (t_1), and still be holding to it after t_{maxhold} ($t_1 + t_{\text{maxhold}} = t_3$). The paragraph below presents the proof: *proof*: Assuming that this lemma is incorrect, then

1. Some entity y holds the resource γ after $t_{\text{maxhold}} - \delta$ from now (after t_2 in Figure 5.2).
2. Entity x can access resource γ between ($t_{\text{maxhold}} - \delta$) and t_{maxhold} from now (period between t_2 and t_3 in Figure 5.2).
3. Entity x lurking for a period of δ (from t_0 to t_1 in Figure 5.2) and hearing no requests for resource γ.

4. Due to the policy limiting the longest period a resource may be reserve for (t_{maxhold}), in order for entity y to hold γ after $t_{\text{maxhold}} - \delta$ from now (condition 1), the earliest time y can sent its request is $t_{\text{maxhold}} - (t_{\text{maxhold}} - \delta) = \delta$ ago (in Figure 5.2, at t_0).

5. By Lemma *sending area*, an entity sends a request to every entity that might access the same resources at the same time. In addition, the entity only holds a resource after it has received a feedback for its request from every entity in this group (c.f., Section 5.4). Entity x can access resource γ between ($t_{\text{maxhold}} - \delta$) and t_{maxhold} (condition 2), which overlaps the period that entity y holds the resource for (condition 1). Therefore, entity x must be in y's sending area.

6. Entity x is in y's sending area (condition 5) and has been listening for δ (condition 3). Entity y sents its request not earlier than δ ago (condition 4). Therefore x, must have heard y's request for resource γ (a contradiction).

Therefore, Lemma *Lurking time* is shown. Note: in the event of a communication failure, entity y could not have received x's feedback and could not be holding on to resource γ.

5.3.2.2 Mutual exclusion version

Unlike the scheduling version where entities know the period for which they require a resource (i.e., $R_a = \{\langle r_{x,y}, t_s, t_e \rangle\}$) entities using the mutual exclusion version do not know when they will finish using the resource (i.e., t_e is undefined). As another entity do not know when an entity will finish using a resource, the mutual exclusion version assumes that there is a reliable resource sensor for the detection of any entity using a resource. Furthermore, the policy for the mutual exclusion version depends on the resource start usage time (i.e., t_s).

> *Lemma (Sending Area-Mutual exclusion)*: Given a policy that defines that an entity must either start using the resources within t_{maxME} after the resource acquisition or give up accessing the resources, an entity, x, must send its request to everyone that might access the resources within $(2 * (t_{\text{maxME}} + MsgLatency)$.

The sending area lemma in mutual exclusion version looks different from that of the scheduling version because the former uses resource start usage time (t_s) and the latter uses the resource end usage time (t_e). As the sending area and lurking time are related, both lemmas are provided before their proofs, the lurking time definition is:

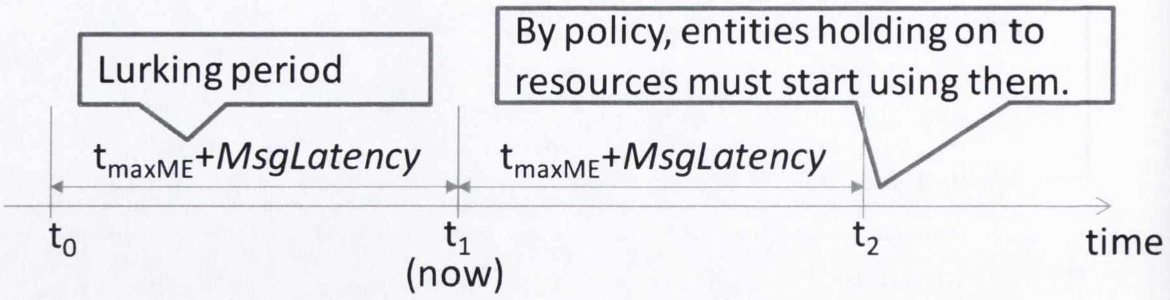


Figure 5.3: Lurking time overview (mutual exclusion version)

Lemma (Lurking Time-Mutual exclusion): If an entity x is in the *sending area*, heard no transfer messages for a period of $t_{\text{maxME}} + MsgLatency$ and the resource sensors do not register any entity currently using the resources, then there is no entity holding exclusive rights to the resources.

There are two properties regarding the sending area and lurking time for the mutual exclusion version: i) $t_{\text{maxME}} \geq 0$ therefore, the lurking time must be at least $MsgLatency$ and ii) the sending area is double the lurking time. Figure 5.3 shows an overview of the lurking time for the mutual exclusion version. Note that similarly to the scheduling version, the sending area is defined using time, actual implementation need to translate this time representation into geographical area.

proof: Assuming that the lurking time lemma is false, then there is an entity, y, holding exclusive rights to some resources, despite:

1. Entity x:

    (a) being in the sending area for a period of $t_{\text{maxME}} + MsgLatency$ and

    (b) did not hear any conflicting resource request,

2. The resource sensors not registering any entity using the resources.

3. Entity y holding exclusive rights to some resource could only have two possibilities, either y is currently using the resources, or that y holds rights and will use it before t_{maxME}. Since condition 2 states that the resource sensors do not register any entity using the resources, y is not currently using the resources, implying that y must be holding the resource rights for usage before t_{maxME}. In order for y to hold these rights:

- The latest time for y's request to be delivered is just before now (time t_1 in Figure 5.3), this case happens when y uses the resource just before t_{maxME} from now.
- The earliest time for y's request to be delivered is $t_{\text{maxME}} + MsgLatency$ ago (time t_0 in Figure 5.3), such that y may use the resource now (at time t_1). In this case, y's message took 0 time to be delivered at x, and y only holds the resources after $MsgLatency$ after sending the request (t_{maxME} before now). Note that $MsgLatency$ only provides an upper bound to message delivery latency, therefore the fastest time for a message to be delivered could be 0.

By design y holds exclusive rights to the resource only after sending its request to every entity in the sending area and receiving feedback from every of these entities (c.f., Section 5.4). Since y's request is delivered to everyone in the sending area at most $t_{\text{maxME}} + MsgLatency$ ago (condition 3; between time t_0 and t_1 in Figure 5.3). By condition 1a (entity x is in the sending area during this period), entity x must have heard y's message which contradicts condition 1b.

The calculation of sending area follows: the earliest time for which x could have receive y's request is $t_{\text{maxME}} + MsgLatency$ ago (i.e., before t_1, at t_0). In order for y to access the resources after $t_{\text{maxME}} + MsgLatency$ (at t_2), y's request must be delivered to $2 * (t_{\text{maxME}} + MsgLatency)$.

5.3.3 Race conditions

An entity's behavior when it receives a conflicting request from another entity is dependent on when it receives the message. There are three situations:

1. Before sending its own request

   When an entity receives a request (independently of whether the request conflicts with the entity's own request), the request/feedback protocol (c.f., Section 5.4) accepts the request as valid and stores the received request in the entity's situation picture. The entity does not formulate requests for resources that conflict with existing requests stored in its situation picture.

2. After sending its request and before committing to crossing

   This race-condition is handled by the request/feedback protocol (c.f., Section 5.4). The protocol either accepts or ignores the conflicting request; if the protocol accepts the conflicting message to be valid, the entity cancels its intention to access the shared resources. The entity can send a new request after canceling its intention; that is, still ensuring that the new request is not in conflict with existing requests.

3. After committing to access the shared resources

   An entity ignores any conflicting requests received after it has committed to access the shared resources; this is because the request/feedback protocol ensures that the sending entity will cancel its intention to access the shared resources.

5.3.4 Intersection collision avoidance scenario

This section presents the calculation of the sending area and lurking time for intersection collision avoidance scenario. In this scenario, the intersection is modeled as a grid (Figure 3.2) with each grid square representing a uniquely identified resource, $r_{x,y}$. A vehicle crossing the junction is required to obtain either a schedule for or mutual exclusion on the use of the set of squares (resources) that it will traverse.

A vehicle requests the required resources using the request/feedback protocol (c.f., Section 5.4); it proceeds to cross only if the protocol succeeds in obtaining a schedule or mutual exclusion. However, in the case where the vehicle cannot obtain the required resources or the request/feedback protocol fails due to communication failures, the vehicle must come to a complete stop before entering the junction (the LLFSM). As a vehicle requires time and distance to stop, the vehicle, x, must decide by its decision point, dec_x, whether it can cross or not.

A vehicle, x, with initial velocity, v_x, and uniform deceleration, d_x, has a breaking time of $brk_x = \frac{v_x}{d_x}$. Entity x's decision point expressed in time before arriving at the junction is brk_x.

The formula for calculating the distance traveled, d, with v_o the initial velocity, a the acceleration and t the time, is:

$$d = v_o t + \frac{1}{2} a t^2$$

In the intersection collision avoidance scenario, to obtain the breaking distance, $v_o = v_x$, $t = brk_x = \frac{v_x}{d_x}$, $a = -|d_x|$ (d_x is a positive value for deceleration).

$$dec_x = v_x \frac{v_x}{d_x} - \frac{1}{2} d_x \frac{v_x^2}{d_x^2} = \frac{v_x^2}{2d_x}$$

Therefore, the decision point, measured in distance from the junction is $dec_x = \frac{v_x^2}{2d_x}$. That is, the request/feedback protocol should be completed dec_x away from the junction. As defined, the request/feedback protocol requires $MsgLatency$ time to complete, therefore vehicle x is required to start the protocol at least $MsgLatency$ before arriving at its decision point.

The following two sections describe the implementation of the intersection collision avoidance scenario using the CwoRIS pattern implemented for scheduling and mutual exclusion respectively.

Before starting the scheduling and mutual exclusion sections, we define a property used in both sections: Given that vehicle x is $dist_x(t)$ from the junction, and the maximum speed for traveling in the scenario is $v_{\max}$, the earliest time vehicle x can arrive at the junction is bounded by:

$$earliestArrival_x(t) = \frac{dist_x(t)}{v_{\max}}$$

5.3.4.1 Implementation with CwoRIS scheduling

This section describes the steps and parameters for an autonomous vehicle to obtain a schedule for crossing the intersection using the coordination scheme. This section first presents the sending area and lurking time required, then its describes the vehicle's actions in various situations.

Obtaining a Schedule in intersection collision avoidance scenario Using the CwoRIS pattern, a node sending a request must ensure that the message has been delivered to everyone that might access the shared resources during the same time period. When applied to vehicles scheduling in this scenario, a vehicle must deliver its message to every other vehicle that might arrive at the junction before its crossing is completed.

> *Intersection Sending Area:* Given a set of required resources $R_x = \{\langle r_{x,y}, t_s, t_e \rangle\}$, and $v_{\max}$ as the vehicles' maximum speed, a vehicle, x, must deliver its request to every vehicle within $sendArea_x(t) = (\max_{z \in R_x}(z.t_e) - t) * v_{\max}$ of the junction in order to cross, where t is the sending time.

The sending area for this scenario just converts the general time-based sending area for scheduling (Section 5.3.2.1) into a geography-based sending area using vehicles' maximum speed in the junction.

> *Intersection Lurking Time*: Given a policy that defines the longest period for which a resource can be reserved is t_{maxHold}. In order for a vehicle x to ascertain that no other entities is holding on to a resource γ after $earliestArrival_x(t)$, x must has been lurking for $t_{\text{maxHold}} - earliestArrival_x(t)$ without hearing a conflicting request.

Similarily, the intersection lurking time is a direct translation from Lemma *Lurking Time*. The parameters of the original Lemma to the definition above has the relationship $earliestArrival_x(t) = t_{\text{maxHold}} - \delta$. Therefore, the parameters mapping are:

Original	intersection collision avoidance scenario adaption
$t_{\text{maxHold}} - \delta$	$earliestArrival_x(t)$
δ	$t_{\text{maxHold}} - earliestArrival_x(t)$

Vehicle Behavior in Junction Applying the coordination scheme, the process for the vehicle, x, to obtain a schedule is as follows:

1. Start listening and building up an image of the situation t_{maxHold} before arriving at the junction.

2. Send a resource request to an area of size $(\max_{z \in R_x}(z.t_e) - t) * v_{\max}$, which is to be delivered by $MsgLatency$, any time after lurking for at least $crossingTime_x$ and before $MsgLatency + brk_x$ from the junction (t is the current time).

   Vehicle formulates its request using the situation picture; searching for a time that is not in conflict with existing requests for its required resources. In the case where its requested resources are not available (exist only after t_{maxHold}), the vehicle delays sending its request until they are available.

3. Deliver own request and receive communication feedback, at least brk_x before arriving at the junction.

   In the event where the vehicle arrives at its decision point, dec_x, without acquiring exclusive use to its required resources, it must break and stop so as to not enter the junction.

4. Decide whether to commit to crossing at dec_x, brk_x from the junction.

   In the case where a vehicle accepts a conflicting request, it reschedules using the internal simulation method (Section 5.5.2) and send a new request.

   When the request is delivered to every vehicle in the sending area without receiving a conflicting request, the vehicle commits to crossing. The vehicle crosses the junction using only the resources specified in R_x.

5.3.4.2 Implemented with CwoRIS mutual exclusion

Unlike the scheduling scenario in which vehicles must be certain about when they can complete crossing, vehicles in the mutual exclusion scenario do not need to know how long they require to cross the junction - the mutual exclusion scenario allows vehicles to access the critical section for an unbounded period.

The unbounded period, coupled with the presence of dynamic participants and unreliable communication, complicates the process to release resources after the vehicles complete their crossing. If messages are used to inform other vehicles about the resources release, the messages could be lost; after the sending vehicle leaves the system, a deadlock could occur. Bouroche et al. [2006] assume that the time that a vehicle needs to traverse a junction is bounded, providing a method for other vehicles to time-out resource allocations; time-outs are unsuitable in the mutual exclusion scenario due to the assumption of unbounded access period to the shared resources. In contrast, CwoRIS pattern's mutual exclusion version assumes the usage of resource sensors to detect entities using the resources; the intersection scenario assumes junction sensors are available for vehicles to check whether there are any vehicles currently crossing the junction.

This section describes the steps and parameters for an autonomous vehicle to obtain exclusive access for crossing the intersection using the coordination scheme. This section first presents the sending area and lurking time required, then its describes the vehicle's actions in various situations.

Obtaining exclusive access in intersection collision avoidance scenario Using the mutual exclusion version of the coordination scheme (Section 5.3.2.2) a vehicle must arrive at the junction within t_{maxME} after its request is approved or it has to abort its crossing. The time required for a vehicle, x, to travel with uniform velocity, v_x, from its decision point, dec_x, to the junction is (note: $dec_x = \frac{v_x^2}{2d_x}$ was defined at the beginning of the section):

$$\frac{dec_x}{v_x} = \frac{v_x}{2d_x}$$

Assuming the minimum deceleration (intentional breaking) for all vehicles is d_{min}, the maximum time for any vehicle to travel from its decision point, without breaking, to the junction such that it can be picked up by the junction sensor is

$$t_{\text{maxME}} = earliestArrival_{\text{decisionPoint}}(t) = \frac{v_{\text{max}}}{2d_{\text{min}}}$$

Applying t_{maxME} to the generic definitions:

Intersection Sending Area-Mutual exclusion: Given a policy that defines that a vehicle must either start using the resource within $t_{\text{maxME}} = \frac{v_{\text{max}}}{2d_{\text{min}}}$ after its resource acquisition or give up accessing the resources, a vehicle, x, needs to send its transfer message to every vehicle within $sendArea_{\text{ME}} = 2(t_{\text{maxME}} + MsgLatency) * v_{\text{max}}$ surrounding the junction in order to cross.

The sending area is a direct adaption from the generic definition where the sending area is defined as geographical area.

Intersection Lurking Time-Mutual exclusion: If vehicle, x, is in $sendArea_{\mathrm{ME}}$, heard no transfer message for a period of $t_{\mathrm{maxME}} + MsgLatency$ and the junction sensors do not register any vehicles currently crossing the junction, then there are no vehicles holding a valid request for crossing.

The intersection lurking time definition is exactly the same as that of the general definition.

Vehicle Behavior in Junction Applying the coordination scheme, the process for a vehicle, x, to obtain exclusive access to the junction is as follows:

1. Start lurking and building up an image of the situation $sendArea_{ME}$ from the junction.
2. When vehicle x is $MsgLatency + brk_x$ before reaching the junction, it checks that there is no conflict before sending a transfer message to an area of $sendArea_{\mathrm{ME}}$, which is to be delivered by $MsgLatency$.

   The vehicle checks the situation picture and junction sensors for conflicts. The vehicle can send its request only when there are no conflicting requests recorded in the situation picture and there are no vehicles on the junction sensors. If a conflict is detected, the vehicle delays sending its request.

   The vehicle sent its request after the junction sensors is clear and the recorded conflict in situation picture is at least after t_{maxME}.
3. Deliver own request and obtain communication feedback brk_x (time) before reaching the junction.

   In the case where a vehicle accepts a conflicting request, it resends another request as described in Step 2. If the vehicle arrives at its decision point, dec_x (location), without acquiring exclusive use to its required resources, it must break and stop so as to not enter the junction.
4. Decide whether to commit to crossing at its decision point.

   If the vehicle's request is delivered to every entity within the sending area and it did not receive a conflicting request, the vehicle may commit to crossing; at which case, it must start arrive at the junction before t_{maxME}.

5.3.5 Summary

In summary, the four steps of the coordination scheme are: i) lurking, ii) resource request sending, iii) request delivery and feedback receipt, and iv) decision making. This section presented various definitions (i.e., decision point, lurking time, sending area) for the implementation of the coordination scheme. The section also demonstrated the application of the coordination scheme to the intersection collision avoidance scenario.

5.4 Request/feedback Protocol

The request/feedback protocol belongs to the second and third step of the coordination scheme, the protocol is designed to handle race conditions and communication errors while obtaining schedules and mutual exclusion. The request/feedback protocol uses the lower-layer communication protocol to multicast the request to the sending area (entities that might access the requested resources at the same time). On the receiver side, after the lower-layer protocol receives a message and the request is delivered to the receiver's request/feedback protocol, then an acknowledgment is replied automatically. When the receiver's request/feedback protocol receives a delivered request, it may either *accept* or *ignore* the request.

> *Definition (ignores)*: A node **ignores** a request when it can determine that the sender of the request will not act on the request.
>
> If a node does not ignores a request, it **accepts** the request.

In Section 5.4.2 the request/feedback protocol is enhanced with an inference mechanism that allows a node to determine that a sender will not act on the message, thereby allowing a node to ignore a transfer message.

$MsgLatency$ after sending a request, the sender's lower-layer communication protocol aggregates the received acknowledgments, the set of acknowledgment's node ID are returned to the request/feedback protocol. The request/feedback protocol determines whether the node's request is valid.

> *Definition (valid)*: A node's request is said to be **valid** when the node can exclusively access the requested resources (at the time specified in its request).

Two incidents may prevent a node from obtaining a valid request; communication errors and race conditions. Communication errors happen when there requests or acknowledgments are lost. Race conditions happen when two nodes send requests for conflicting resources. A request for conflicting resources is named conflicting request for brevity:

> *Definition (conflicting request)*: A **conflicting request** refers to a transfer message, m_r, from another node requesting access to a set of resources, R_r, such that the set of resources required by the receiving node, R_o, is in conflict ($R_o \cap R_r \neq \emptyset$) (see Section 3.4.2.3).

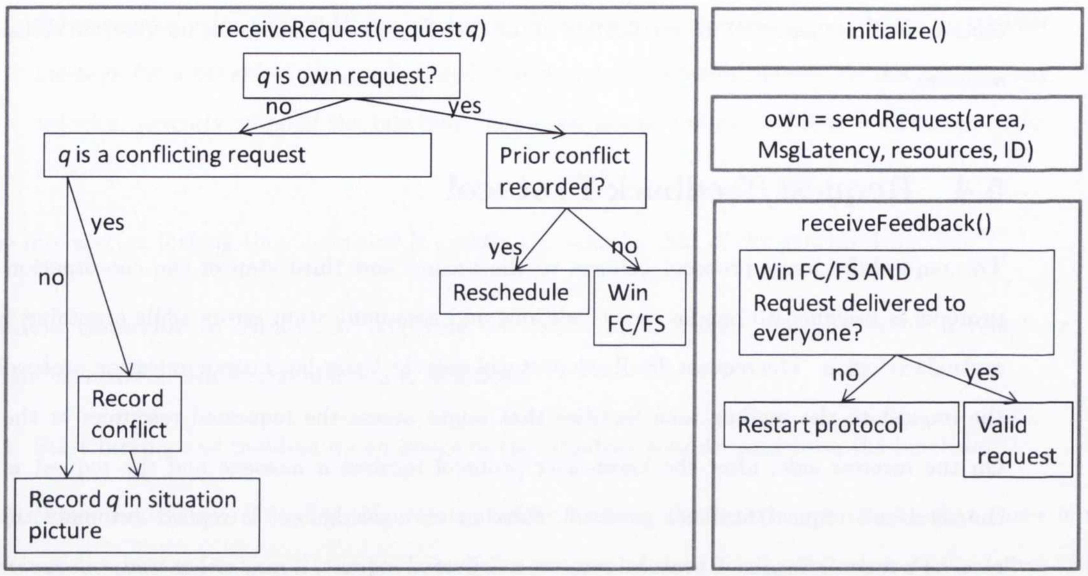

receiveRequest(request *q*)
q is own request?
no
yes
q is a conflicting request
Prior conflict recorded?
yes
no
yes
no
Reschedule
Win FC/FS
Record conflict
Record *q* in situation picture
initialize()
own = sendRequest(area, MsgLatency, resources, ID)
receiveFeedback()
Win FC/FS AND Request delivered to everyone?
no
yes
Restart protocol
Valid request

Figure 5.4: The basic protocol

The next sub-section describes the basic request/feedback protocol which determines race winners by using only the first-come/first-served mechanism. The following sub-section describes our request/feedback protocol enhanced with inference and priority. Finally, Section 5.4.3 presents the protocol enhanced with various β-constraints defined for scheduling.

5.4.1 The basic protocol

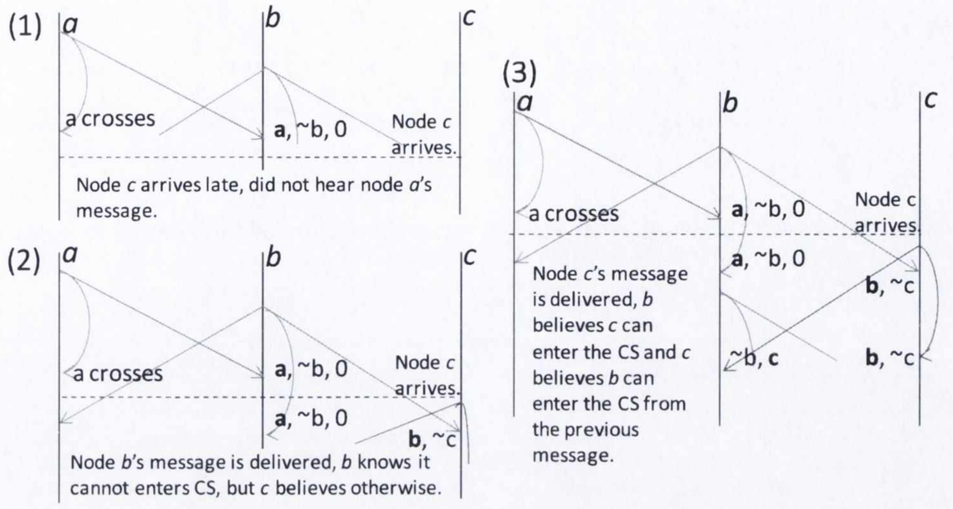
Figure 5.4 shows the basic protocol composed of four functions : *initialize*(), *sendRequest*(), *receiveRequest*() and *receiveFeedback*().

initialize() is used to provide some initial values to variables (e.g., setup the initial situation picture) when the node starts lurking.

sendRequest(*area*, *MsgLatency*, *resources*, *ID*) is used when a node with a unique *ID* sends a request to other nodes within an *area* for some set of *resources*, for which it requires feedback on the delivery of the request within *MsgLatency*.

receiveRequest(*request*) is invoked when the node receives a request. Received requests are either other nodes' transfer requests or the node's own request being delivered.

In the basic protocol, a node accepts all received transfer requests (left sub-tree of *receiveRequest*() in figure 5.4) as valid. In the algorithm, a *situation picture* is used to record all current requests to facilitate scheduling algorithms; a newly-received request replaces an old request from the same node. A node whose own request is delivered without it having received any prior conflicting request wins the first-



(1)
a
b
c
a crosses
a, ~b, 0
Node c arrives.
Node c arrives late, did not hear node a's message.
(2)
a
b
c
a crosses
a, ~b, 0
Node c arrives.
a, ~b, 0
b, ~c
Node b's message is delivered, b knows it cannot enters CS, but c believes otherwise.
(3)
a
b
c
a crosses
a, ~b, 0
Node c arrives.
a, ~b, 0
b, ~c
~b, c
b, ~c
Node c's message is delivered, b believes c can enter the CS and c believes b can enter the CS from the previous message.

Figure 5.5: Deadlock in late comer scenario.

come/first-serve race; if it received a conflicting request before its own request is delivered, the node did not obtained exclusive access to the shared resources. It then reschedules another request (right sub-tree of *receiveRequest*() in figure 5.4).

receiveFeedback() is invoked once the lower layer communication protocol times-out after $MsgLatency$. A node holds a valid request only when it wins the first-come/first-served (FC/FS) race and its request is delivered to every node that might access the resources at the same time. When a node does not hold a valid request it restarts the protocol by sending another request.

The basic protocol is able to ensure mutual exclusion, but unless we assume the 'total collision' model, late-comers might create a deadlock. Figure 5.5 shows a scenario where a late-comer, node c, did not receive a message previously sent by node a, and enters into a deadlock with node b. The scenario assumes that the three nodes are requesting conflicting resources.

In the diagram, a short-hand representation of a node's situation picture is shown at the point where a message is delivered. For example, in figure 5.5.1, after node b receives the message from node a, the situation picture at node b is "**a**, ~b, 0". The **bold** font represents the fact that b accepts the last message received from node a. A '~' before the node represents that the node ignores the last message received from that node; when applied to itself (in this example, node b), the node knows that it cannot access its required shared resources. '0' represents the fact that the node's situation picture does not record the existence of another node; node b may not know anything about node c because they have not communicated.

1. Figure 5.5.1

   (a) Node a sends a message msg_a, requesting access to the shared resources.

   (b) Node b sends a message msg_b, requesting access to the shared resources (before it receives msg_a).

   (c) msg_a, is delivered at a and b, b accepts msg_a and cancels its intention to access the shared resources, a accesses the shared resources.

   (d) Node c arrives without having received a's message.

2. Figure 5.5.2

   (a) Node c sends a message msg_c, requesting access to the shared resources.

   (b) msg_b is delivered to nodes a, b and c; c accepts msg_b and cancels its intention to access the shared resources.

   The situation picture in b and c are now inconsistent, but neither will access the shared resources.

3. Figure 5.5.3

   (a) Node b, who cannot proceed, sends a new request $msg_{b1}$for the same resources at a later time.

   (b) msg_c is delivered to b, b accepts msg_c and cancels its intention.

   By now, a circular wait has been formed between node b and c; node b accepts node c's request is valid and backs off, but node c accepts node b's request is valid and backs off too. Note that this is safe in that no entity accesses the shared resources at the same time. However, without a mechanism to either prevent or break the deadlock, node b and c face starvation, any subsequent nodes that arrive and wait on b and c will also be deadlocked.

5.4.2 Protocol with inference and priority

In this section, the basic protocol is enhanced with a mechanism that allows a node to ignore a received message. In this enhancement, an entity, x, may infers about another entity's, y's, intention after it has received a message from y. In addition, if x acknowledged to y's request, then y may also infers that x knows about y's intention. This enhancement detects a deadlock by first building up a common knowledge; the knowledge of i) others' intention, ii) whether others' knows the entity's intention, and iii) whether these intention are in conflict. After such a deadlock is detected, the enhanced protocol breaks the circular waits by allowing a unique entity to ignore other partys' intention.

In implementation, a node may ignore a received message (breaks a deadlock) if it has had a previous message delivered to that message's sender that will result in the sender backing off. A unique priority is associated with each node to decide who may breaks the circular waits. The unique priority can either be derived from the node identifier or generated (Note: the priority can increase from one request to the next; it may not decrease) and sent as part of the request message.

In order to implement the protocol with inference and priority, there are several minor additions to the basic protocol. Two further data structures are introduced on each node (in addition to the situation picture): *prevRequest* and *prevMembers*. *prevRequest* remembers the previous request that the node sent. The *prevMembers* data structure records the set of node identifiers from which an acknowledgment for its *prevRequest* was received.

> *Definition (knows intention)*: A node, x, is sure that another node, y, **knows** its **intention**, if x has received an acknowledgment from y for its last message.

By construction, *prevMembers* is a list of nodes that know x's intention. Note that although node y may know x's intention, it may chose to ignore it.

The protocol checks whether received requests can be ignored by checking whether three conditions are satisfied:

1. the receiver is sure that the sender knows its previous intention (i.e., the sender of the received request is in *prevMembers*), and
2. the receiver's previous intention and the sender's message have a resource conflict, and
3. the receiver has higher priority. (Only higher priority nodes can ignore messages from lower priority nodes.)

Figure 5.6 shows the algorithm for the request/feedback protocol with inference and priority.

The three main points that implement the inference and priority are shown in figure 5.6, box A, B and C:

1. Figure 5.6, box A. The protocol prepares the data required for inference after the protocol fails to acquire exclusive use of the critical section. Nodes who have acknowledged the delivered message are recorded in the *prevMembers* structure and the delivered message is remembered as the *prevMessage*.
2. Figure 5.6, box B. The protocol checks whether to ignore the message by checking the three conditions:

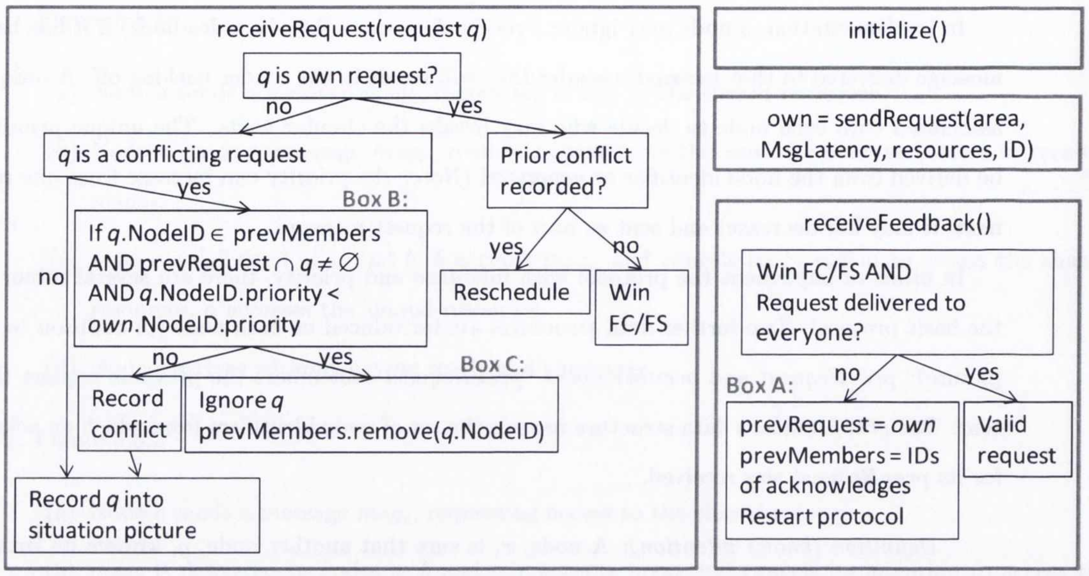

receiveRequest(request q)
q is own request?
no
yes
q is a conflicting request
Prior conflict recorded?
yes
Box B:
If q.NodeID ∈ prevMembers AND prevRequest ∩ q ≠ ∅ AND q.NodeID.priority < own.NodeID.priority
no
yes
no
Reschedule
Win FC/FS
no
yes
Box C:
Record conflict
Ignore q prevMembers.remove(q.NodeID)
Record q into situation picture
initialize()
own = sendRequest(area, MsgLatency, resources, ID)
receiveFeedback()
Win FC/FS AND Request delivered to everyone?
no
yes
Box A:
prevRequest = own prevMembers = IDs of acknowledges Restart protocol
Valid request

Figure 5.6: Decision when a node received a request

(a) $q.NodeID \in prevMembers$: whether q's sender has heard the node's previous request.

(b) $prevRequest \cap q \neq \emptyset$: whether the previous request conflicts with the received message.

(c) $q.NodeID.priority < own.NodeID.priority$: whether the node has higher priority then$q$'s sender.

If all three conditions are satisfied, a node can safely ignore the received message (because it knows that the other node has accepted its request). Conditions (a) and (b) form the inference part of the protocol to detect deadlock while condition (c) forms the priority part of the protocol to break deadlocks.

3. Figure 5.6, box C. The node ignores q by not recording q in its situation picture.

5.4.3 Protocol with β-constraints

Section 3.4.3.2 presented the β-constraints on 'sequence dependent' and 'job-family' setup time, 'preemption', 'blocking', 'no-wait' and 'breakdown'. This section shows how the request/feedback protocol can be modified to support these constraints.

5.4.3.1 No-wait

As mentioned, a resource-entity schedule exhibits the *no-wait* requirement when an entity is not allowed to wait between access to two successive resources.

During implementation, this implies that an entity must 1) reserve a set of resources that are either continuous or overlapping in time, and 2) atomically reserve the set of resources. The *situation picture* data structure records which resources might be unavailable. Using the situation picture, a local search algorithm finds a set of available resources that fits the continuous constraint (Section 5.5 presents this local scheduling). Requests are made by specifying the required resource basket (Section 3.4.2.2) which implements the required atomic resource reservation.

5.4.3.2 Blocking

The presence of the *blocking* constraint implies that a completed job (e.g., vehicle) has to remain on the upstream machine (e.g., grid square) preventing (i.e., blocking) that machine from working on the next job (e.g., grid square from accepting the next vehicle). As a result of the blocking constraint, an entity may have to use a resource for an unbounded period of time. The CwoRIS pattern supports implementation of the blocking constraint via the protocol's mutual exclusion version.

5.4.3.3 Preemption

The *preemption* constraint implies that it is not necessary that an entity completes its resource use even though it has acquired exclusive access to it. A higher priority entity may request (preempt), and the lower priority entity must give up, the resource. Preemption is implemented in the request/feedback protocol with a two-part extension: communicate the intention to preempt resource usage, and preempted entity's behavior.

Communicate preemption intention In the absence of a central scheduler, entities have to coordinate preemption in a distributed manner. There are essentially three parties involved: a) the high-priority entity sending the preemption request, b) the low-priority entity that must give up (or minimize its usage of) the resources it has acquired exclusive access to, and c) the other entities who do not hold any exclusive resources that the high-priority entity requires.

Figure 5.7 shows the extended *receiveRequest*() function from the request/feedback protocol in order to support preemption constraint. The higher-priority entity whose request preempts another's request (Case A in the figure) records the preempted request. The lower-priority entity who receives the preemption request (Case B in the figure) performs its preempted behavior (as detailed in the following sub-section) and records the conflict. Due to the absence of consensus, other entities who receive a request would not know about the preemption (Case C in the figure); these entities act conservatively and accept both requests (by recording them in the situation picture).

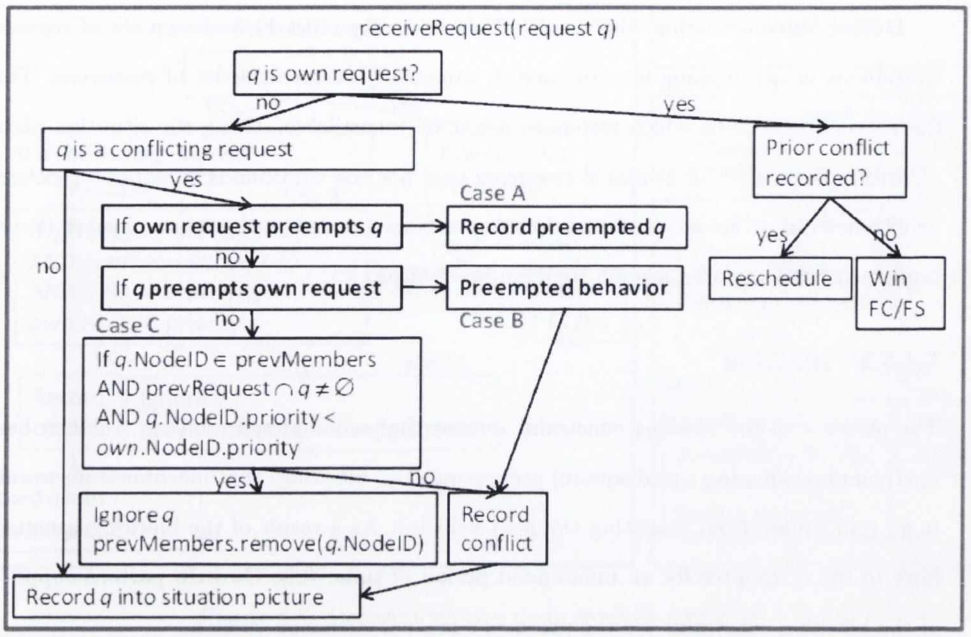

receiveRequest(request q)
q is own request?
no
yes
q is a conflicting request
yes
Prior conflict recorded?
Case A
If own request preempts q
Record preempted q
yes
no
no
no
If q preempts own request
Preempted behavior
Reschedule
Win FC/FS
Case C
no
Case B
If q.NodeID ∈ prevMembers AND prevRequest ∩ q ≠ ∅ AND q.NodeID.priority < own.NodeID.priority
yes
no
Ignore q prevMembers.remove(q.NodeID)
Record conflict
Record q into situation picture

Figure 5.7: ReceiveRequest function with preemption β-constraint

Preemption Behavior When an entity holds a valid request for some resources, it may have committed to the use of these shared resources. In particular, an entity that has crossed its decision point would be in a behavior that cannot avoid accessing the shared resource.

In the event where a lower-priority entity is preempted before it reaches its decision point, it simply releases the resources held and cancels its intention to access the shared resources. When a preemption happens after the lower-priority entity's crosses its decision point, it may be required to store its current situation, release the usage of its resources and plan for its access to the shared resources on a later schedule; implementation of the lower-priority entity handling preemption is dependent on the actual scenario.

Using the preempted records (Figure 5.7, Case A), a higher-priority entity may know whether the lower-priority entities have passed their decision points using either i) the scenario sequence constraints (see Section 5.4.3.6 for an example), or ii) the contract with feedback protocol described in Bouroche's thesis [2007]. Consequently, the higher-priority entity who successfully preempted other entities must allocate the necessary time for those preempted entities to give-way before it access the preempted resources.

5.4.3.4 Breakdown

Resource *breakdowns* imply that a resource may not be continuously available. In general, the handling of breakdowns involves two parts: detection and reaction. Breakdown detection involves the use of sensors or communication to inform entities that the resource is not available. This work does not handle the actual detection mechanisms; instead it assumes that the time required for reliable detection of a breakdown is bounded by t_{detect}. Reaction to breakdowns involves entities changing their behavior to avoid using the unavailable resources. This work assumes the time required for an entity to change its behavior is bouned by t_{react}.

In Dresner and Stones'[2008a] intersection scenario, detection is achieved by having the broken vehicle send a signal to the centralized manager, which then informs all vehicles. Vehicles react by not entering the junction or switching on their on-board sensors to perform 'individual collision avoidance' if they are already in the junction. Their work does not provide time allowances between vehicles for breakdown detection and reaction; a vehicle which is crossing close behind a vehicle which breaks down might suddenly find that it does not have time to react to the event, resulting in a collision. This work's implementation of intersection collision avoidance trades efficiency for safety by providing an allowance of $t_{\text{detect}} + t_{\text{react}}$ between resource reservations. t_{detect} is assumed to be a constant and is added to each resource usage time and t_{react} is dependent on the situation. The following section describes how this work handles different reaction times.

5.4.3.5 Sequence dependent & job families setup times

Section 3.4.3.1 defined the specification of sequence dependent and job-family setup time using $R_a \cap_{\text{setup}} R_b$. As mentioned, *sequence dependent setup time*, s_{jk}, is the time required to prepare a resource between the processing of entities j and k. If the setup time between entities j and k depends on the resource i, then the subscript is included, i.e., s_{ijk}.

For example, a vehicle's breaking time (reaction time) is proportional to its speed and deceleration, and the traction of the road (resource dependent). Adding another vehicle to the picture, the available time for a vehicle to react is dependent on both vehicles' speed, deceleration and direction; vehicles traveling in opposite directions have a smaller reaction time available than vehicles traveling in the same direction. In this example, the reaction time between two entities is dependent on both entity's parameters (the subscripts j and k in the definition).

Job-families group entities into families and setup times are defined between any two families. Illustrating using the same vehicular example: vehicles crossing a junction with the same origin and destination can be categorized into the same family. The setup times between families can then be calculated by assuming a maximum speed in the junction and fixed deceleration rate.

In order to implement sequence dependent setup or job families, three modifications are required:

1. An entity includes in its request the relevant parameters (for sequence dependent setup) or the family-identifier (for job-families). The *receiveRequest*() function in the request/feedback protocol stores these parameters/family-identifiers together with the received request into its situation picture when the request cannot be ignored (Figure 5.7).

2. A *x.calculateSetup*(*y*'s request) function is defined to return the setup time between entities x and y. The receiving entity, x, can then calculate the required setup time using the parameters/family-id stored in the situation picture.

3. The normal conflict relation ($R_a \cap R_b \neq \emptyset$) in the receive/feedback protocol and the scheduling protocol is replaced with the conflict relation defined for setup time ($R_a \cap_{\text{setup}} R_b \neq \emptyset$). (Note: the definition below was presented in Section 3.4.3.2; it is repeated here only for easy readability.)

   Resource conflict with setups: Two sets of resource tuples, R_a and R_b, with their respective setup times, s_{ab} and s_{ba}, are in conflict, $R_a \cap_{\text{setup}} R_b \neq \emptyset$, when some resources appear in both sets and their setup time added to resource required time overlaps.

$$\begin{aligned} R_a \cap_{\text{setup}} R_b \neq \emptyset := \quad & \exists x \in R_a, y \in R_b \bullet \\ & x.r = y.r \wedge \\ & (x.t_s - s_{ryx}) < y.t_e \wedge \\ & (y.t_s - s_{rxy}) < x.t_e \end{aligned}$$

5.4.3.6 Intersection collision avoidance scenario enhanced with β-constraints

This section describes two extensions of the intersection collision avoidance scenario using the β-constraints described. In particular, it examines a vehicle sequence for entering the junction and the prevention of vehicle's collision in the event of vehicle breakdowns.

Vehicle sequence for entering a junction In the scenario (Figure 5.8), a vehicle behind, y, may prevent another vehicle in front, x, from crossing if y obtained some of the resources that x requires, In addition, y cannot move because x is in-front of it, forming a deadlock. This scenario has a constraint on vehicle sequence for entering the junction; there is a sequence constraint regarding vehicles' exiting the road scenario (Section 3.4.3 presents the sequence constraint). Note: this scenario is about the sequence in which a vehicle leaves the road scenario and enters the junction; it does not matter whether the junction scenario is implemented with the scheduling or the mutual exclusion versions.

Two example cases for which these out-of-sequence resource reservations may happen are:

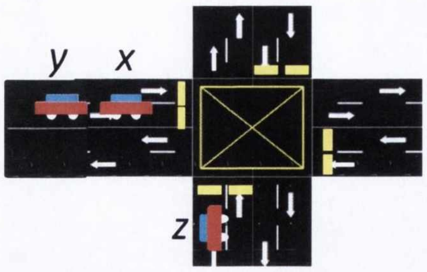


Figure 5.8: Two vehicles crossing in sequence.

- y's message was delivered before x's message

  In this case, y wins the first-come/first-served race and has exclusive rights to the crossing resource.

- x's message was delivered before y's message

  In this case, y believes that x is crossing and acquires the next set of resources. However, x could be prevented from crossing because of either communication failure (x's message is not delivered to everyone) or it has lost a race to another vehicle requesting a conflicting resource (e.g., vehicle z). Consequently, x requires the next set of resources, which y is holding.

In either case, the result is that both vehicles cannot cross until y's request expires and x wins the next resource request; which is inefficient.

This work models the above problem as a preemption β constraint: a resource that supports preemption (small yellow squares in Figure 5.8) is included at each road's lane exit (a lane's entrance into the junction). Note that all other resources in the junction are non-preemptive. Using the request/respond protocol with preemption, vehicles request the relevant preemptive resources with all other required resources in a resource basket.

As presented in Section 5.4.3.3, preemption implementation requires a two-part extension:

1. Communicate intention

   In this scenario, vehicle x's request preempts vehicle y's request when i) x's request and y's request hold the same preemptive resource, and ii) x is nearer to the junction than y.

   (a) The set of preemptive resources are fixed in this scenario.

   (b) By including an entity's distance from the junction in its request message, the receiving entity can use the request/response protocol (Figure 5.7) to check for the preemption condition.

2. Preempted behavior

   In this scenario, when a vehicle in-front preempts another vehicle (at the back) of its resource usage, by the scenario's resource usage sequence constraints, the vehicle at the back is dependent on the vehicle in-front, therefore, it could not have started crossing the junction. As such, the preempted vehicle only need to record that it has received a conflict. The rest (i.e., slowing down at its decision point) will be handled by the coordination scheme.

Prevent collisions during vehicle breakdown A vehicle may break down in the intersection and use resources that it did not reserve. Other vehicles that have acquired exclusive access to those resources may crash into the broken vehicle. This section presents an example for the elimination of collisions due to vehicle breakdowns by trading efficiency for safety: vehicles drive with bigger inter-vehicle gaps, so as to provide enough allowances for vehicles to react during a breakdown. This solution assumes that a brokendown vehicle slows to a stop and does not skid into other lanes/resources for simplicity.

Applying the β-constraint for resource breakdowns (Section 5.4.3.4), a vehicle should allocate an allowance of $t_{\text{detect}} + t_{\text{react}}$ time for reacting to resources unavailability. The maximum time required to detect a breakdown, t_{detect}, is defined as a constant and the time for a vehicle to react, t_{react}, is modeled using the job families β-constraint.

We use the observation that vehicles traveling in the same direction (in the same family) have shorter stopping distances than vehicles traveling in different directions (from different families) to define t_{react}. Vehicles in the scenario belongs to one of the 12 families represented by vehicles' origins and destinations (four origin directions and three destination directions (no U-turns)). As defined in Section 5.4.3.5, the three modification steps are:

1. A vehicle entity includes in its request its family-identifier, and the request/feedback protocol records the family-identifiers of each vehicle (based on received messages) into its situation picture.
2. A *x.calculateSetup*(*y*'s request) function is defined to return the setup time between two entities: x and y. This can be implemented as a lookup table that stores the pre-calculated 12^2 permutations in each vehicle.
3. The normal conflict relation ($R_a \cap R_b \neq \emptyset$) in the receive/feedback protocol and the scheduling protocol is replaced with the conflict relation defined for setup time ($R_a \cap_{\text{setup}} R_b \neq \emptyset$) (Section 3.4.3.2).

In addition, both the lurking time and the sending areas extended by $t_{\text{detect}} + \max(t_{\text{react}})$ and $(t_{\text{detect}} + \max(t_{\text{react}})) * v_{\max}$ respectively.

5.4.4 Summary

This section presented the request/feedback protocol for entities to send and receive request in step two and three of the coordination scheme. The basic protocol is able to ensure mutual exclusion, but entities may be involved in a deadlock. The basic protocol is then extended with inference and priority to break deadlock. The section also showed the handling of β-constraints in the request/feedback protocol and an example of β-constraints use in the intersection collision avoidance scenario.

5.5 Local Scheduling/rescheduling

As the appropriate scheduling policy is application dependent, this section discusses how a rescheduling policy can avoid livelocks. The following sub-section motivates the requirement for a rescheduled plan by describing an example of a set of nodes involvement in a live-lock due to improper rescheduling plan. After this, the section presents a rescheduling method that prevents live-lock.

5.5.1 Livelock example

In this thesis, a live-lock refers to a condition whereby two or more entities continuously change their internal states in response to requests received, the result is that none of the entities can proceed to access the shared-resources. This example assumes that the entities do not have a strategy for resending a schedule, therefore an entity may choose any schedule as long as it is reasonable; the entity would not schedule for resources that it believes are unavailable.

Figure 5.9 shows a sequence of messages leading to three entities (a, b, and c) being in a live-lock. The live-lock example involves four steps (numbered: A, B, C and D) with each step split into two diagrams (left and right). The left diagram shows the sequence in which entities' requests are sent and delivered. The left diagram also shows the situation picture stored whenever an entity receives a request. In the diagram, the first request from entity x is labeled m_x, and the second request from x is labeled $m_{x'}$ to differentiate between the two requests. When an entry, y, receives a request m_x, the situation picture records that the entity either accepts (x) or ignores ($\sim x$) the request, when the record is applied to the entity's own request it means that it either gave up its message ($\sim y$) or still believes that it has a chance to access the shared resources (y) (e.g., Figure 5.9.C, entity b has a final situation picture of $a', \sim b', \sim c$, meaning that entity b accepts request $m_{a'}$ (a'), ignores request m_c ($\sim c$) and that it ($\sim b'$) may not access the shared-resources). The right diagram shows the time requested for a resource in the entity's request. The four steps are:

Figure 5.9.A The delivery of request m_x after entity a has sent out its request, m_a. The request results in entity a accepting m_x and backing off (i.e., a's situation picture is x,$\sim a$).

Entity b arrives at the scenario and sends its request, m_b, before entity a's request is delivered. When entity a's request is delivered, entity b accepts it and backs off (i.e., b's situation picture is a,$\sim b$).

Entity c arrives at the scenario and sends its request, m_c, before entity b's request is delivered. When entity b's request is delivered, entity c accepts it and backs off, entity a ignores m_b because of a's delivered request (i.e., c's situation picture is b,$\sim c$).

Figure 5.9.B Entity a resends a request, $m_{a'}$ before entity c's request, m_c, is delivered. When m_c is delivered:

- Entity a accepts it because it does not have a conflicting request delivered to c, and entity a backs off by ignoring its own resent message a' (i.e., a's situation picture is $\sim a'$,$\sim b$,c).
- Entity b ignores c's request because it knows that entity c has received its previous conflicting request (i.e., b's situation picture is a,$\sim b$,$\sim c$).
- Entity c has backed off in Figure 5.9.A when it receives entity b's request.

Figure 5.9.C Entity b resends a request, $m_{b'}$, before entity a's resend request, $m_{a'}$, is delivered. When $m_{a'}$ is delivered:

- Entity a has backed off in Figure 5.9.B when it receives entity c's request.
- Entity b accepts it because it does not have a conflicting request delivered to a, and entity b backs off by ignoring its own resent message b' (i.e., b's situation picture is a',$\sim b'$,$\sim c$)..
- Entity c ignores request a' because it knows that entity a has received its previous conflicting request (m_c) (i.e., c's situation picture is $\sim a'$,b,$\sim c$).

Figure 5.9.D Entity c resends a request, $m_{c'}$, before entity b's resend request ($m_{b'}$) is delivered. When $m_{b'}$ is delivered:

- Entity a ignores request $m_{b'}$ because it knows that entity b has received its previous conflicting request (i.e., a's situation picture is $\sim a'$,$\sim b'$,c).
- Entity b has backed off in Figure 5.9.C when it receives a'.
- Entity c accepts $m_{b'}$ because it does not have a conflicting request delivered to b, and entity c backs off by ignoring its own resent message c' (i.e., c's situation picture is $\sim a'$,b',$\sim c'$).

Assuming that these three entities choos their requested resource's usage time and send out their request in some order, they can be involved in a live-lock. While a three-entity live-lock might happen only occasionally, the possibility of encountering a live-lock increases when the number of entities requesting for the same resource increases and the number of messages increase.

5.5.2 Internal simulation on others' behavior

The internal simulation method for preventing live-locks consists of two parts: i) entities gathering near-perfect information about other entities after two round of perfect communication and ii) the entities' scheduling given the near-perfect information.

The following sub-section starts by explaining entities' scheduling with near-perfect information. The second sub-section then explains how the entities may gather the near perfect information required after two rounds of perfect communication.

5.5.2.1 Pseudo centralized

Dresner [2009] uses a centralized intersection manager to schedule entities crossing an intersection. Schermerhorn and Scheutz [2006] presented an agent coordination protocol which assumes that every agent has a reliable world view and each agent internally simulates every agents' behavior to perform coordination. Although Dresner's method is centralized and Schermerhorn and Scheutzs' method is distributed, both of these methods presented some similarities:

1. The entities have perfect information on the resources that other entities require.
2. Using the perfect information either the centralized entity or every entity can perform the same calculations to arrive at the same result (assuming a deterministic algorithm).
3. When all the entities arrive at a single result (thereby providing consensus), a live-lock can be detected.

Similar to Schermerhorn and Scheutzs' method, our pseudo-centralized algorithm requires every entity to perform the same calculation. In contrast, our pseudo centralized protocol does not assume a shared world model. In order for each entity to derive the same world model, the algorithm assumes that every entity has access to i) a method providing a unique ordering of the entities (e.g., by using the entities' arrival sequence, priority), ii) each entity's resources requirement and iii) every entity uses the same protocol and algorithm for rescheduling requests. This rescheduling algorithm takes the approach that given that each entity knows what each other entity requires, and follows the same sequence of calculation, they would

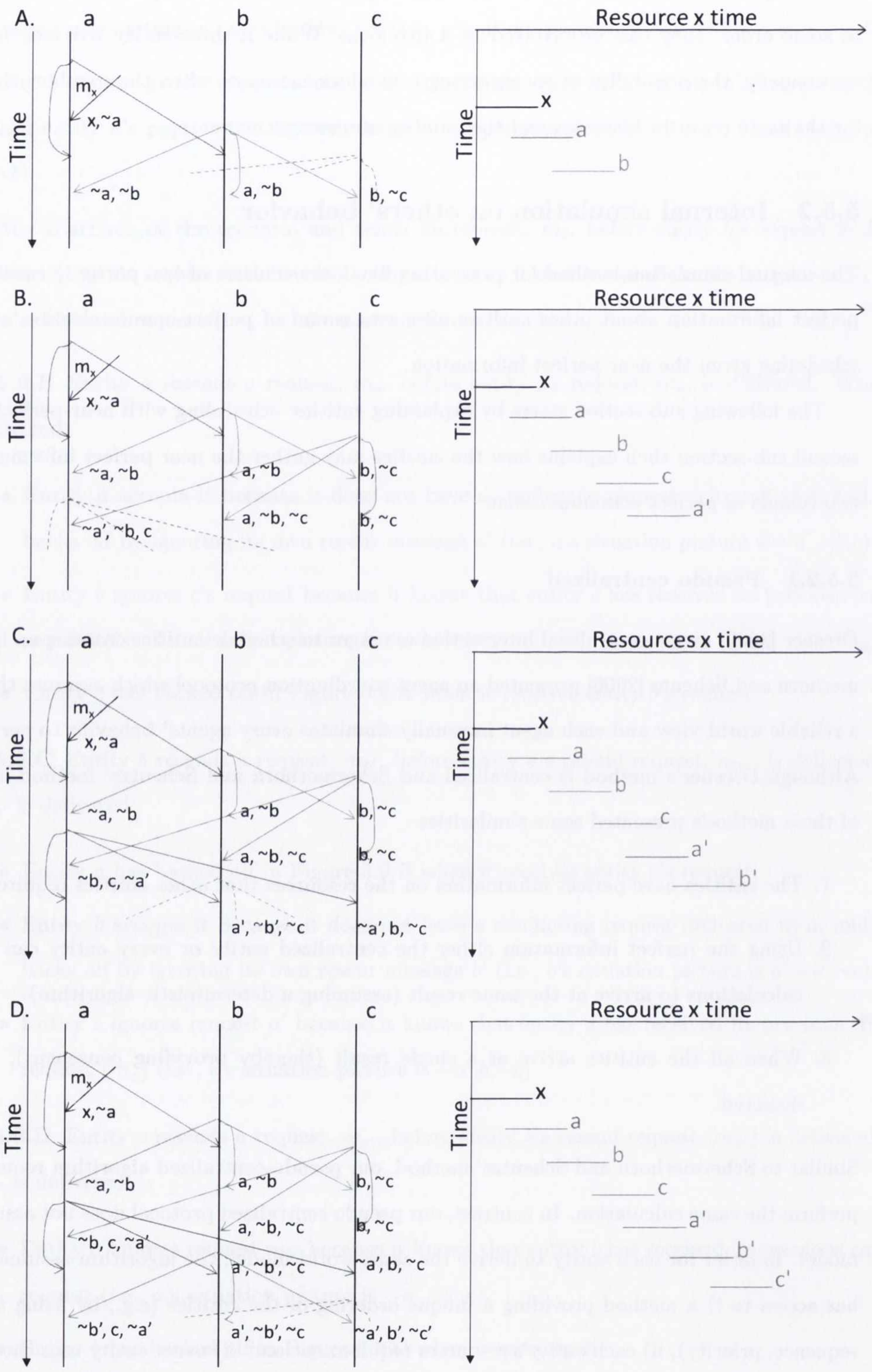


Figure 5.9: A live-lock schedule

be able to arrive at the same results which can ensure that everyone can access their required resources. Every entity uses the following algorithm:

1. Rank all entities by their priority.
2. From the highest priority entity until the entity itself (because the entity does not need to know about lower priority entities).
    (a) Identify the highest priority entity and its resource requirements from the stored situation picture (from the request/feedback protocol).
    (b) Calculate the highest priority entity's next request based on a simulated situation picture.
    (c) Update this calculated request into the simulated situation picture; assuming that the highest priority entity's request is valid.
3. After finding the entities own schedule (last loop in step 2), the entity sends its request using the request/feedback protocol.

Since every entity follows the same deterministic algorithm, they will arrive at the same result. As the calculation for an entity's next request searches only for available resources, each calculated entity's next request is non-conflicting. Therefore, entities will not sent conflicting requests, and they would not be involved in a live-lock.

5.5.2.2 Obtaining near-perfect information

A request message in the request/feedback protocol contains the entity's identifier, its priority and the resources that it requires. After two rounds of perfect[1] communication, an entity who has been listening for other entities' requests for a certain period (lurking time) knows:

1. Who are the entities in the vicinity; entity identifiers in all received requests.
2. A unique ordering of the entities; ranked using the entities' priority (differentiating using the entity's identifiers in the event that two entities' priorities are the same).
3. Every entity's resource requirements; calculated based on the received request. The calculation uses two assumptions i) an entity wants to access its requested resources as early as possible but not earlier than specified in its request and ii) the time required for every resource specified in the request is relative to the earliest resource requirement time. For example, given a resource request

[1]Note: Safety is ensured even without perfect communication, the two rounds of perfect communication is only required for prevention of livelock.

with $\{< r_1, 5, 7 >, < r_2, 3, 6 >, < r_3, 6, 8 >\}$ then the earliest time the request should be scheduled is at time= 3 with the relation $\{< r_1, +2, +4 >, < r_2, 0, +3 >, < r_3, +3, +5 >\}$. Therefore a valid schedule could be at time = 10 such that $\{< r_1, 12, 14 >, < r_2, 10, 13 >, < r_3, 13, 15 >\}$.

This information fulfills the pseudo-centralized algorithm's requirements presented in Section 5.5.2.1.

5.5.3 Summary

This section presents the usage of a rescheduling policy to avoid livelocks. The section presented how a livelock may be present. Then the section shows that given two rounds of perfect communication, an entity knows i) every entity in the vincinity, ii) a unique ordering of the entities and iii) the entities' resource requirements. Assuming that every entity perform the same deterministic algorithm, entities can find a set of resources that satisfy its intention and do not conflict with other entities' request. Therefore, entities will not sent conflicting requests, and will be able to prevent entering a live-lock.

5.6 Summary

This section presented the CwoRIS pattern, which is the final step in Comheolaíocht for deriving entities coordination protocols. The section first presented the communication and sensor requirements followed by an overview of the pattern. It then presented the coordination scheme which defines:

- The decision point where an entity must decide whether to access the shared resources or transition into a LLFSM
- The lurking time which specifies for which duration an entity must listen to know that there is no conflicts reservations
- The area over which an entity must send its request message in order to obtain exclusive use to its required shared resources
- An entities' actions in various situations (e.g., race condition, arriving at the decision point)

The section also presented the request/feedback protocol and its two extensions (inference and score, and β-constraints). The request/feedback protocol is used by entities in the coordination scheme to send and receive requests, it is designed to handle races and communication errors while obtaining schedules and mutual exclusion. The section also presented a local rescheduling protocol for the prevention of live-locks during entities' coordination.

Chapter 6

Evaluation

This thesis presents Comheolaíocht, which provides a systematic approach for the development of scalable and reliable coordination protocols with different time constraints. Comheolaíocht uses our coordination pattern, CwoRIS, that ensures that the entities have exclusive access to shared resources for dynamic participants.

In Comheolaíocht, problems with different time constraints are modeled as entities' exclusive access to shared resources. This chapter first shows that protocols developed using Comheolaíocht are scalable (Section 6.1) and reliable (Section 6.2). Next, the chapter demonstrates the use of Comheolaíocht in the development of protocols for entities coordination in the intersection collision avoidance scenario and shows how the intersection scenario can be composed into a complex scenario that involves two adjacent intersections (Section 6.3). This adjacent intersection scenario also demonstrates Comheolaíocht's capability to divide a complex problem with interleaving constraints into smaller problems for concurrent development and composing them back into a system. Finally, the scalability and reliability of these scenarios are demonstrated in a large-scale simulation experiment (Section 6.4). The simulation results confirms that coordination protocols designed by Comheolaíocht are scalable and reliable.

6.1 Scalability

One of the main challenges in an multi-entity coordination system is scalability due the disproportionate increase in required resources such as processing power, memory and bandwidth as the number of entities increases (Section 2.1.4.1). Resources usage can be minimized by limiting coordination to the relevant parties, for instance, coordination amongst physical entities may utilize the geographical location of the entities. However, in an environment where entities are mobile, such localized communication results in dynamic participation. This section shows that Comheolaíocht-developed protocols support system scalability to entity numbers, Comheolaíocht's support for dynamic participation is discussed in the next section where it is shown that the protocol is safe (Section 6.2.1).

The following sub-section shows that the CwoRIS pattern supports localized coordination and that upper bounds on the number of entities which need to participate in the coordination protocol can be derived. Given that an entity only needs to coordinate with at most a fixed number of entities, it can be shown that requirements on bandwidth (Section 6.1.2), memory (Section 6.1.3) and computation power (Section 6.1.4) can be bounded.

6.1.1 Local coordination

As presented in Section 5.3.2, the CwoRIS pattern supports local coordination by defining some policies and the entities' *Sending area* and *Lurking time*. The *Sending area* ensures that messages are only sent to relevant parties and the *Lurking time* ensures that entities only need to listen for a certain period. The following policies are defined in the CwoRIS pattern:

1. Scheduling version: the period for which a resource can be reserved is bounded by t_{maxHold}.
2. Mutual exclusion version: the period within which an entity must start using the resource after its acquisition is bounded by t_{maxME}.

These two policies serve to bound the time for which information is relevant. In the scheduling case, information beyond (before/after) t_{maxHold} is not important by design. In the mutual exclusion case, information older than t_{maxME} can be verified using the resource sensors, allowing an entity to ignore any messages older than t_{maxME}. These two parameters provide an upper bound on the amount of information that need to be considered. This upper bound is translated to savings in entities' memory (for storing the relevant information) and computation (for performing local scheduling).

Since physical entities' speed is bounded (e.g., vehicles' maximum velocity, $v_{\max}$, in the intersection collision avoidance examples), both *Sending area* and *Lurking time* can be translated into upper bounds in physical area. Since physical entities have actual physical bodies, there are located in some physical space, it implies that an area can only have a certain number of entities. Together, these translate to an upperbound on the number of entities in the sending area.

The following sections present calculations of the resources usage (i.e., bandwidth, memory and computation power) bound based on the upper bound on the number of entities, n_{entity}, within the sending area.

6.1.2 Bandwidth requirements

An entity requiring access to the shared resources sends out a request to entities in its *sending area.* In each request, the maximum number of resources required by an entity can be calculated based on the flow-shop, flexible flow-shop and the job-shop constraints (See Section 3.4.3.2). Therefore, there is an upper bound, s_{request}, on the request size. On receiving a request, every entity within the sending area will reply with an acknowledgment, let the size for an acknowledgment message be s_{ack}.

By definition, the entity sends its request to everyone that might access the shared resources at the same time. Since the entity sends the request in order to access those resources, a rational entity would only request for resources that it may access. Therefore, the entity must be in its own sending area when its request is delivered. This implies that the number of entities sending a request at any one time is bounded by the number of entities in the sending area, n_{entity}.

For fairness, it is designed that an entity sent at most one request during the $MsgLatency$ period. As such, the total required bandwidth over the sending area is no greater than:

$$\text{Bandwidth upper bound}: \frac{n_{\text{entity}}(s_{\text{request}} + n_{\text{entity}} s_{\text{ack}})}{MsgLatency}$$

In the event that the bandwidth upper bound cannot be accommodated, some ways to lower this upper bound include:

- Lowering n_{entity}: By reducing the sending area, by coordinating only with the minimal set where entities must enter their LLFSM to coordinate (see Section 4.4.3.2), instead of the optimal set, where entities may decide before arriving at their decision point (see Section 5.3.1).
- Lowering s_{request}: By using a coarser resource model where the environment is represented by fewer resources.
- Increasing $MsgLatency$: Having entities send less messages over a period.

Overall, we showed that an upper bound on the required bandwidth over the sending area can be derived and this upper bound does not depend on the total number of entities in the system.

6.1.3 Memory requirements

An entity may need to remember some information about other entities in order to coordinate. In the CwoRIS pattern, an entity stores the received requests (in the situation picture data structure), and the identifiers of entities who have acknowledged its previous request (in the *prevMembers* data structure).

By the definition of sending area, a request is only relevant to entities in the sending area. That is, the calculation of sending area has already taken into account the duration for which requests are relevant (by the t_{maxhold} and t_{maxME} policies) and entities entering the scenario during this period (by the maximum speed of entities $v_{\max}$). Therefore, an entity is only required to remember requests only from entities in the sending area, which is bounded by n_{entity}. Since an entity only needs to remember the last received request from each entity, the total number of requests that an entity needs to remember is also bounded by the number of entities in the sending area, n_{entity}. As such, the upper bound on memory usage for the situation picture data structure is: $s_{\text{request}} * n_{\text{entity}}$.

The *prevMembers* data structure records the identifiers of entities who acknowledged the entity's previous message. The size of this data structure is bounded by: n_{entity} *size to store a vehicle's identifier.

Other information items stored in the developed protocol are of fixed size and therefore do not increase with the number of entities. We showed that an upper bound on the required memory (which does not depend on the total number of entities in the system) can be derived.

6.1.4 Computational requirement

There are two computationally-expensive methods in CwoRIS: the request/feedback protocol and the local scheduling/rescheduling methods.

As mentioned, an entity sends at most one request in *MsgLatency*. Since there are at most n_{entity} in the sending area at any one time, the number of requests received by an entity within a period of *MsgLatency* is bounded by n_{entity}.

Receive request Receiving a message involves calling the *receiveRequest* function in the request/feedback protocol (Figure 5.6). There are two potentially expensive operations in this function: checking for conflicts between two requests and maintenance of the situation picture.

In our implementation, the resources in a request are sorted by their identifier, therefore a comparison of two resources sets for conflict has the order of $O(2 * |R|) = O(|R|)$ operations (where $|R|$ denote the number of resources in the environment). Note that even if the resources are not sorted in advance, the comparison is $O(|R| log|R|)$ (i.e., using a sorting algorithm like merge sort, [Cormen et al., 2001]). Since $|R|$ is a constant defined at design time, conflict checking between two requests has constant computation time.

The situation picture in each entity can be implemented using a stack data structure [Cormen et al., 2001] where newly-received messages require a constant time to insert. Expired records are removed periodically (i.e., every $2 * MsgLatency$) using a cleanup method. Therefore, records could be kept in the stack for a period of $2 * MsgLatency$. As mentioned, there are at most n_{entity} requests received during a period of $MsgLatency$, therefore the situation picture stack holds at most $2 * n_{\text{entity}}$ records and the cleanup operation has the order or $O(n_{\text{entity}})$.

Since both checking for conflict between two requests and inserting an accepted request into the situation picture can be performed in constant time, the receive request function can be performed in constant time. An entity receives at most n_{entity} requests for the $MsgLatency$ period, therefore the receive request requires $O(n_{\text{entity}})$ operations during the $MsgLatency$ period. The situation picture maintenance has the order of $O(n_{\text{entity}})$ computations. Therefore, the computation required for the request/feedback protocol is bounded.

Local scheduling/rescheduling Two other expensive operations in CwoRIS are the local scheduling and rescheduling functions. The local scheduling function searches for available resources by checking the entity's required resources against the situation picture. Our implementation uses a brute-force method that compares every record in the situation picture to the entity's required resource, the method takes $O(n_{\text{entity}} * |R|^3)$ computation.

Rescheduling involves simulating every entities' local scheduling for live-lock prevention (Section 5.5.2), therefore rescheduling costs $O(n_{\text{entity}}^2 * |R|^3)$ computation.

Since $|R|$ is a constant and n_{entity} has an upper bound. The operation can be completed within a fixed amount of time.

6.1.5 Summary

We showed that the CwoRIS pattern (which Comheolaíocht uses) supports local coordination thereby providing an upper bound to the number of entities in the sending area. As a result, requirements on bandwidth, memory and computational power can be bounded and this upper bound does not depend on the total number of entities in the system (i.e., it only depends on the number of entities in the sending area). The system is therefore scalable with the number of entities.

6.2 Reliability

Protocols developed by Comheolaíocht are scalable because of the localized coordination, which may result in dynamic participation when the entities are mobile (Section 6.1). The dynamic participants property implies that entities may arrive at a scenario at the same time (the simultaneous arrival property) and may leave the scenario without telling other entities (the absent without notice property).

The system's safety constraints, goals and optimizations form the system's objectives (Section 2.1.2.4). This section shows that protocols developed by Comheolaíocht are reliable by proving their *safety* and *progress* property; optimizations are not within Comheolaíocht's scope. We show that protocols developed by Comheolaíocht support:

Safety: by proving that entities'/system's safety constraints are not violated despite unreliable sensors and actuators, entity failures, unreliable communication and simultaneous entity arrival (Section 6.2.1).

Progress: by showing that entities will not create a deadlock or a live-lock (Section 6.2.2). The sections also describe how Comheolaíocht can be used to overcome starvation.

6.2.1 Safety

The coordination strategy derived from Comheolaíocht's system analysis step ensures that entities can deterministically transition into a LLFSM before its transition into a non-fail-safe mode, therefore, entity failures and unreliable sensors and actuators are handled in the analysis step. In addition, the request/feedback protocol's support for setup time β-constraint can be use to provide allowance between resource reservations so that an entity may have enough time to detect and react to other entities' failure.

The CwoRIS's coordination scheme defined that an entity which does not have access to its required shared-resources must start to prepare its transition into its LLFSM at its decision point. By design, the coordination strategy from Comheolaíocht combined with the coordination scheme from CwoRIS is able to ensure that entities only access their shared-resources after they have acquired access to them. This section shows that when an entity has acquired access to the shared resources using CwoRIS's request/respond protocol, the entity is ensured exclusive access to those resources despite imperfect communications and simultaneous arrivals.

The proof begins by first enumerating all the factors that may affect an entity's internal representation (i.e., communication errors, prior requests and delivery order; See Section 6.2.1.1). Each of these factors are then mapped to the entity's internal representation in the request/feedback protocol (Section 6.2.1.2). Using the conditions in the request feedback protocol (Section 6.2.1.3), the proof shows that there cannot be a situation where two entities are granted access to the same resources (Section 6.2.1.4).

6.2.1.1 Enumeration of factors

Factors that may change the results of the request feedback protocol are:

- Message delivery & errors

  Communication errors may result in a lost request, or lost acknowledgment. There are three possible outcomes when entity x sends a request to an area that includes entity y:

  Success: A request sent by x is delivered to y and y received x's acknowledgment.

  LostAcknowledgement: A request sent by x is delivered to y but x did not receive y's acknowledgment.

  LostMessage: A request send by x is not delivered.

- Previous requests

  The number of prior requests that x has sent to y can be categorize as:

  None: x has not sent any request to y (within the period specified by the policies: t_{maxhold} or t_{maxME}).

  Once: x has sent a single request to y prior to the current request.

  Many: x has sent more than one requests to y prior to the current request.

- Request delivery order (race conditions)

  Due to the simultaneous arrival property, entities may enter the system and send their requests at the same time. CwoRIS assumes a lower-layer communication protocol that provides ordered delivery, therefore, the delivery order are the same at each entity. The cases for sending and delivery of a request between two entities entering the scenario at the same time are:

  SDX x sent a request that is delivered to y before y sends its request (Figure 6.1.a).

  SDY y sent a request that is delivered to x before x sends its request (Figure 6.1.b).

  SSX x and y both sent a request with x's request delivered to y after y send its request but before y's request is delivered (Figure 6.1.c).

  SSY x and y both sent a request with y's request delivered to x after x send its request but before x's request is delivered (Figure 6.1.d) .

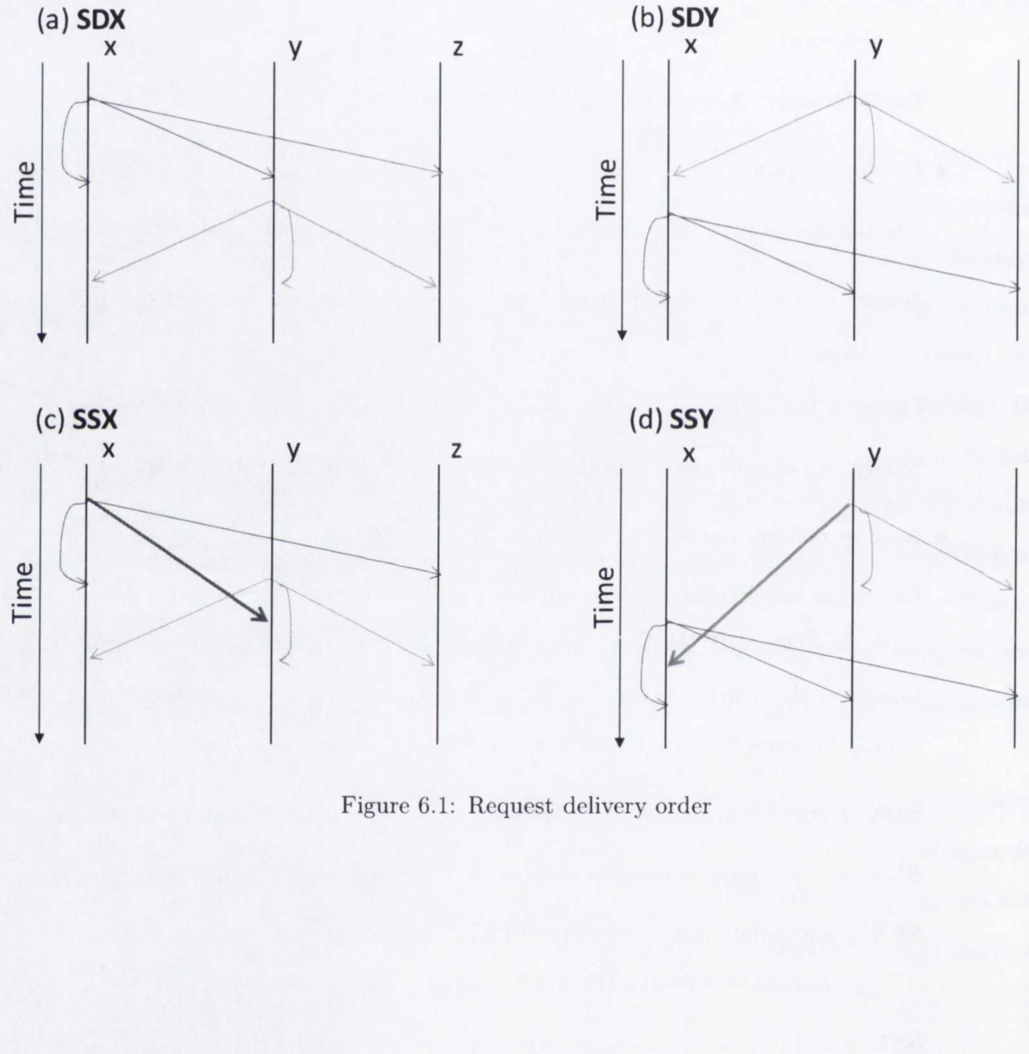


Figure 6.1: Request delivery order

6.2.1.2 Internal representation in the request/feedback protocol

This section shows how the values of the 'previous requests' and 'message delivery and errors' are represented in the request/feedback protocol's internal representation. Information is stored the protocol's two data structures (situation picture *sitPict* and *prevMembers*). The mappings are:

None x has not sent any request to y.

$y \notin x.prevMembers \wedge y.sitPict$ does not contain x's intention

Once x has sent a single request to y, and the result was:

> **Success:** the request was successfully delivered and acknowledged.
>
> $y \in x.prevMembers \wedge y.sitPict$ contains x's intention
>
> **LostAcknowledgement:** the request was delivered to y but x did not receive y's acknowledgment.
>
> $y \notin x.prevMembers \wedge y.sitPict$ contains x's intention
>
> **LostMessage:** the request was not delivered.
>
> $y \notin x.prevMembers \wedge y.sitPict$ does not contain x's intention

Many x has sent more than one request to y. The last request that x sent was:

> **Success:** successfully delivered to y.
>
> $y \in x.prevMembers \wedge y.sitPict$ contains x's intention
>
> **LostAcknowledgement:** delivered to y, but x did not receive y's acknowledgment.
>
> $y \notin x.prevMembers \wedge y.sitPict$ contains x's intention
>
> **LostMessage:** not delivered.
>
> $y \notin x.prevMembers \wedge (y.sitPict$ does not contain x's intention $\vee$ $y.sitPict$ contains x's obsolete intention)

Grouping situations with similar internal representations, we get a set of four possible representations:

- None & Once-LostMessage

  $y \notin x.prevMembers \wedge y.sitPict$ does not contain x's intention

- Once-Success & Many-Success

  $y \in x.prevMembers \wedge y.sitPict$ contains x's intention.

- Once-LostAcknowledgement & Many-LostAcknowledgement

  $y \notin x.prevMembers \wedge y.sitPict$ contains x's intention

- Many-LostMessage

  $y \notin x.prevMembers \wedge$ ($y.sitPict$ does not contain x's intention $\vee$ $y.sitPict$ contains x's obsolete intention)

6.2.1.3 Conditions in the request/feedback protocol

Analyzing the request/feedback protocol (Figure 5.6), the conditions required for an entity to access the shared resources are:

1. From $receiveFeedback()$: Win first-come/first-serve $\wedge$ request delivered to everyone (who may access any of the required resources at the same time)
2. From $receiveRequest()$: Win first-come/first-serve $\Leftrightarrow$
   (a) did not receive other request OR: $\sim Received(other's\,request) \vee$
   (b) other's request is non-conflicting OR: $q \cap own\,request = \emptyset \vee$
   (c) can ignore the other's request:
      i. sender know own's intention: $q.NodeID \in prevMembers \wedge$
      ii. $prevRequest \cap q \neq \emptyset \wedge$
      iii. $q's\,priority < own\,priority$

By design, a node therefore holds a valid request only when the conditions $(1 \wedge (2a \vee 2b \vee (2ci \wedge 2cii \wedge 2ciii))$ are satisfied.

6.2.1.4 Exclusive access proof

After enumerating all the cases, this section begins the proof:

> *Theorem (Exclusive Access)*: Using the CwoRIS protocol, there is no situation whereby two entities, x and y, use the same shared resource at the same time.

Assuming Theorem *Exclusive Access* is false, then there is a situation where two entities, x and y, access the same shared resource at the same time.

Since a shared resource is used by entity x and y at the same time, the set of resources required by x and y, R_x and R_y respectively, has the relationship, $R_x \cap R_y \neq \emptyset$; condition 2b is $false$, therefore, either condition 2a or 2c must be $true$.

Case: SDX Assuming the last requests sent by x and y before they are allowed to access the shared resources were delivered in the order of '**SDX**'; x has sent its last request (that allows x to access R_x) that is delivered before y sent its request (that allows y to access R_y). Since x's last request allows x to access the resources, by x's condition 1, x's last request must have been delivered to everyone that may be accessing the resources at the same time.

- Scheduling version

  In the scheduling version, according to the formulation of Lemma *Sending area*, x sent its request to $sendArea_x(t)$, an area which includes all entities that may use any of the resources at the same time. Entity y uses (some of) the same resources at the same time as entity x, therefore y is within $sendArea_x(t)$ and has received x's last request.

- Mutual exclusion version

  In the mutual exclusion version, according to the formulation of Lemma *Sending Area-Mutual exclusion*, x has sent its request to $SendArea_{\mathrm{ME}}$, an area big enough to include all entities that may reach their sent-request location ($MsgLatency$ before arriving at their decision point) before x starts using the shared resources.

  In the '**SDX**' situation, x's request was delivered to y before y sent its request. x sent the request to everyone within $SendArea_{\mathrm{ME}}$.

  - If y is within $SendArea_{\mathrm{ME}}$, it hears x's request.
  - If y is outside of $SendArea_{\mathrm{ME}}$, y requires longer than t_{maxME} to reach its sent-request location. Therefore, y only attempts to send its request t_{maxME} later. Entity x must start using the resources before t_{maxME}, therefore, y may detect that x is using the shared resources with the resource sensors and it will not send its request until after x has completed using the shared resources, a contradiction to the assumption that x and y use the shared resources at the same time.

Therefore, entity y must have heard x's last request (both scheduling and mutual exclusion versions). Entity y receiving x's request implies that y's condition 2a is false. As such, condition 2c must be true.

Case: SDY The argument for delivery order of '**SDY**' could be similarly shown by reversing the roles of x and y.

Case: SSX The '**SSX**' case describes a race condition: the last requests sent by x and y, prior to accessing the shared resources, arrive in an order such that x's request is delivered to y between y sending its request and y's request being delivered.

Similarly to the **SDX** case, x's condition 1 ensures that x's last request must be delivered to everyone that may be accessing the resources at the same time before x accesses the shared resources. The following proof shows that y is within x's sending area when x's request is delivered in both scheduling and mutual exclusion versions.

- Scheduling version

  The **SSX** scheduling scenario follows the same argument as the **SDX** scenario: according to the formulation of Lemma *Sending area*, x has send the transfer message to $sendArea_x(t)$, an area which includes all entities that may use the resources at the same time. Since entity y uses the resources at the same time as entity x, y is within $sendArea_x(t)$ and has received x's last request.

- Mutual exclusion version

  Under the '**SSX**' race condition, x's message is delivered before y's message is delivered. Therefore, y could not have started accessing the resources when x's message is delivered. Since the '**SSX**' condition specified that y sent its message before x's message delivery, therefore y must be located after its send-request location when x's message is delivered. According to the formulation of Lemma *Sending area-Mutual Exclusion*, x sent its transfer message to $SendArea_{\text{ME}}$, an area big enough to include all entities that may reach their send-request location before x starts using the shared resources. Since y is after its send-request location and has not started using the shared resources, y is within $SendArea_{\text{ME}}$ when x's last request is delivered.

In both versions, y is in x's sending area when x's last request is delivered, therefore y has received x's last request due to x's condition 1 (i.e., x's message is delivered to everyone within the area). Therefore, y's condition 2a is false in the '**SSX**' case.

Case: SSY The argument for delivery order of '**SSY**' could be similarly shown by reversing the roles of x and y.

So far, y's condition 2a and 2b has been shown to be false (i.e., in all **SDX**, **SDY**, **SSX** and **SSY** cases), therefore, in order for entity y to access the shared resources at the same time as entity x, entity y's condition 2c must be true; condition $2ci \wedge 2cii \wedge 2ciii$ must be satisfied.

Proving condition 2c In order for entity y to access the shared resource and violate mutual exclusion, it must ignore x's last message, in other words, entity y's conditions 2ci, 2cii and 2ciii must be true. Note: If entity y accepts (does not ignore) x's last message (i.e., a conflicting request) it must either reschedule a non-conflicting request or delay sending a new request until its required resources are available.

Considering all possible permutations of entity y's internal representation and its satisfaction of condition 2c:

- None & Once-LostMessage

  In this case, y has not sent x any prior request (None) or that it has made one request but the message has not been received by x (Once-LostMessage). Therefore, y has the internal representation:

  - $x \notin y.prevMembers \wedge x.sitPict$ does not contain y's intention; y's condition 2ci is false (contradiction)

  Therefore y cannot have this internal representation when it receives x's message.

- Once-LostAcknowledgement & Many-LostAcknowledgement

  In this case, y has sent one or many requests to x but y's prior request exhibited a lost acknowledgment error.

  - $x \notin y.prevMembers \wedge x.sitPict$ contains y's intention; y's condition 2ci is false (contradiction)

  Therefore y cannot have this internal representation when it receives x's message.

- Many-LostMessage

  In this case, y has sent several requests to x prior to the current request, and y's prior request was not received by x.

  - $x \notin y.prevMembers \wedge (x.sitPict$ does not contain y's intention $\vee$ $x.sitPict$ contains y's obsolete intention); y's condition 2ci is false (contradiction)

  Once more, y cannot have this internal representation when it receives x's message.

- Once-Success & Many-Success

  In this case, y has sent one or more previous requests to x, x has also acknowledged y's request

  - $x \in y.prevMembers$; y's condition 2ci is satisfied
  - In order for y to access the shared resources, let's assume that both condition 2cii and 2ciii are satisfied

* Condition 2cii: y's prevRequest $\cap$ x's request $(q) \neq \emptyset$
* Condition 2ciii: x's priority $<$ y's priority

Condition 2ci implies that x has received and acknowledged y's previous request.

Condition 2cii implies that y's previous request is in conflict with x's last crossing request.

Condition 2ciii implies that entity y has a higher priority. Entity y's previous request could be delivered either before or after x sent its last request:

After: y's conflicting previous message is delivered after x sent its last request. Therefore,

* x's condition 2a, $\sim Received(y's\,prevContract)$ is false
* x's condition 2b, y's prevRequest (q) $\cap$ x's request $= \emptyset$ is false
* In order for x to access the shared resources, x's condition 2c must be true; i.e., condition 2ci, 2cii and 2ciii must be true
* x's condition 2ciii: y's priority $<$ x's priority is false (contradiction).

Therefore, entity x could not have satisfied the conditions required for it to access the shared resources.

Before: y's conflicting previous message is delivered before x sent its last request

* x could either accept or ignore y's previous request.
* If x accepts y's previous request,
  · In the scheduling scenario, x would have rescheduled another time to access the shared resources, such that x's request $\cap$ y's request $= \emptyset$. The entities are not using the same set of shared resources at the same time (contradiction).
  · In the mutual exclusion scenario, x will delay sending its request, therefore x cannot have use the resources at the same time. (contradiction)
* In order for x to ignore y's previous request,
  · x must have sent y a previous request in order for x's condition 2ci, 2cii and 2ciii to be true
  · x's condition 2ciii, x's priority $>$ y's priority contradicts with y's condition 2ciii which we assumed to be x's priority $<$ y's priority. (contradiction)

All the cases for which entity x and entity y access the same resource at the same time lead to a contradiction. Therefore, the assumption that entity x and entity y can access the same resource at the same time is not true; Theorem *Exclusive Access* is proven.

6.2.2 Liveness

This section shows the liveness properties of the protocols developed using Comheolaíocht; these protocols prevent deadlocks and live-locks. Section 5.5 presented how to prevent live-locks using the CwoRIS pattern; therefore, this section does not addresses live-locks. The following section proves that the request feedback protocol will not get into a deadlock. Section 6.2.2.2 then argues that starvation have to be prevented in a case-by-case basis and presents how CwoRIS's request/feedback protocol can be used in some of these cases to prevent starvation.

6.2.2.1 No deadlocks

A deadlock situation can arise only when all of the following conditions exists in a system [Coffman et al., 1971]:

1. Mutual exclusion: At least two resources are non-shareable.
2. Hold and wait: A process is currently holding at least one resource and requesting additional resources which are being held by other processes.
3. No Preemptive: Resources already allocated to a process cannot be preempted.
4. Circular Wait: A process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource.

It can be observed that the first three conditions may arise in CwoRIS. Comheolaíocht allows the definition of multiple resources that are non-shareable; condition mutual exclusion may arise. Entities who recieved an entity's request believes that the requesting entity is holding on to the resources, and the same requesting entity may sent another request for resources, thus allowing the hold-and-wait condition to arise. CwoRIS's preemption β-constraint (Section 5.4.3.3) is an optional inclusion.

> *Theorem (No deadlock)*: There are no deadlocks in the request/feedback protocol after two consecutive rounds of perfect communication.

Therefore, in this non-deadlock proof, we shall show that the circular wait condition cannot arise. The following proof assumes that when an entity retransmits a request, it only requests the same set or a subset of the resources from the previous request. The subset relation between two resource baskets R_1 and R_2 is defined as:

$$R_1 \subseteq R_2 := \forall \langle r_i, t_{s1}, t_{e1} \rangle \in R_1 \bullet \langle r_i, t_{s2}, t_{e2} \rangle \in R_2 \wedge (t_{s2} \geq t_{s1}) \wedge (t_{e2} \leq t_{e1})$$

This assumption is valid in CwoRIS's mutual exclusion version where resource requests are for the period of $[t_0, \infty)$, where t_0 refers to the current time. Since t_0 only increases, subsequent requests are naturally a subset of an entity's previous request. In scenarios where entities may change their plans (e.g., in the scheduling version), subsequent requests may not be a subset of their previous requests, in such cases, the nodes may get into a live-lock. Section 5.5 presented CwoRIS live-locks prevention.

In order to prove that the request/feedback protocol prevents deadlocks, let us define a minimal circular wait:

> *Definition (Minimal circular wait)*: A system has a *minimal circular wait* when all the nodes requiring access to the shared resources form a set S, such that every node in S waits for another node in S.

The proof starts by showing that CwoRIS's request/feedback protocol may resolve a minimal circular wait.

> *Lemma (Resolve minimal circular wait)*: If entities only request for a subset of resources in their subsequent requests and there are no new entities arriving, the request/feedback protocol can resolves a minimal circular wait situation after two consecutive rounds of successful communication.

Intuitively, the protocol uses a first-come/first-served policy and ordered delivery to assign which entities get access to the shared resources. The system resolves a minimal circular wait by creating a common ordering and uses this ordering to break out of the circular wait. In the proof, we shall first show that after a round of successful communication, the highest priority will not accept a conflicting message. The proof then shows that the highest priority node confirms that it has the right of way and can break out of the circular wait in the second round.

The lemma below describes that after a round of successful communication the highest priority will not accept a conflicting message:

> *Lemma (Highest priority ignores all)*: After a node with the highest priority, x, in S receives feedback from the set of nodes, $prevMembers_x$, it can ignore any conflicting requests from $prevMembers_x$ until the request expiry or after it has another message m'_x delivered.

Requests in the mutual exclusion version expire after t_{maxME} and requests in the scheduling version expire after their resource usage time has passed which lasts most t_{maxHold}. By design, an entity using CwoRIS's request/feedback protocol ignores a request if all three conditions are satisfied:

1. The sender has heard the entity's previous request.

2. The entity's previous request conflicts with the received message.

3. The entity has a higher priority.

Since x has highest priority amongst the nodes in S, condition 3 is satisfied. By design, $prevMembers_x$ contains identifiers of all the entities who heard x's previous request. If x receives a conflicting request, R_y, from an entity, y, in $prevMembers_x$, then y has heard x's previous request (condition 1 is satisfied). Since R_y is conflicting with x's current request and by assumption, x's current request is a subset of its previous request, therefore, R_y is conflicting with x's previous request (condition 2 is satisfied). As such, x can ignore any received conflicting requests from S; Lemma *Highest priority ignores all* is shown.

> *Lemma (Highest priority wins)*: Assuming that there are no newly arrived entities between two consecutive successful request deliveries, an entity with the highest priority, x, in S can acquire the resources it requires after its second message is delivered.

Given that there are no new arrivals, by Lemma *Highest priority ignores all*, entity x with its highest priority can ignore all conflicting requests after it has successfully delivered its first request, m_x, to everyone in S. Entity x can ignore any conflicting its requests while R_x has not expire. By design, entity x resends a request marked for delivery before the previous request expires. Therefore, entity x can ignore all messages that conflict with m_x before its second message, m'_x, is delivered. After m'_x is delivered, x can acquire the resources that it requires (thus proving lemma *Highest priority wins*).

Lemma *Resolve minimal circular wait* is a direct result from Lemma *Highest priority wins*; that is, the entity with the highest priority in the minimal circular wait would be able to gain access to its required shared resources after two rounds. Since a circular wait condition may not persists in CwoRIS request/feedback protocol, Theorem *No deadlock* is shown.

6.2.2.2 Starvation & design considerations

Starvation happens when an entity is perpetually denied the resources it requires. Without those resources, the entity can never reach its goal. Using two scenarios, this section shows that while Comheolaíocht does not have a generic solution that prevents all starvation, the CwoRIS pattern does provide some tools for starvation prevention.

Hidden dependency constraint The scenario where vehicles traveling on a road require access to the shared-resources in an intersection collision scenario was presented in Section 5.4.3.6 as an example of preemption β-constraint. This section first describes starvation in the example scenario, then shows that the problem arises because of a missing dependency constraint and demonstrates how the preemption β-constraint can be used in this situation.

In Dresner's (2009) and Bouroche's (2007) intersection designs (the former uses a centralized intersection manager and the latter uses a distributed contract protocol to obtain exclusive access to the junction), if a vehicle in front sends a request that is not delivered to all required entities, it cannot access the required resources for crossing. Subsequently, other vehicles behind the front vehicle may send a request for crossing the junction. Without proper handling, requests sent by the vehicles behind may be accepted; the result is that neither vehicles can cross the junction and the reserved resources are never used. In the worst case, many vehicles come after the first vehicle and their requests may prevent the first vehicle's request from being delivered, thereby causing starvation.

In this example scenario, vehicle entities exiting the road have a dependency constraint that is often overlooked - that a vehicle at the back cannot exit the road (to cross the junction) if there is another vehicle in front.

A simpler alternative to handle such dependency constraints is to use sensors; in this case, we can enforce that only the front vehicle may send a request (the vehicle uses sensors to deduce that it is the front vehicle). On one hand, this sensor-based method results in the advantage of reducing the total number of sent requests. On the other hand, it means that vehicles behind may only send their requests after the front vehicle has left the road (started crossing), therefore vehicles behind may not be able to send their requests and decide before arriving at their decision points; they must start breaking at their decision points.

The proof that both preemption and first-vehicle-sent solutions prevent the starvation described in this scenario is trivial. In the first-vehicle-sent solution, a vehicle behind will not send a request, therefore the first vehicle's request can be delivered and it may exit the road. Using the preemption β-constraint, using their sensors (e.g., cruise control) vehicles behind the front vehicle have to slow down and stop if the vehicle in front stops. Therefore, the front vehicle knows that vehicles behind it have not crossed their decision point and committed to crossing the junction. The front vehicle can then safely preempt the resources promised to any vehicles behind it.

Priority tweaking The request/feedback protocol uses entities' priorities to decide the winner in the event of races. This priority can also be used for preventing starvation, for instance an entity who has waited for longer can be assigned a higher priority compared to an entity who has just arrived. In addition, priorities may also be used in special situations whereby some entities require differential treatment (e.g.,

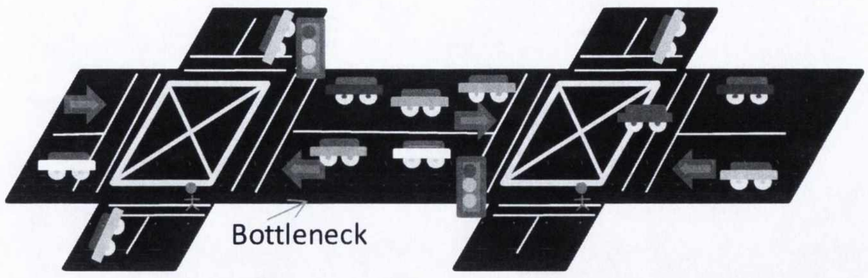


Figure 6.2: Bottleneck between two intersection

emergency vehicle). A direction for future work may involve deriving a generic method for using the priority for various purposes like non-starvation and providing special treatment like emergency vehicle. This section however, presents a scenario with a hand-crafted priority system for starvation prevention.

This second example describes a scenario (Figure 6.2) with two junctions connected by a short road in-between. The diagram shows the short connecting road with available room for only three vehicles. In this scenario, any vehicle that stays on the connecting road takes up one of these available spaces, effectively blocking any east-west traffic. The next section presents the modeling and analysis of this scenario using Comheolaíocht, and Section 6.4 presents our simulation results for this scenario. In our implementation, entities on the connecting road are provided with a higher-than-normal priority so as to let them win the first-come/first-serve race. This slight change in priority allows vehicles in the connecting junction to be cleared faster thereby allowing the east-west traffic to flow more smoothly.

Summary It can be seen from these two examples that starvation may be closely related to the scenario; a fair-solution providing every entity with an equal chance in accessing the shared resources may result in entities starvation when applied to scenarios with dependency or bottlenecks. While Comheolaíocht may not produce protocols that are fair or prevents starvation, it provides some tools which a developer may use to prevent starvation.

6.2.3 Summary

In this section, we showed that Comheolaíocht develops protocols that are safe and have no deadlocks. An entity will never use the shared-resources without first obtaining access to the shared-resources, and that no two entities may obtain access to the same set of resources; this ensured that a different time event ordering safety constraint will not be violated. The section also shows that entities will not be involved in a deadlock. In addition, we had presented a method for live-lock prevention in Section 5.5, and also shown that the Comheolaíocht provided some tools for starvation prevention.

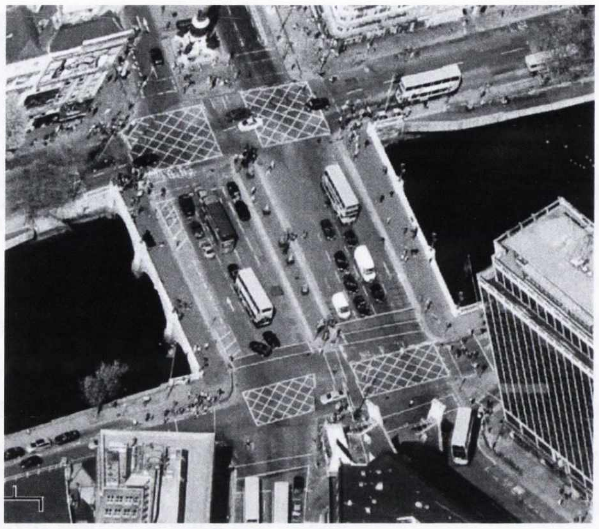


Figure 6.3: Intersection along Liffey River - Dublin (Picture from Google map)

6.3 Methodology steps

This section demonstrates the use of Comheolaíocht in the development of protocols for entities coordination in the intersection collision avoidance scenario and shows how the intersection scenario can be composed into a complex scenario that involves two adjacent intersections. This adjacent intersection scenario also demonstrates Comheolaíocht's capability to divide a complex problem with interleaving constraints into smaller problems for concurrent development and composing them back into a system.

The following sub-sections present the application of Comheolaíocht's three steps (system modeling, system analysis and protocol derivation) to design coordination protocols for the Liffey scenario (named after the road and intersection over the Liffey river in Dublin, Figure 6.3). Note: the section only presents the challenging steps in Comheolaíocht. In these scenarios, we assume that vehicles are driving on the left.

6.3.1 System modeling & specification

This section presents the modeling of the Liffey scenario using Comheolaíocht. For brevity, only the final model is presented (the intermediate steps are not shown). The scenario only models the vehicle-entity (Note: pedestrians are not supported in this evaluation). Its behaviors are:

- Driving: vehicle's velocity is greater than zero
- Stopped: vehicle's velocity is equal to zero
- Crossing junction: vehicle's velocity is greater than zero and is crossing the junction

- Change lanes: effect is that vehicle changed to the another lane
- Overtaking: vehicle x senses another vehicle, y, in front, the final observable effect is that vehicle y is behind x
- Breakdown: an uncontrollable behavior, vehicle's velocity falls to zero
- Accelerate: vehicle's velocity is greater than previous velocity.
- Decelerate: vehicle's velocity is smaller than previous velocity.

The Liffey scenario (Figure 6.3) is partitioned into two scenario-types: road, and junction. The junction scenario models the physical partition of the yellow-box of the intersection (Figure 6.3), there are four intersections in the diagram. The road scenario models the multiple lanes roads in the Liffey scenario. Note: the road scenario can be further decomposed into multiple lane scenarios, in this evaluation step we choose not to further decompose the road scenario for brevity.

The following sub-section presents the junction scenario's specification, the next section presents the road scenario's specification.

6.3.1.1 Junction scenario

This section presents the specification of the junction scenario. In particular, it presents a shared resources model and its constraint specifications for the junction scenario using the syntex provided in Comheolaíocht.

Scenario abstraction; different time constraints The constraint that two vehicles may not cross the junction at the same time is modeled as shared-resources with a grid-representation of the junction. The following specification models the junction scenario's shared resources. The specification represents the junction as multiple near-identical copies (i.e., $n{*}m$ copies) of resources differentiated by their location: (i, j).

$$\text{Junction Resource} := \text{junctR}^{n*m(i,j)}$$

The specification below defines the mapping of the resource representation to its physical location, it is assumed that every entity knows this resource mapping.

$$\begin{aligned}\text{junctR}^{n*m(i,j)} := \; & \text{let } x_{\text{step}} = \frac{(\text{junction}.x_{\text{top}} - \text{junction}.x_{\text{bottom}})}{n},\; y_{\text{step}} = \frac{(\text{junction}.y_{\text{right}} - \text{junction}.y_{\text{left}})}{m} \\ & 0 < i \leq n,\; 0 < j \leq m \,\text{in} \\ & \text{rectangle}(\; \text{position}(\text{junction}.x_{\text{bottom}} + i * x_{\text{step}}, \text{junction}.y_{\text{left}} + j * y_{\text{step}}), \\ & \qquad \text{position}(\text{junction}.x_{\text{bottom}} + (i+1) * x_{\text{step}}, \text{junction}.y_{\text{left}} + (j+1) * y_{\text{step}}))\end{aligned}$$

Scenario setting; sequence constraints This step defines the resource usage order; $\alpha-$ and $\beta-$constraint for the junction scenario. The $\alpha-$constraint is modeled as a job-shop:

$$\text{junctR}^{n*m(i,j)} \equiv J^{n*m(i,j)}$$

The following specification lists the job-classes. For instance, the record $J_{\text{southLeftTurn}}(\text{junctR}_{(1,1)})$ describes that all vehicle entities traveling from the south-direction and making a left-turn requires the resource $\text{junctR}_{(1,1)}$. The second record, $J_{\text{southStraight\_Lane1}}(\text{junctR}_{(1,1)}, \text{junctR}_{(1,2)}, ..., \text{junctR}_{(1,m)})$, describes that all vehicles traveling from the south-direction and driving straight on lane one requires the resources $\text{junctR}_{(1,1)} \blacktriangleright \text{junctR}_{(1,2)} \blacktriangleright ... \blacktriangleright \text{junctR}_{(1,m)}$ in sequence.

$$\begin{aligned} J = \; & \{J_{\text{southLeftTurn}}(\text{junctR}_{(1,1)}), \\ & J_{\text{southStraight\_Lane1}}(\text{junctR}_{(1,1)}, \text{junctR}_{(1,2)}, ..., \text{junctR}_{(1,m)}) \\ & J_{\text{southRightTurn}}(\text{junctR}_{(\frac{n}{2},1)}, \text{junctR}_{(\frac{n}{2},2)}, ..., \text{junctR}_{(n,\frac{m}{2})}) \\ & ,..., \\ & J_{\text{westLeftTurn}}(\text{junctR}_{(1,m)})\}\end{aligned}$$

The β-constraints of a junction are identified as: *nowait*, *brkdown*, and $job - family$ setup time. The *nowait* constraint states that vehicle entities may not stop in the center of the junction, the *brkdown* constraint specifies that some resources may become unavailable due to vehicle breakdowns and the $job - family$ setup time constraint is used to allocate enough time between entities usage of resource in the event of entity breakdown.

Pre-condition, post-condition and safety-constraints This step records the entrance- (pre-condition), goal- (post-condition) and safety-constraints of the junction scenario. The following specification captures the safety constraint for the junction scenario. The first line states that a vehicle whose mode is 'crossing' and whose location is at (x, y) in the junction must hold the resource $\text{junctR}_{(x,y)}$. The second line states that when a vehicle 'breakdown' and location is at (x, y) in the junction, then the resource $\text{junctR}_{(x,y)}$

has broken down. Note that this definition uses the short-hand defined in Section 3.4.4.2.

$$\begin{aligned}\text{safety constraint}_{\text{junction}} := \quad & \text{vehicle.crossing} \wedge \text{vehicle.location} \rightarrow (x,y) \implies \mathit{holds}(\text{junctR}_{(x,y)}) \vee \\ & \text{vehicle.breakdown} \wedge \text{vehicle.location} \rightarrow (x,y) \implies \mathit{brkdown}(\text{junctR}_{(x,y)})\end{aligned}$$

There is no entrance constraint for the junction scenario; the scenario's entrance condition is the same as its safety constraint. Note: the location where vehicles enter the junction is already captured in the job-shop constraints, therefore there is no need to specify it again in the entrance constraint.

An entity's goal-constraint in the junction scenario is to arrive at its destination direction.

$$\text{goal constraint}_{\text{junction}} := \text{vehicle.location} = \text{vehicle.destination direction}$$

6.3.1.2 Road scenario

This section presents the modeling of the road scenario. For easier reading, this section presents the modeling of the sequence constraints before the modeling of the different time constraints.

Scenario abstraction; different time constraints Vehicle entities in the road scenario should not collide. The road scenario is modeled as shared resources with m lanes, each further divided into n segments:

$$\text{Road Resource} := \text{roadR}^{n*m(\text{segmentid},\text{laneid})}$$

For simplicity, the road scenario reuses the grid shared resource model from the junction scenario:

$$\begin{aligned}\text{roadR}^{n*m(\text{segmentid},\text{laneid})} := \quad & \text{let } x_{\text{step}} = \tfrac{(\text{road}.x_{\text{end}} - \text{road}.x_{\text{start}})}{n}, \\ & 0 < \text{segmentid} \leq n,\ 0 < \text{laneid} \leq m \text{ in} \\ \text{rectangle}(\quad & \text{position}(\text{road}.x_{\text{start}} + \text{segmentid} * x_{\text{step}}, \text{road.lane}_{\text{laneid}}.\text{start}), \\ & \text{position}(\text{road}.x_{\text{start}} + (\text{segmentid}+1) * x_{\text{step}}, \text{road.lane}_{\text{laneid}}.\text{end})\end{aligned}$$

Scenario setting; sequence constraints This step defines the α-constraint and β-constraints for the road scenario. The road scenario's α-constraint is modeled as a flexible flow shop where all entities travel in a single direction from segment 1 up to segment n:

$$\text{roadR}^{n*m(\text{segmentid},\text{laneid})} \equiv FF^n :: [\text{roadR}_1^m, \text{roadR}_2^m, ..., \text{roadR}_n^m]$$

Note: the scenario is modeled as a flexible flow-shop instead of flow-shop to signify that entities may change lanes.

The β-constraints of the road-scenario are defined as $block(\text{roadR})$ and $prmp(\{\text{roadR}_{(n,1)}, \text{roadR}_{(n,2)}, ..., \text{roadR}_{(n,m)}\})$. The first constraint $block(\text{roadR})$ defines that a vehicle entity may stop on the road thereby blocking other vehicles from using the road. The second preemption constraint makes every last segment of a lane a preemption resource; this is an implementation specific constraint to avoid entities starvation when they are leaving the road, the example is explained in Section 6.2.2.2.

Pre-condition, post-condition and safety-constraints This step defines the entrance, goal and safety constraints for the road scenario. The safety constraint of the road defines that in order for a vehicle to perform the change lane behavior, it must hold exclusive access to either of the adjacent lanes:

$$\begin{aligned} \text{safety constraint}_{\text{road}} := \; & \text{vehicle.changelane} \wedge \text{vehicle.location} \rightarrow (x, y) \\ & \implies ((holds(\text{roadR}_{(x,y+1)}) \vee holds(\text{roadR}_{(x,y-1)}) \wedge \quad y - 1 > 0 \wedge y + 1 \leq n) \end{aligned}$$

The safety constraint specification assumes that entities driving forward (without changing lanes) are capable of detecting the vehicle in front. Therefore, instead of requiring entities to reserve resources for traveling straight, entities are assumed to have reserved the resources in front of the vehicle (Section 6.3.3 shall define how far in front a vehicle is assumed to reserved resources).

The entrance constraint of the road specifies that any vehicle entering the road (with the driving behavior) must hold the first segment resource. This is to ensure that there is at most one vehicle entering the road at any one time. Note, once a vehicle is on the road (segment 2 and onwards), the non-collision constraint is handled by the flexible flow shop α-constraint.

$$\begin{aligned} \text{entrance constraint}_{\text{road}} := \; & \text{vehicle.drive} \wedge \text{vehicle.location} \rightarrow (x, y) \\ & \implies holds(\text{roadR}_{(x,y)}) \wedge x = 1 \end{aligned}$$

An entity's goal-constraint in the road scenario is to arrive at the end of the road, which is captured by the constraint vehicle.location $= \text{roadR}_{(n,y)}$ which binds the segment to n, the last segment on the road.

Modes	Velocity	Breakdown	Detected breakdown
Crossing	> 0	No	No
Breakdown	$= 0$	Yes	Any
Avoidance	$= 0$	No	Yes

Table 6.1: Vehicle entity's modes in junction scenario

$$\begin{aligned}\text{goal constraint}_{\text{road}} := \quad & \text{vehicle.location} = \text{roadR}_{(n,y)} \wedge \\ & \text{vehicle.destination.isLeftTurn} \implies y = 1 \wedge \\ & \text{vehicle.destination.isRightTurn} \implies y = n\end{aligned}$$

6.3.2 System analysis

Comheolaíocht's second step uses the algorithm presented in Chapter 4 to analyze the model specified in the first step to produce two results. Firstly, it determines whether Comheolaíocht can provide a reliable solution to the Liffey scenario; i.e., ensure that the vehicles do not collide. Secondly, if a reliable solution exists, this step outlines a coordination strategy that will be used to derive the coordination protocols in Comheolaíocht's third step.

6.3.2.1 Mode design

Junction scenario By applying the mode design step (see Section 4.2.5), the variable 'Breakdown' is identified as a participation variable because a vehicle that has broken down may exceed usage of its requested resources. The **crossing** and **breakdown** modes are partitioned using the 'Breakdown' participation variable. In addition, a broken vehicle has zero velocity, and a crossing vehicle has positive velocity, therefore, vehicle velocity is also a participation variable. In addition, an entity may react to another entity's breakdown after detecting the breakdown, therefore another participation varible 'Detect breakdown'. The **avoidance** mode is partitioned. The identified modes in the junction scenario are shown in Figure 6.1.

Following the steps to draw the mode transition diagram (see Section 4.2.6), the **crossing** mode is identified as the scenario's entrance mode. A **complete crossing** exit mode (i.e., end of the junction) is included to differentiate between the entrance and exit modes. Since a vehicle's velocity in the **crossing** mode is positive, it eventually reaches the **complete crossing** exit mode, therefore it is a timed-transition. This scenario assumes that a broken-down vehicle is eventually removed (e.g., by a recovery team, or the driver pushes the vehicle away from the junction), thereby making the **breakdown** mode another exit-mode. In addition, the breakdown transition has an uncontrollable cause. When a vehicle detects another vehicle's breakdown, it avoids colliding with the brokendown vehicle by transitioning to the **avoidance**

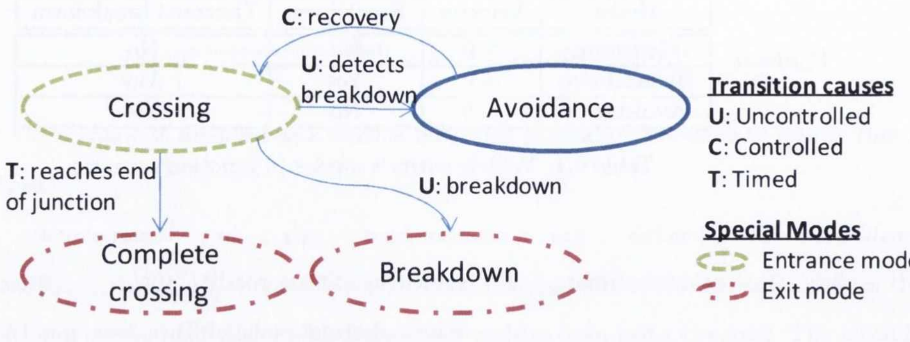


Figure 6.4: Junction scenario's mode transition diagram

Modes	Velocity	Heading
Driving	Any	Parallel to road
Change lane	> 0	Not parallel to road

Table 6.2: Vehicle entity's modes in road scenario

mode; thus this transition is uncontrollable. After the broken down vehicle is removed, vehicles in their **avoidance** mode may transition back to the **crossing** mode. The mode transition diagram for the junction scenario is shown in Figure 6.4.

Road scenario Using the mode design steps, the **change lane** mode is partitioned from the **driving** mode because of the road scenario's safety constraint, which describe a violation of the safety constraint when the vehicle perform the changes lane action. The identified modes in the road scenario are shown in Table 6.2.

In order to draw the mode transition diagram for the road scenario, an **enter-road** entrance mode and an **exit-road** exit mode are included. The **enter-road** mode is included because the road scenario has an entrance constraint that specifies that vehicle entering the junction must hold an entrance resource. The **exit-road** mode is included so as to show that a vehicle driving may exit the road scenario, it is assumed that vehicles do change lane and exit the road.

Vehicles who enter the road travel a certain distance to arrive in the **driving** mode and eventually reach the end of the road. Compared to the junction scenario, where vehicles reach the end of junction with a timed-transition, the transitions in the road scenario are controlled transitions; this is because vehicles are allowed to stop in the middle of a road. In addition, a vehicle may control when it wants to transition into the **change lane** mode. A vehicle changing lane has a positive speed, therefore, the transition to new lane is a timed transition. The mode transition diagram for the road scenario is shown in Figure 6.5.

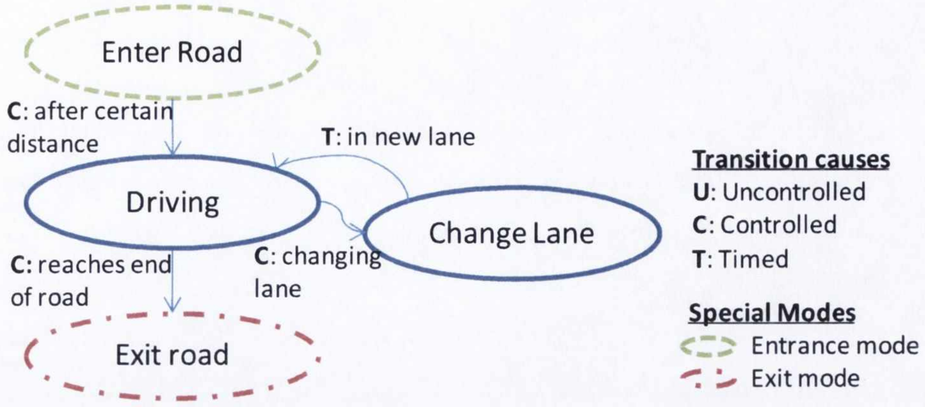


Figure 6.5: Road scenario's mode transition diagram

6.3.2.2 Solvability in Comheolaíocht

By following the steps in Section 4.3, a coordination strategy can be derived for each scenario. This section presents the coordination strategy for both scenarios.

Junction scenario

1. Identifying LLFSM

   Except for the **complete crossing** mode, all other modes in this scenario are non-fail-safe. The **crossing** mode is non-fail-safe because it is specified in the safety constraint. Both **avoidance** and **breakdown** modes are non-fail-safe because their usage of resources may exceed their reservations. **Complete crossing** mode is a fail-safe mode because when a vehicle has completed crossing it no longer requires the resources, however the mode is not a LLFSM because of the $no - wait$ β-constraint; the entity must move away.

2. Completing the coordination strategy table:

   By following the algorithm presented in Section 4.3.3, the coordination strategy can be derived:

Index	Mode	Result	Condition	Comments
1	Crossing	Unsafe	-	Entrance, non-fail-safe
2	Avoidance	Unsafe	-	non-fail-safe
3	Complete crossing	Safe	-	Exit, fail-safe-mode
4	Breakdown	Unsafe	-	Exit, non-fail-safe

   As seen from the coordination strategy above, the entrance mode is marked as *Unsafe*, therefore, vehicles do not have a stand-alone safe solution. The scenario's safety constraint is *exposed* to other scenarios; i.e., the constraints must be handled before entities enter the scenario.

Road scenario

1. Identifying LLFSM

   The non-fail-safe mode of the road scenario are the **enter road** mode and the **change lane** mode. The **change lane** mode has a safety-constraint that requires the vehicle to coordinate (holds some resource) and the **enter road** mode has a pre-condition that requires coordination; making both modes non-fail-safe.

   The **driving** mode is a LLFSM; all its out-edges are controlled transitions and it is a fail-safe mode. The **exit road** mode is a fail-safe mode.

2. Completing coordination strategy table:

   The **exit road** mode is marked as *Safe* because it is an exit mode and a fail-safe mode. Next the **change lane** mode is *Safe*, $(I : 2)$; an entity transitioning from the **driving** mode can ensure the mode safety. The **Driving** mode is marked *Safe* because it is a LLFSM and both its destination modes are marked *Safe*. The enter road scenario is marked as *Unsafe*; its pre-condition constraint is exposed. The coordination strategy table is:

Index	Mode	Result	Condition	Comments
1	Enter road	Unsafe		Entrance, non-fail-safe
2	Driving	Safe	-	LLFSM
3	Change Lane	Safe	I : 2	non-fail-safe
4	Exit road	Safe	-	Exit, fail-safe mode

6.3.2.3 Scenario composition

This section composes the road and the junction scenarios into the Liffey scenario (Figure 6.2 and Figure 6.3). The composition is presented in two steps: the first step composes the intersection scenario with one junction and eight roads and the second step composes the two intersection scenarios into the Liffey scenario.

Intersection scenario The intersection scenario is made up of eight road scenarios and a junction scenario; four roads leading into the junction and four roads out of the junction. Let's start by composing the four roads leading into the junction:

- Roads leading into a junction (Four roads merge into a junction)

  1. Check for physical incompatibilities. There are no physical incompatibilities, for brevity, the actual steps are not shown.

2. Redesigning & analyzing modes

   (a) Identify mode transitions between exit and entrance modes:

      i. There is only one exit mode in the road scenario: **exit road**.

      ii. There is only one entrance mode in the junction scenario: **crossing**.

      iii. A vehicle may transition from **exit road** to **crossing** only if velocity > 0, the transition has a timed transition cause.

   (b) Identify entrance modes with resource usage pre-conditions. The junction scenario's **crossing** mode has a safety condition that is exposed (a pre-condition).

   (c) Revisit mode design step for the composite scenario.

      The last deterministically-accessible LLFSM before arriving in the junction mode is the **driving** mode. This mode is partitioned so as to differentiate between modes that will eventually enter the **crossing** mode (i.e., **driving before junction**) and modes that will not enter the **crossing** mode (i.e., **stopped before junction**). The modes are:

Modes	Velocity	Heading
Driving before junction	> 0	Parallel to road
Stopped before junction	$= 0$	Parallel to road
Change lane	> 0	Not parallel to road

      Figure 6.6 shows the combined mode transition diagram for a vehicle entity moving from a road scenario into a junction scenario before arriving at its destination road scenario. In the figure, the road scenario is shown on top and the junction scenario is shown in the middle (let's ignore the destination road scenario at the bottom of the figure for this discussion).

   (d) Apply coordination strategy analysis

      The completed coordination strategy table is as shown in Table 6.3. For brevity, the table shows only two roads (the other two roads are the same).

      Note: Example on how to use this coordination strategy table follows shortly.

3. Information required by entities entering the junction scenario

   (a) When to start the coordination protocol: as shown in the coordination strategy table (Figure 6.3), a vehicle entering the junction scenario from road.1 scenario starts coordination at mode index number 2. The entity transitions into mode index number 3 if it has not gain exclusive access to the shared resource by its decision point.

Index	Mode	Result	Condition	Comments
1	Enter road.1	Unsafe	-	Entrance, non-fail-safe
2	Driving before junction.1	Safe	O:5|3	
3	Stopped before junction.1	Safe	-	LLFSM
4	Change Lane.1	Safe	O:2	non-fail-safe
5	Exit road.1	Safe	I:2|3, O:21|3	
6	Enter road.2	Unsafe	-	Entrance, non-fail-safe
7	Driving before junction.2	Safe	O:10|8	
8	Stopped before junction.2	Safe	-	LLFSM
9	Change Lane.2	Safe	I:7	non-fail-safe
10	Exit road.2	Safe	I:7|8, O:21|8	
11...20	Modes for the other two roads			
21	Crossing	Safe	I:5|3, I:10|8, I:15|13, I:20|18 O:22|3,8,13,18, O:24|3,8,13,18	non-fail-safe
22	Avoidance	Safe	I:21|3,8,13,18	non-fail-safe
23	Complete crossing	Safe	-	Exit, fail-safe mode
24	Breakdown	Safe	I:21|3,8,13,18	Exit, non-fail-safe

Table 6.3: Coordination strategy for four roads leading into a junction scenario

Note: Vehicle from road.2 starts coordination at mode index 6 and transitions to 7 at its decision point. Vehicles from the other two roads are similar.

(b) Who to coordinate with: in order to enter the junction scenario (mode index number 21), an entity needs to coordinate with all the entities (with the highest priorities or no dependencies) in their last deterministically-accessible LLFSM (i.e., modes with index number 3, 8, 13 and 18).

4. Check whether the composition has high requirements

   (a) The minimum distance required for coordination is the distance between the vehicles in modes with index number 3, 8, 13 and 18; in this case the front-most vehicle of each lane. The minimum distance required for coordination is thus the size of the junction + two vehicles' length.

   (b) The minimum set of entities that may participate in the coordination is thus the total number of lanes across all roads leading into the junction.

Let's assume that both numbers are within communication limit, therefore, there are no 'high-requirements' and the composition could be solved.

- Junction exiting into four roads (Junction merge with four roads)

Let's continue the second part of the intersection scenario by composing the junction with the four roads vehicle exits into.

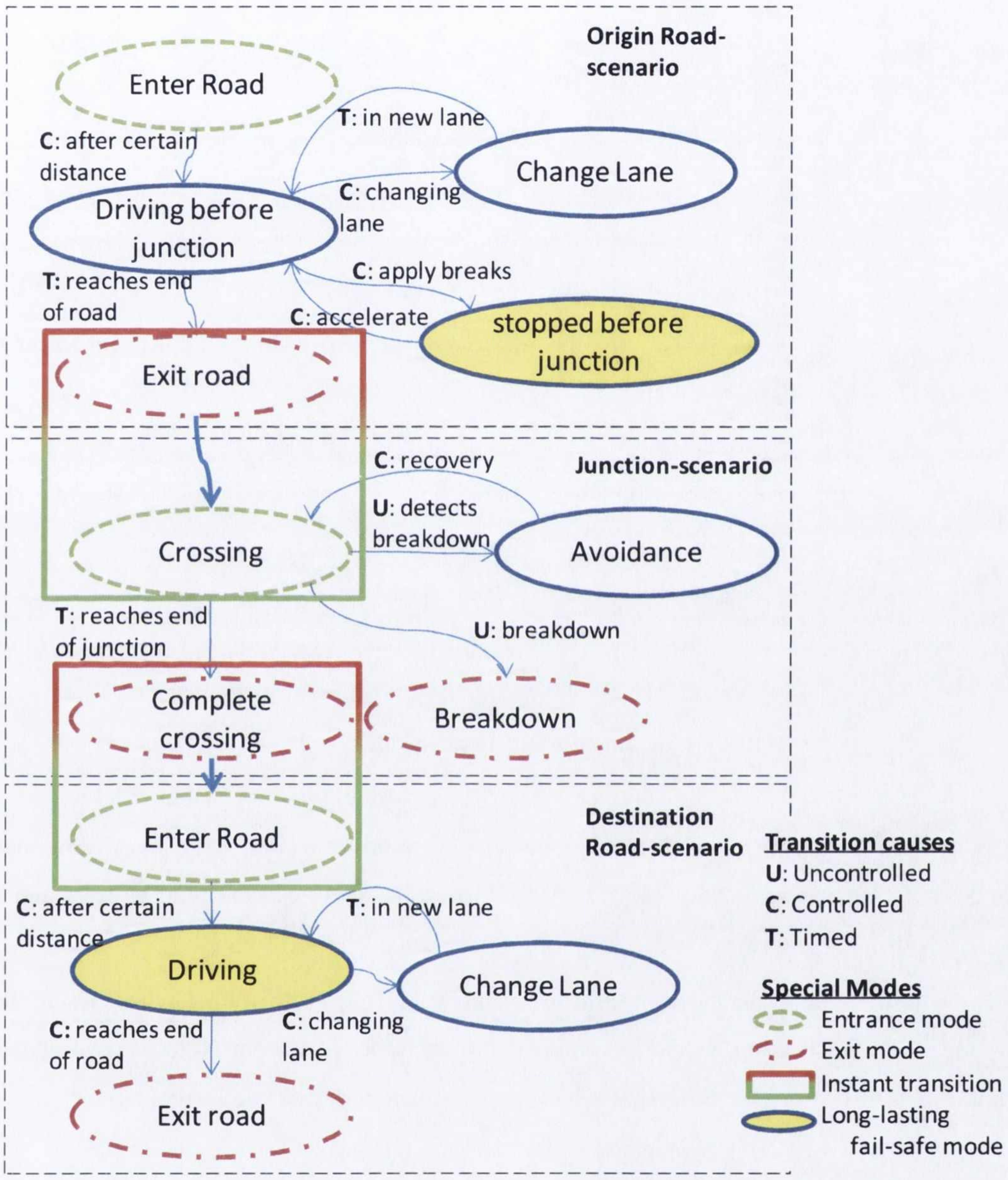


Figure 6.6: Mode transition diagram of road scenario leading into junction scenario

1. Checking for physical incompatibilities. There are no physical incompatibilities, for brevity, the actual steps are not shown.
2. Redesigning & analyzing modes
   (a) Identifying mode transitions between exit and entrance modes:
      i. There are only two exit modes in the junction scenario: **complete crossing** and **breakdown**.
      ii. There is only one entrance mode in the road scenario: **enter road**.
      iii. A vehicle may transition from **complete crossing** to **enter road** only if velocity $>$ 0, the transition has a timed transition cause. There are no transitions out of the **breakdown** mode.

   (b) Identifying entrance modes with resource usage pre-conditions. The **enter road** mode has a pre-condition.

   (c) Revisiting mode design step for the composite scenario: no new modes.
   The bottom of Figure 6.6 shows the combined mode transition diagram. Note: the figure only shows one exit road (instead of four).

   (d) Applying the coordination strategy analysis
   The completed coordination strategy table is as shown in Figure 6.4, for brevity, this table continues Table 6.3. Note: Table 6.3 is edited at mode index number:
      i. 21: the condition is edited to include $O : 23|3, 8, 13, 18$
      ii. 23: the condition is edited to $I : 21|3, 8, 13, 18$, $O : 25|3, 8, 13, 18$.

   The change is required because the entrance modes of the four exiting roads requires the coordination, and the last deterministically-accessible LLFSM are found at mode index number 3, 8 , 13 and 18.

In general, the analysis should compose the exit scenarios before composing the entrance scenarios, so that the coordination strategy table does not need to be recalculated. In this case, the author knows that the composition will not affect earlier results.

3. Information required by entity entering the junction scenario
   (a) When to start coordination: a vehicle exiting the junction scenario to enter a road scenario must start coordination at mode index number 2, 7, 12 or 17; i.e., at the same time as coordinating to enter the junction. The entity transitions into mode index number 3, 8, 13 and 18 respectively if it has not gained exclusive access to the shared resource by its decision point.

Index	Mode	Result	Condition	Comments
25	Enter road.5	Safe	I:23|3,8,13,18	non-fail-safe
26	Driving.5	Safe	-	LLFSM
27	Change Lane.5	Safe	O:26	non-fail-safe
28	Exit road.5	Safe	-	Exit, fail-safe mode
29-40	Mode for the other three roads (6,7,8)			

Table 6.4: Extended coordination strategy table for intersection scenario

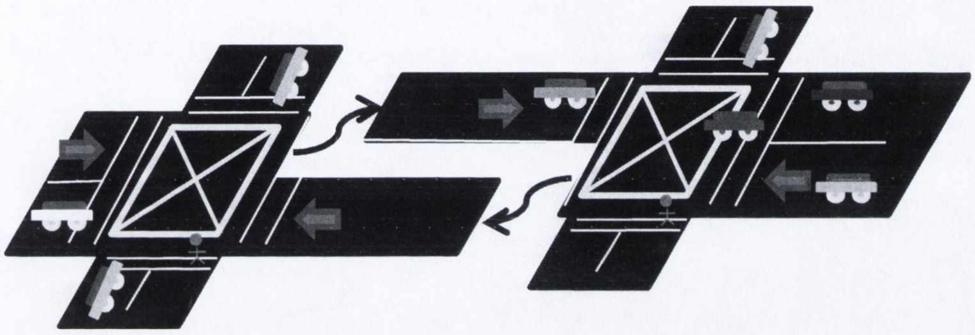


Figure 6.7: Liffey scenario composed by two intersection scenarios

(b) Who to coordinate with: all the entities (with the highest priorities or no dependencies) in the last deterministically-accessible LLFSM (i.e., modes with index number 3, 8, 12 and 17).

4. Check if composition has high requirements. Let's assume there are none (so that the development can continue; otherwise, the problem is not solvable.)

In summary, entities must coordinate in order to enter the junction and to enter the road after the junction, since the last LLFSM in both cases are only found in the origin-road scenario, the entities must start their coordination in that scenario.

Liffey scenario In order to differentiate between the two intersection scenarios, let's arbitrarily call one of them the first junction, and the other one the second junction. In addition, let's refer to the road in-between the two junctions as the middle-road. Instead of designing the Liffey scenario from scratch, this section composes the Liffey scenario using two intersection scenarios with an exiting road removed for each (Figure 6.7).

This composition is a straight (1:1) composition. When composing these two intersection scenarios into the Liffey scenario, the same four steps apply:

1. Check for physical incompatibilities. There is no physical incompatibilities.

2. Redesigning & analyzing modes

(a) Identifying mode transitions between exit and entrance modes:

i. The first junction's **complete crossing** mode transitions to the second intersection scenario's road scenario's **enter road** mode.

ii. Similarly, the second junction's **complete crossing** mode transitions to first intersection scenario's road scenario's **enter road** mode.

iii. Both transitions have a timed cause.

For brevity, lets skip the next two steps (identifying entrance modes with pre-condition and revisiting mode design)

(b) Applying coordination strategy

Following the steps for analysis of the coordination strategy, the coordination strategy table can be discovered. However, step 4 below shall question this coordination strategy.

3. Information required by entity before entering the junction scenario

The two questions, "when to start coordination" and "who to coordinate with", if answered without considering the length of the middle road, are exactly the same as the two intersection scenarios being handled separately.

Note: The next step considers the length of the road.

4. Check for composition with high requirements.

By examining the length of the middle road, some other questions appears,

(a) Can the entity transition into the middle road scenario's **stopped before junction** LLFSM that is required to coordinate an entity's entrance to the second junction?

(b) How many vehicles can stay within the middle road?

Depending on the length of the middle road, a vehicle may or may not be able to access the middle road scenario's **stopped before junction** LLFSM. Due to these considerations, the next section presents two different protocols for handling the Liffey scenario, termed the independent-Liffey and the combined-Liffey.

(a) Independent-Liffey: this scenario is composed as shown in the steps above, with the road scenarios between the junction scenarios and vehicles may access the stopped before junction LLFSM in the middle-road.

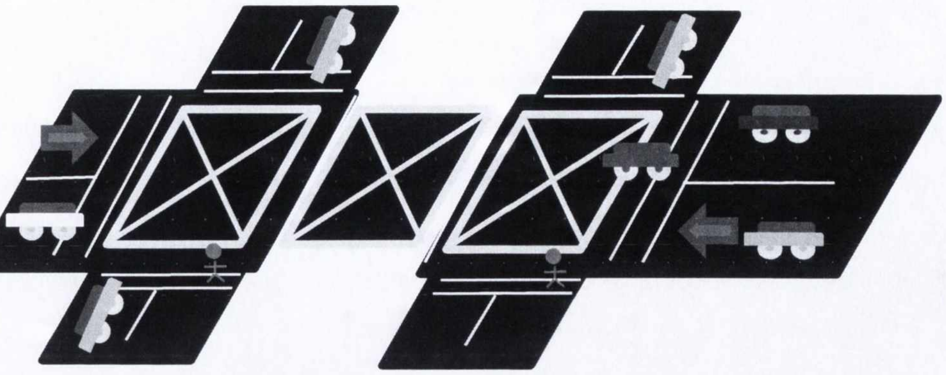


Figure 6.8: Combined-Liffey scenario

(b) Combined-Liffey: in this composite scenario, the middle road is represented as another junction scenario. Therefore the scenario is made up of three junction scenarios composed with 12 road scenarios (Figure 6.8).

Note: For brevity, the detailed steps for the composition of these scenarios are skipped, however, the next Section presents the coordination protocols for both Liffey scenarios and Section 6.4.4.2 presents the simulation results of both scenarios.

6.3.3 Coordination protocol with CwoRIS

This section applies the CwoRIS pattern to the scenarios for safe coordination. From the mode analysis steps, there are three situations which requires coordination:

1. Road scenario: vehicle executing a change-lane behavior
2. Vehicle leaving road scenario to enter junction scenario
3. Vehicle leaving junction scenario to enter road scenario

Since situation 2 and 3 are shown to have the same preparation location (in the **stopped before junction** mode) and require coordinating with the same set of entities (all roads leading into the junction(s)), these two situations are handled as one.

The first sub-section demonstrates the implementation of CwoRIS in the road scenario's vehicle change lane behavior. The intersection scenario was used as an example to explain the CwoRIS pattern (Section 5.3.4), and will not be revisited. Section 6.3.3.2 then demonstrates the CwoRIS pattern in the Liffey scenario.

6.3.3.1 Road scenario - change lane behavior

This section presents the application of the CwoRIS pattern in the road scenario to support vehicle's change lane behavior. The application of the CwoRIS pattern involves three steps: calculating the sending area, lurking time and deriving the vehicle's behavior.

In the road scenario, a vehicle entity usually travels straight without changing lanes. Therefore, instead of requiring entities to reserve resources for traveling straight, entities are assumed to have reserved the resources in front of the vehicle for a period of T_{break}.

$$T_{\text{break}} \geq \frac{v_{\max}}{d_{\min}}$$

In our implementation, the minimum distance for T_{break} must be greater than the amount of time required for a vehicle traveling at the maximum velocity ($v_{\max}$) to decelerate to a stop (let $d_{\min}$ be the minimum deceleration for deliberate breaking).

Concept Figure 6.9 shows the application of the CwoRIS pattern to the vehicle change lane behavior in the road scenario. In the figure, vehicle w (the white car) shows the entity considered in this calculation who wants to execute a change lane to the upper lane; w' shows the intended position of the vehicle after execution of the change lane behaviour.

In order to show the possibilities of the CwoRIS pattern, the concept for implementing the change lane behavior uses a combination of both scheduling and mutual exclusion version. Coordination with the destination lane (the lane with vehicle p, r and w') uses the mutual exclusion version for two reasons: firstly, vehicles may stop on a lane (specified by the *block*(roadR) constraint in Section 6.3). Secondly, a vehicle's cruise control can be use. Coordination with a competing lane (the lane with vehicle y) uses the scheduling version because vehicles in the competing lane cannot use the cruise control to sense the target area.

The assumption in this design is that the change-lane vehicle must be able to sense whether there are any vehicles at its destination (red color road in Figure 6.9) just before it executes the change lane behavior (Step 4 in Figure 6.9).

Vehicle's Behavior The steps required for vehicle w to perform the change-lane behavior are shown with numbers near the bottom of Figure 6.9. The steps are essentially the same as vehicles using the CwoRIS scheduling protocol with some changes to include CwoRIS mutual exclusion protocol:

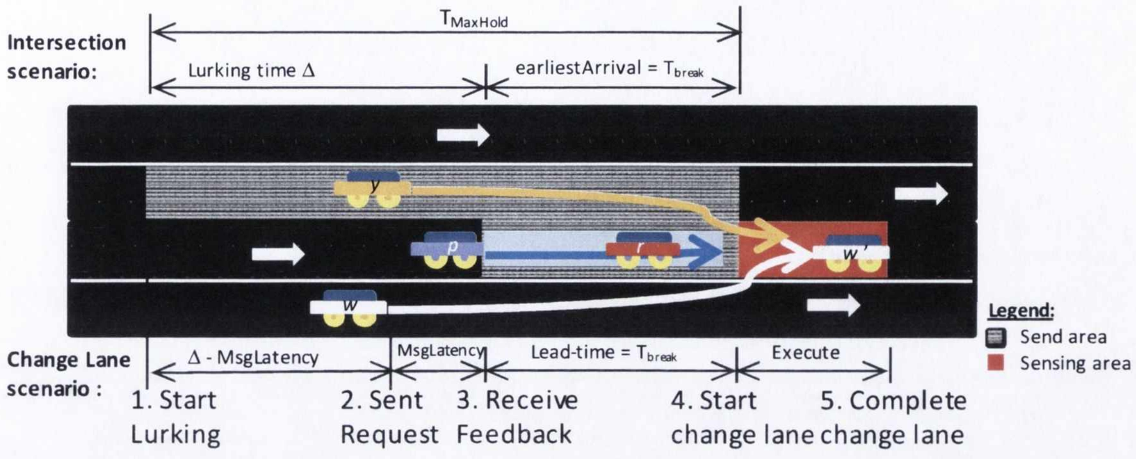


Figure 6.9: CwoRIS - Change lane in road scenario

1. A vehicle starts listening and building up a situation picture T_{maxHold} before the vehicle wants to start changing lane.

2. If the vehicle did not hear any conflicting request, it may send the request for changing lane. The request must be sent to the *sending area* described above and at least $T_{\text{maxHold}} - T_{\text{break}} - MsgLatency$ before the entity uses its resources.

3. The vehicle receives feedback using the request/feedback protocol, which determines whether the request is valid. If the request is valid, it may change lane in the specified time (which is at least T_{break} away).

4. When it is time for the vehicle to change lane, it may start changing lanes if it senses that there are no vehicles in the sensing zone (red color road in Figure 6.9).

5. The vehicle complete change lane behavior.

The three vehicles (p, r, y) in the figure show the different cases of a vehicle's behavior when it receives a change-lane request.

- Vehicle p (purple car): the vehicle is in vehicle w's destination lane and is greater than $v_{\max} * T_{\text{break}}$ away from the intented location where vehicle w wants to change lane to. Vehicle p is not within w's send area and will not receive w's request. When w changes lane, p would sense the vehicle and drives safely behind w using p's cruise control.

- Vehicle r (red car): the vehicle is in vehicle w's destination lane and is traveling near the intended location where vehicle w wants to change lane (within $v_{\max} * T_{\text{break}}$). By r's implicit reservation, vehicle r has rights to the resources that w requires. The behavior for vehicle r is the same as the mutual exclusion version of CwoRIS: r can choose to either:

  - Start using the resources (w's sensing zone, the red color road in Figure 6.9) before T_{break}. Vehicle w will sense that vehicle r is using the shared resources and cancel its change-lane.
  - Give way to vehicle w by not accessing the resources within the period specified in the request.

- Vehicle y (yellow car): vehicle y wants to change lane to the same destination lane as w's destination lane. Vehicle w and y are in a race condition which will be handled by the request/response protocol. Based on the protocol, the vehicle that has its request delivered first has the rights to use the resources.

Lurking time Based on the CwoRIS scheduling version, an entity can only hold some resource for at most a period of T_{maxHold}. Due to the implicit reservation by vehicles in the destination lane, a vehicle can only request a resource at least T_{break} in the future. When applied to Lemma *Lurking time* (repeated below for easy reference):

> *Lemma (Lurking Time)*: Given a policy that defines the longest period for which a resource can be reserved is T_{maxHold}. In order for an entity x to ascertain that no other entities is holding on to a resource γ after $T_{\text{maxhold}} - \delta$ from now, entity x must be able to access γ before $t_{\text{maxhold}} - \delta$ from now and has been lurking for δ without hearing a conflicting request.

Let *Execute* be the time required for a vehicle to perform the change-lane behavior. Therefore, vehicle x wants to change lane (access the resources) T_{break} from now, which is equivalent to $T_{\text{maxhold}} - \delta$ in the above lemma. In addition, the vehicle requires *Execute* time to change-lane, therefore the minimum value for T_{maxHold} is $\max(Execute) + T_{\text{break}}$.

Based on Lemma *Lurking Time*, in order for a vehicle, x, to ascertain that no other vehicles is holding on to the resources for changing lane into after T_{break} from now, entity x must be has been lurking for $\delta = T_{\text{maxHold}} - T_{\text{break}}$ without hearing a conflicting request.

Sending area Figure 6.9 shows the sending area for a vehicle attempting a change-lane behavior. The area shown is relative to the sending vehicle, w. Note that vehicles on the same origin lane are excluded of the send area because of the assumption on cruise control.

- Vehicles in the destination lane use the implicit reservation similar to CwoRIS's mutual exclusion. This means that those vehicles with implicit reservations (nearer than T_{break}) may use the resources at the same time. These entities must be informed of the entity's request.

- Vehicles in the competing lane may also request to change lane to the destination lane. These entities uses the CwoRIS scheduling protocol to request resources, therefore, the sending area must be $T_{\text{maxHold}} * v_{\max}$ away from where the vehicle first expects to use the resource.

In addition, vehicles must use sensors to check that the area is clear just before changing lane (Step 4 in Figure 6.9), this is required because vehicles in the destination lane could have stopped in the lane.

6.3.3.2 Liffey scenario

This section presents the application of the CwoRIS pattern to the Liffey scenario to support vehicles crossing the Liffey junction. In the Liffey scenario, the two junctions are connected by a middle-road. As mentioned there are two cases:

1. Combined-Liffey: The middle road is short, the middle junction is represented as another junction scenario.
2. Independent-Liffey: The middle road is long, the middle junction is represented as two road scenarios.

Note: while it is expected that there is a number which can differentiate long/short road and when is the combined/independent scenario better, determining this number is not within the scope of this thesis.

The application of the CwoRIS protocol involves three steps: finding the sending area, lurking time and the vehicle's behavior.

Combined Liffey In the combined scenario, the middle road is considered as part of the junction, therefore, vehicles cannot stop in the middle road. All three junctions are represented as resources and vehicles send requests to reserve the resources in order to access the combined junctions.

The vehicle's lurking time in the combined Liffey implementation is the same as the single intersection scenario. The calculation of the sending area is also the same. Vehicles must deliver their request to all vehicles within $sendArea_a(t) = (\max_{x \in R_a}(x.t_e) - t) * v_{\max}$ of the junction; the difference is that the combined Liffey scenario has six roads leading in to the junction instead of four, and that the size of the 'junction' in the combined Liffey scenario is larger. In addition, the vehicles' behavior remains the same in the combined Liffey scenario.

Independent Liffey The independent Liffey scenario treats both the two junction scenarios as seperate. However, compared to a scenario where the middle road is long and the two junction scenarios are indeed independent, there are two additional considerations for the Liffey scenario with short middle-road:

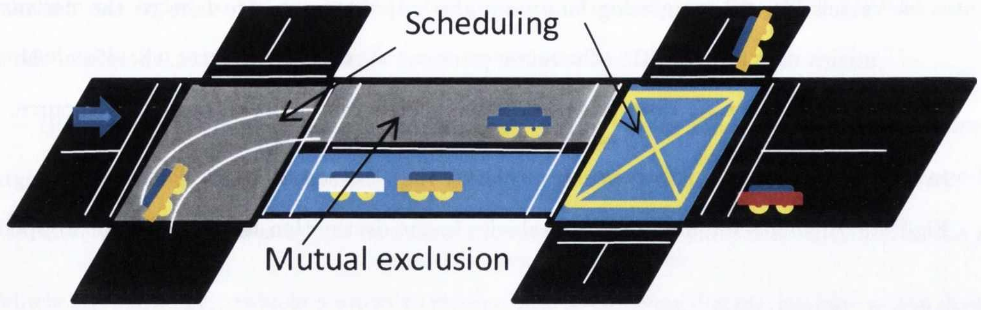


Figure 6.10: Implementation of independent Liffey scenario

1. Vehicles stopping in the middle road creates a bottle neck between the the two junctions. Section 6.2.2.2 describes that the situation may result in starvation for certain vehicles.

   As described in Section 6.2.2.2, starvation could be prevented by allocating vehicles in the middle road a higher priority to complete their crossing.

2. Limited driving space in the middle road: vehicles in the middle road take up space and vehicles may cross the junction then realize that there are no driving space for in the middle. While the normal exit-junction enter-road checks that the first few resources are not occupied before a vehicle attempts its crossing, the Liffey scenario is different because the exit-road (middle road) could be much shorter.

   Recall that a road has been modelled as $\mathrm{roadR}^{n*m(\mathrm{segmentid},\mathrm{laneid})}$. The n in the definition refers to the number of resources available; the CwoRIS pattern can be used to reserve the amount of resources a vehicle requires to occupy on the middle-road. Since a vehicle does not know how long it must stay in the middle-road, the mutual exclusion version is implemented. Therefore, it is assumed that the middle-road is equipped with resource sensors so that vehicles may access the sensors to check if there are any vehicles currently on the middle-road.

Figure 6.10 shows the implementation of the independent Liffey scenario. In the figure, a vehicle is shown crossing the west-side junction and is about to move into the middle-junction. In the figure, the middle junction is seperated into two colors, the northern side of the middle-road is grouped with the junction in the west-side and the southern side of the middle-road is grouped with the junction in the east-side; this is because a vehicle would perform the coordination to cross the west-side junction with the north-side middle road as a pair. Lets define the lurking time, sending area and behavior of the vehicles:

1. Lurking time: vehicles lurking time is the same as the a single junction. Vehicles in the middle road must lurk for the same time as those that just arrived in the Liffey junction.

2. Sending area: vehicles need to sent their requests for the junction and the middle-road to all four roads entering the junction; the middle-road and the three other roads incident to the junction. As the sending area encompasses the entire middle-road, there can be two possible implementations:
    (a) Send requests only to the middle-road: in this implementation, the vehicle lurking time only starts when it enters the middle-road.
    (b) Send requests to the middle-road and all roads leading into the junction: in this implementation, vehicles can 'start lurking' for the second junction even before it crosses the first junction.

   The implementation in the simulation uses the second implementation; the implementation is chosen because having vehicles lurking in the middle-road may contribute to the mentioned bottleneck in the middle-road.

3. Vehicle behavior: nearly the same as the intersection scenario except that vehicles must not exit the junction and enter the middle-road at high-speed if there are already vehicles in the middle-road (i.e., by checking the resource sensors in the middle-road).

6.3.4 Summary

This section demonstrated the use of Comheolaíocht in the development of protocols for the Liffey scenario. It is shown that Comheolaíocht can divides a complex problem into smaller problems which can be solved independently and be composed back to form the system.

6.4 Simulation results

This section demonstrates the scalability and reliability of protocols developed using Comheolaíocht in a large-scale simulation experiment.

The following sub-section presents the simulation configuration. Section 6.4.2 compares the simulation results of Comheolaíocht-developed protocols with other protocols designed specifically for intersection scenarios. The results show that protocols developed using the generic Comheolaíocht have similar performances to the specialized protocols. Section 6.4.3 presents simulation results focusing on the effects of communication and entity errors in Comheolaíocht-developed protocols. The results confirms that protocols developed using Comheolaíocht are reliable. Section 6.4.4 presents simulation results with varying entity numbers and shows that the developed protocol is scalable. The same section presents results for the Liffey scenarios and reinforces the argument that Comheolaíocht supports protocols design for complex scenarios.

6.4.1 Simulation configuration

There are a lot of robot development environments (RDE) for developing robotic applications. Kramer and Scheutz [2007] provide a good survey reviewing nine RDE. The simulation described in this Section is implemented using the Player/Stage RDE [Gerkey et al., 2003]. Player/Stage is used in the evaluation because of its support for multi-robots simulation and ease of usage. The Player/Stage is split into the player controller and the stage simulator, the player controller is the location where the vehicles' control and coordination code are implemented while the stage simulator provides the simulation environment.

This section presents the implementation of various components in the simulation, including the environment (Section 6.4.1.1), sensors (Section 6.4.1.2) and communication (Section 6.4.1.3).

6.4.1.1 Environment

Both the intersection and the Liffey scenarios are simulated on two-lanes roads; with two lanes going into and two lanes out of the junction. The maximum speed of the vehicles is set to 60km/h, and T_{maxHold} is set at 8 seconds. Each road is 280 meters long (i.e., Roundup(60km/h * T_{maxHold}sec * 2); the multiply by 2 is so that the scenario is big enough that geo-cast communication do not imply sending to every entity, and the rounding up is just for simpler calculations). Vehicles are randomly generated at a lane on a road leading into the junction and each vehicle entity has a random destination; vehicles on a left lane may only drive straight or turn left, similarly vehicles on the right lanes may only drive straight or turn right. In the event that a vehicle is generated on top of another vehicle, the vehicle is randomly regenerated on another lane.

In the Player controller, at each step, each vehicle entity makes its own local decisions on their steering and acceleration. Controls on the steering and acceleration are sent to the Stage simulator which simulates the vehicle's movement. In addition the simulator gathers the sensors' information which is returned to the Player controller.

The protocol for the independent Liffey scenario requires that the middle-road is equipped with sensors to detect entities on the road, the Stage environment simulates these sensors by looking up vehicles' actual location.

Any collisions between vehicles are recorded by Stage simulator using vehicles' actual location.

6.4.1.2 Entity's sensors

Each vehicle entity is equipped with a forward sensor that provides the vehicle with the distance to obstacles in front of it. The forward sensor's range is set to be two vehicles length greater than the vehicle's stopping distance. This sensor is used to implement reactive cruise control (applying breaks when there is a vehicle near) for the vehicle. Cruise control is only active on the road scenario and is deactivated when a vehicle is crossing a junction because vehicles are to follow their scheduled resource usage, the cruise control may unnecessary slow down a vehicle when it detects other vehicles crossing the junction.

In addition, each vehicle is equipped with GPS, speed and heading sensors. Stage simulates these sensors by reading the information of the actual representation.

6.4.1.3 Communication

Inter-entities communication is implemented using C++. Messages sent are collected at a centralized data structure and are delivered to the entities based on the specifications in Section 5.1 (i.e., geo-cast with ordered delivery, bounded message latency and real-time feedback.)

Imperfect communication is simulated by two parameters; the probability of a geo-cast failure, and the probability of each message failing given that the geocast has failed. This second parameter is used to simulate delivery failure of a message or an acknowledgment.

6.4.2 Implementation protocols comparision

This section compares the simulation results of Comheolaíocht-developed protocols with other protocols made specifically for the intersection scenarios. The following sub-section describes each of the protocols that is compared. The next sub-section describes the comparison matrix and show how Comheolaíocht-developed protocols compare with other protocols.

6.4.2.1 Intersection collision scenario protocols

The following describes the most significant existing protocols for the intersection collision avoidance scenario:

Traffic lights The traffic lights protocol models actual traffic lights in the intersection. The traffic light protocol is set to cycles of 20 seconds because it provides the best performances in the simulated scenario, and each cycle allows vehicles from a single direction to travel. In this evaluation, the traffic lights protocol is the only implementation that uses only sensors; it does not depend on communication and reservation of crossing space.

Centralized The centralized protocol implements vehicles' reservation for crossing the junction using a centralized server, which is modeled after the Autonomous Intersection Manager [Dresner, 2009]. A vehicle using this protocol sends a request to a centralized Intersection Manager which schedules the vehicle's crossing and replies to the vehicle.

The following describes Comheolaíocht developed protocols for the intersection collision avoidance scenario:

C No optimizations This protocol uses only local scheduling (without the live-lock prevention internal simulation method), it implements the request/feedback protocol with inference and priority presented in Section 5.4.2.

C Delay resend This version enhances the *no optimization* version by delaying an entity's resend request in order to prevent live-locks. Depending on the number of vehicles waiting to cross intersection and how long a vehicle has been waiting at the intersection, the vehicle decides the time it waits before resending its subsequent requests. That is, a vehicle that has been waiting for a long time, does not wait as long as a vehicle that has just arrived.

C Preemption This version implements both preemption β-constraint (Section 5.4.3.6) and the live-lock prevention internal simulation method (Section 5.5.2).

6.4.2.2 Comparison matrices

This section compares the simulation results of the different protocols described in the previous section for perfect communication. The protocols are compared with respect to vehicle density in the scenario; this parameter, vehicle density, records how many vehicles are in the scenario at the same time. In order to maintain the vehicle density, a vehicle is generated and placed on the road whenever another vehicle completes it crossing and leaves the scenario.

Graphs in this section compare different parameters (*y-axis*) (i.e., throughput, crossing time, number of vehicles crossing without stop) for different protocols (*lines*) over different vehicle densities on the road (*x-axis*). Each point on the graph is calculated by averaging over four sets of simulations where each set runs six hours of simulated time.

Throughput The first comparison parameter, throughput, records the number of vehicles granted access to cross the junction per hour. Figure 6.11 shows the graph comparing the throughput of the different protocols for different vehicle density.

The graph shows that each implemented protocol has a maximum throughput, such that a higher density of vehicles does not increase the throughput any further. The maximum throughput for simulations using:

- *Traffic lights* is reached when the vehicle density reaches 30.
- The other protocols is reached when the vehicle density is around 20.

It can be seen from the graph that the throughput of Comheolaíocht's protocol with *no optimization* decreases by around 50 vehicles after reaching its maximum at 20 vehicle density. The decrease in throughput is due to vehicles involved in live-locks when the density of vehicles increases.

The *Traffic lights* protocol has the highest maximum throughput at 1100 vehicles per hour. The *traffic lights* protocol gives the highest throughput because the implementation naturally provides a 'platooning' effect where vehicles form platoons at the traffic lights which moves off at the same time. In contrast, all other protocols are based on reservation where vehicles are allocated higher inter-vehicle distances for reaction time in the event of vehicle breakdowns. In contrast, *traffic lights* protocol provides the least throughput when vehicle density is low, this is because a vehicle has to stop at the junction even when there are no vehicles crossing.

Ranked second in maximum throughput is the Comheolaíocht's protocol with *preemption* at 900 vehicles per hour. The *preemption* version provides higher throughput compared to other version of Comheolaíocht's protocols (i.e., *delay send* and *no optimization*) because it has no live-locks. The *delay send* version is better than the *no optimization* version also because the *delay send* version uses time as a heuristic to break out of live-locks.

An unexpected result is that Comheolaíocht's protocol with *preemption* actually provides higher throughput than the *centralized* protocol. Further investigation reveals that two factors contribute to this result:

1. Comheolaíocht's protocols (including the *preemption* version) allow a slightly higher ratio of vehicles traveling straight and turning-left to cross the junction (33-35% each) when compared to vehicles turning right (31-33%) despite vehicles being generated with equal probability in each cases. This unbalanced ratio is attributed to two properties:

   (a) Right-turning vehicles require more resources and for a longer period of time.

   (b) Entities using Comheolaíocht's protocol coordinate in a distributed manner, they back-off when their required resources are not available and re-send their request after some time.

   These two properties mean that right-turning vehicles find their requests harder to be satisfied and take more time to cross. In contrast, vehicles driving straight require resources for a shorter period and vehicles turning left require less resources, these vehicles may therefore satisfy their allocations easier.

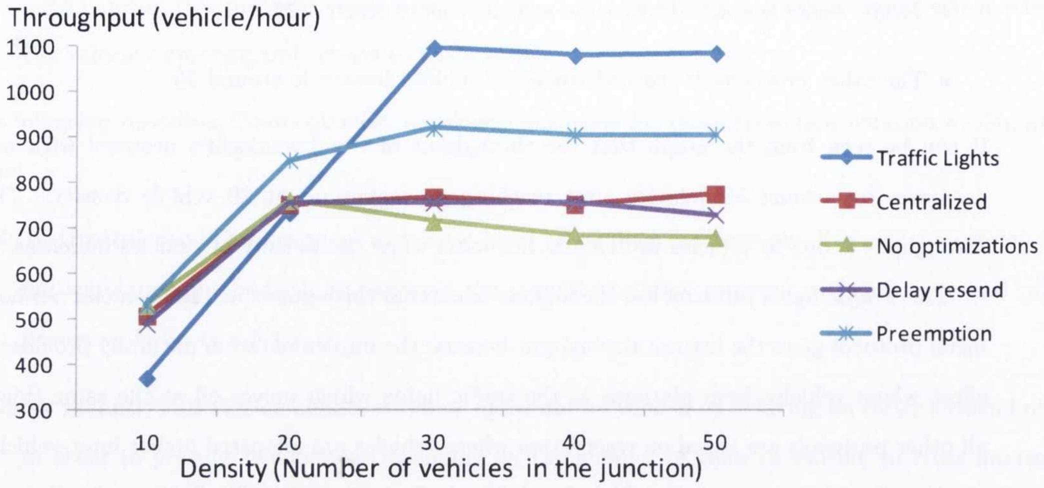


Figure 6.11: Junction throughput with different implementations

2. The *centralized* protocol grants vehicles' request based on a first-come first-serve allocation. Observation of the allocations revealed that resources usage in such allocations is fragmented; once a right-turning vehicle resources' are allocated, it is difficult for the centralized scheduler to fit other vehicles (e.g., driving straight or doing another right turn) in the schedule. The result is that other resources in the junction are not utilized.

Crossing time The second comparison parameter, crossing time, records the average (Figure 6.12(a)) and the maximum (Figure 6.12(b)) time recorded for vehicles crossing the junction. This crossing time is recorded from the period when the vehicle first arrives 250 meters from the junction untill the time the vehicle arrives at its destination road (just completed crossing the junction). Figure 6.12 shows the crossing time plot against the different vehicle density for comparison between different protocols.

The graph shows that the *traffic lights* protocol has a crossing time that is independent from vehicle density. This is because our simulation of 280 meters and 20 seconds is optimized to allow all vehicles queuing on a road to cross the junction in a single round. In all other protocols, the average and maximum crossing time increase with higher vehicle density. The crossing time for the other protocols however is smaller than the *traffic lights* version when the vehicle density is low.

Comheolaíocht's *no optimization* protocol has the highest crossing time because of live-locks; the duration for live-locks are reduced in both *delay send* and *preemption* version. The maximum crossing time is similar in the *centralized*, *delay send* and *preemption* protocols. While the maximum crossing time is high (near to twenty minutes) in instances where the vehicle density is high, it is observed (from the

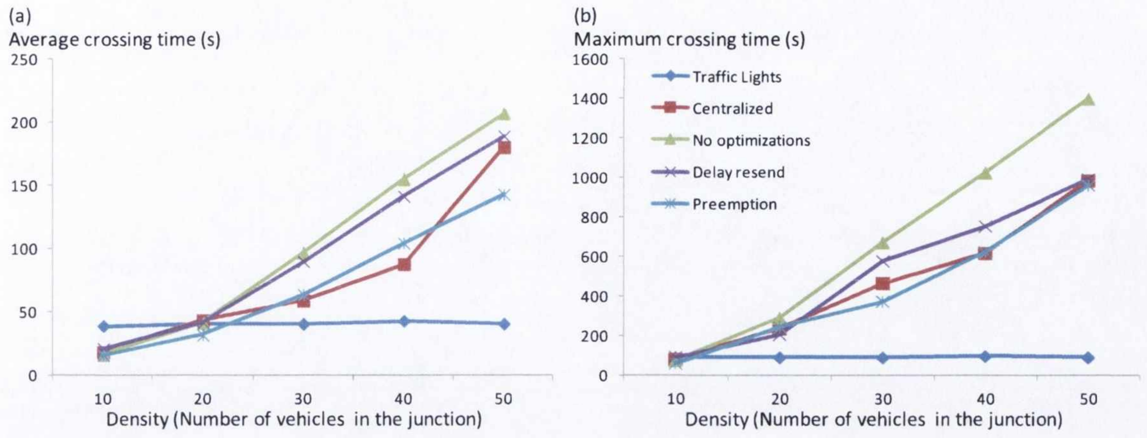


Figure 6.12: Crossing time in seconds. (a) Average, (b) Maximum

simulation output) that every vehicle eventually crosses the junction (i.e., there is no starvation). Note: by observing the last few minutes of each simulation, the author deduce that there are no vehicles that faces starvation - if there was any vehicle that faced starvation and cannot crosses the junction, then there would be a lane that has lots of vehicles queuing behind it. There are no simulation runs observed with such a long queue.

Comheolaíocht's *preemption* version provides average crossing times that are better than the *centralized* protocol in several cases, this is attributed to the protocol allowing more driving straight and turning left vehicles to cross, which brings down the average crossing time.

Crossing without stopping The third comparison parameter, crossing without stopping, records the number (Figure 6.13(a)) and the percentage (Figure 6.13(b)) of vehicles crossing the junction without stopping.

Comheolaíocht's *preemption* and *no optimization* protocols allow the most vehicles to cross the junction without stopping. The *preemption* version provides the best results because of the preemption β-constraint, a vehicle in-front may preempt another vehicle behind since the vehicle in-front knows that the vehicle behind could not have used the resources it had reserved. In contrast Comheolaíocht's *delay resend* version has a smaller number of vehicles crossing without stopping because they only wait for some time before resending a request, many vehicles arrive at their decision point during the wait and have to stop. Vehicles in the *centralized* version stop because there are no available resources, in many cases, an entity required resource has been allocated (i.e., to a right turning vehicle).

The *traffic lights* protocol provides a very high number of vehicles crossing without stopping when the vehicle density is at 30. However, on cross-examining the percentage of vehicles crossing without stopping,

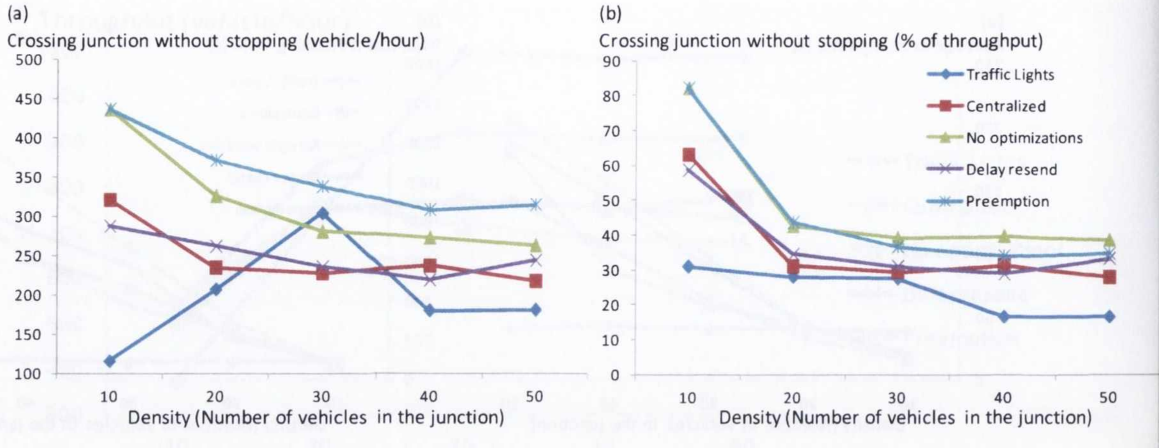


Figure 6.13: Vehicles crossing the junction without stop. (a) Number, (b) Percent of throughput

it shows that the percentage of vehicles crossing the junction without stop is constant at 30% for vehicle density below 30, and it drops to 20% when vehicle density is above 30.

In conclusion, Comheolaíocht's *preemption* provides better results than the *centralized* protocol because of two reasons: fragmented allocations in the *centralized* protocol, and Comheolaíocht's allowing more left-turn and straight crossings. Comheolaíocht's protocol allows the most vehicles to cross the junction without stopping because of the preemption β-constraint. However, the *traffic lights* implementation naturally provides vehicle platooning thereby giving the best throughput and crossing time when vehicle density is high. An interesting future work would be to extend Comheolaíocht scope to support entity grouping together (i.e., coalitions) which can be use to implement platoons.

6.4.3 Protocol's reliability evaluation

A goal of Comheolaíocht is to ensure that the developed protocols are reliable; that the protocol is able to ensure system's safety despite communication and entity errors. This section presents simulation results of the intersection collision scenario implemented with various versions of Comheolaíocht-developed protocols. The following sub-section presents the results focusing on how errors in communication affects the system. The next sub-section then presents simulation results of a scenario where there are simulated vehicle failures in the junction.

6.4.3.1 Communication errors

This section presents simulation results of the intersection scenarios with simulated communication errors (Figure 6.14). As mentioned, communication errors in the simulations are simulated using two parameters describing: i) the probability of a geocast failure, and ii) the probability of each message failure given that the geocast has failed. Three versions of Comheolaíocht-developed protocols are compared in this section: *no optimization*, *delay resend* and *preemption*. In each simulation run, the parameter describing "the probability of a message failure given geocast has failed" is set at 70%, in addition, the *total collision* model is simulated using Comheolaíocht's *preemption* version with this parameter initialized to 100%. The other communication error parameter, the "probability of a geocast failure", is simulated using the values of {5%, 10%, 20%, 30%, 50%}.

The graphs in Figure 6.14 show the results from the simulation. These graphs (Figure 6.14.1 to 5) compares throughput and average crossing time (*y-axis*; figure *a* and *b* respectively) over the probability of a geocast error (*x-axis*) for scenarios with different vehicle density. In each graph, four lines are plotted to represent the *no optimization*, *delay resend*, *preemption* and preemption with *total collision* model. Each point on the graph is calculated by averaging over four sets of simulations where each set runs six hours of simulation time.

There are no collision recorded in all simulation runs - showing that Comheolaíocht-developed protocols ensure entity's exclusive access to the junction despite communication failures.

In the simulations, the *total collision* simulation set serves as a baseline, and the other protocols show the effect of 70% of single message losses (given that the geocast transaction fails). Throughput and average crossing time are chosen to evaluate the simulations because the former is a measurement of the system's progress (e.g., live-locks and deadlocks) and the later measures entities' progress.

As expected, simulation runs with *total collision* provide the highest throughput and the lowest average crossing time. Conversely, simulation runs using the *no optimization* version have the lowest throughput and the highest average crossing time.

An interesting observation is that the *preemption* version provides the same throughput as the *total collision* simulations. The *preemption* version's i) internal simulation protocol and ii) preemption β-constraint, allows the entity's to claim junction resources that would otherwise be wasted due to single message failures. This non-wastage of resources is the reason for the *preemption* version having the same throughput as the *total collision* version. In contrast, simulations using the other two versions have significantly lowered throughput when the error rate is increased.

The average crossing time for simulations using the *preemption* version is higher than that of the *total collision* version. This is because, entities in the *total collision* version can use consensus to determine whether another vehicle can cross the junction compared to the *preemption* version which requires two rounds of communication without errors to prevent live-locks.

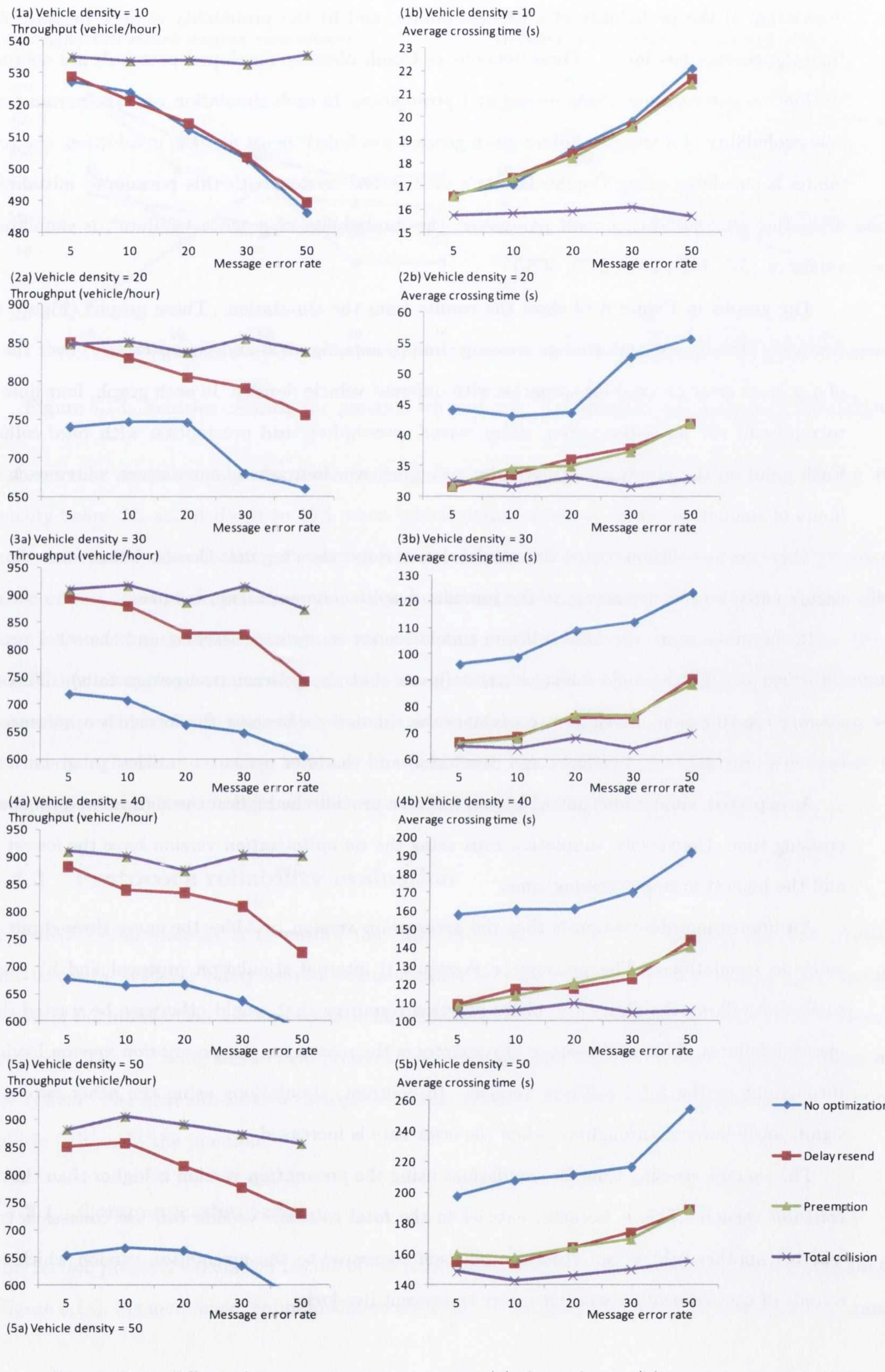


Figure 6.14: Effect of communication errors on (a) throughput, (b) average crossing time.

6.4.3.2 Vehicle breakdown

This section presents simulation results with simulated vehicle breakdowns in the junction. This simulation is only carried out for Comheolaíocht's preemption version and for vehicle density = 20. Simulations are evaluated using these values because these scenarios have the most vehicles moving at high speed. It is expected that scenarios with more vehicles traveling at a higher speed are more complex when one of the vehicle breaksdown. The table below shows the results of the simulation. Each row in the table corresponds to the average of four rounds of simulation with each round equivalent to 24 hours of simulation time.

Communication error	Vehicles	Average cross time	Maximum cross time	Breakdowns	Crashes
0%	19187	34.8	241.6	646	0
5%	18869	36.3	254.6	662	0
10%	18859	36.2	238	675	0
20%	18356	38.4	249	667	0
30%	18234	39.2	274.2	609	0
50%	17187	44.9	325	569	0

As can be seen from the table above, there are no collisions recorded in the simulations despite more than 14,000 breakdowns simulated with varying degrees of communication errors.

6.4.3.3 Conclusion

In conclusion, simulation results in this section confirm that Comheolaíocht-developed protocols are safe - they ensures that the safety constraints are not violate (i.e., no collisions in the scenario) despite imperfect communication and entity failures. The simulation results also confirm that Comheolaíocht-developed protocols prevents live-locks.

6.4.4 Protocol's scalability evaluation

A challenge is to build coordination solutions that scale when the number of entities increase in the system. Comheolaíocht-developed protocols overcome the scalability problem by utilizing local coordination, thereby providing an upper bound to computational resource usage (i.e., bandwidth, memory and processing power). This section presents simulation results that confirm that Comheolaíocht-developed protocols provide such upper bounds. In particular, the following sub-section presents simulation results focusing on resources usage parameters (i.e., messages received per second, number of records stored in memory) in the intersection collision scenario with different vehicle density values. The next sub-section then presents the simulation results implementing the Liffey scenario. Results in the Liffey scenario show that i) Comheolaíocht can be used to develop protocols for complex environments where multiple scenarios are combined together, and ii) that protocols of the Liffey scenario are reliable and scalable.

The simulation results presented in this section (both intersection collision avoidance and Liffey scenarios) use the Comheolaíocht's *preemption* version of the protocol, each data point in the graphs records the average over four rounds of six hours of simulated time.

6.4.4.1 Higher vehicle density

This section presents simulation results for the intersection collision avoidance scenario focusing on two parameters: messages received per second and number of records stored. The first parameter is a direct measurement of bandwidth usage and the second parameter measures memory usage. Together, these parameters affect computational power usage; computational power is required for i) deciding whether to accept or ignore a received message, and ii) to search through the accepted messages (i.e., the stored records) for a schedule (i.e., the internal simulation method that prevents live-lock).

Figure 6.15 shows the average and maximum number of requests received per vehicle in the intersection collision avoidance scenario. It can be seen that the numbers increase proportionally with the vehicle density in the junction. This increase in requests received per vehicle is the result of vehicles staying longer in the vicinity of the junction, which can be seen from Figure 6.16. Figure 6.16 plots the average number of records received divided by the average time a vehicle takes to cross the junction, it thereby shows the average records received per second for each vehicle in the junction. It can be seen from the graph that the records received per second reach a ceiling after the vehicle density increases beyond 20. This implies that the number of messages received per second is bounded, and that the system's bandwidth requirement is bounded as claimed.

Figure 6.17 plots the average (a) and maximum (b) number of records stored in vehicles' situation picture over the density of vehicles in the intersection collision avoidance scenario. The value for "number of records stored" for each vehicle is the maximum number of records stored in the vehicle's situation picture, from when the vehicle enters the scenario (at the beginning of the road) untill it leaves the scenario (after it has crossed the junction). The graph in Figure 6.17 shows that:

1. the average number of stored records is significantly lower than the number of entities in the scenario, (e.g., an entity stores an average of eight records for scenarios with 50 vehicle density).

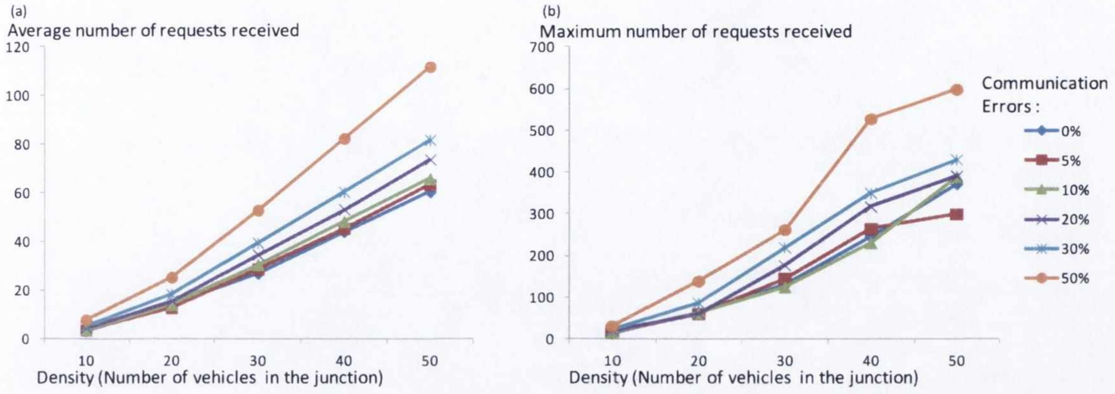


Figure 6.15: Number of requests received

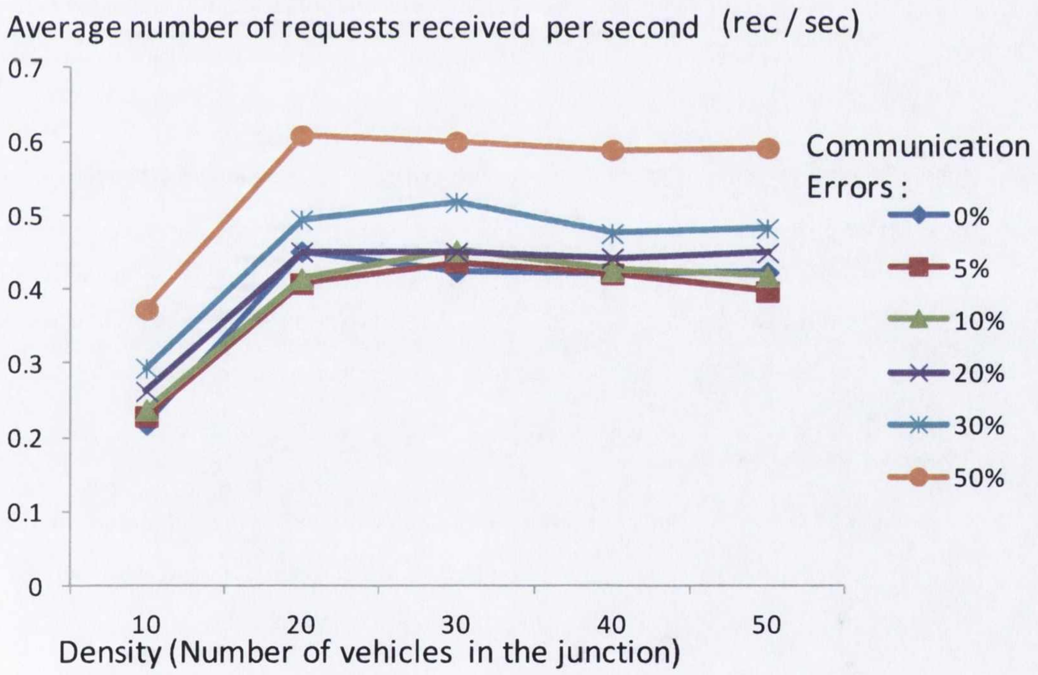


Figure 6.16: Average records received per second

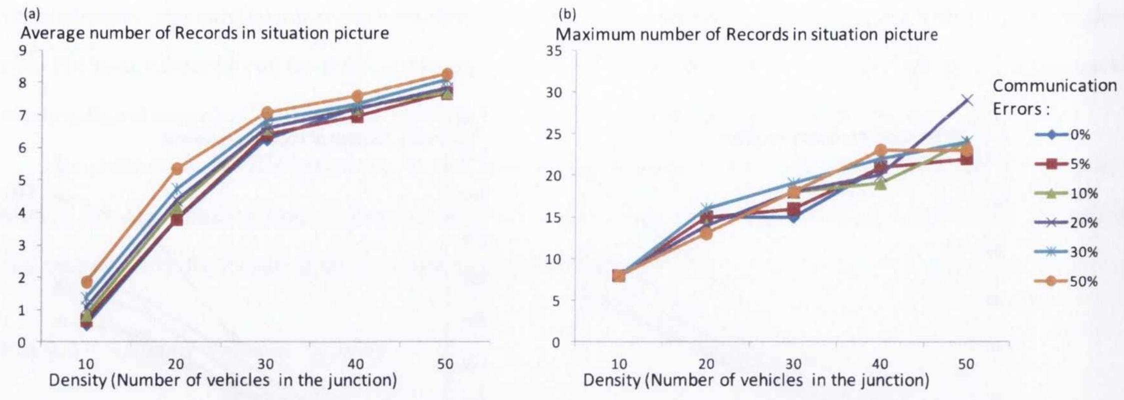


Figure 6.17: Number of records in situation picture

2. the average number of stored records only increases slowly beyond 30 vehicle density. Further simulations for 60 and 80 vehicle density show that the average number of records stabilizes at around 8.5.

3. the maximum number of records stored for a vehicle is much higher and only stabilizes at around 24. (Note: there is a vehicle with 29 records in situation picture when the vehicle density is 50, this vehicle was performing a right turn and stayed in the scenario for an extended period of time while there are more vehicles arriving and crossing during this period, resulting in the higher-than-normal number of records.)

These observations imply that the number of stored records are bounded, therefore an entity's memory usage can be bounded.

Because both stored records and received messages per second is bounded, an entity's requirement on computational resources is also bounded. Therefore, the results show that Comheolaíocht's *preemption* version of the protocol is scalable with the number of entities in the junction.

6.4.4.2 Liffey scenario

This section presents the simulation results of the Liffey scenario using the protocol developed in Section 6.3. In this simulation, the length of the middle road is ten meters; enough space only for two vehicles.

As shown in Section 6.3, there are two protocols for the scenario:

- Combined Liffey: where the middle road is represented as another junction scenario, vehicles crossing the junctions must obtain the resources for all three junctions before crossing.
- Independent Liffey: where the middle road is represented as a road scenario. Vehicles may stop in the middle road to coordinate their crossing of the second junction.

As mentioned, each data point in the graphs records the average over four rounds of six hours of simulated time.

There are no collisions recorded in the simulations of both implementations. The results of these simulations are presented in Figure 6.18. It can be seen from the graphs that:

1. The throughput (Figure 6.18(a)) and average cross time (Figure 6.18(b)) graph patterns are similar to those of the single junction scenario. For instance, the throughput hits a maximum at vehicle density 20, and the vehicle crossing time increases when the vehicle density increases. However, in the Liffey scenarios, throughput is lower and crossing times are higher than the intersection collision avoidance scenario, this is because of the bottleneck in between the two junctions (i.e., the middle road).
2. The independent Liffey implementation only has better throughput and crossing time when the vehicle density is low (at vehicle density = 0), otherwise the combined Liffey implementation is better.
3. The number of requests sent (Figure 6.18(c)) and received (Figure 6.18(d)) is lower in the combined Liffey. Similarly, the average number of records received per second (Figure 6.18(f)) is also lower in the combined Liffey. This is because vehicles who require to cross both junctions only coordinate once using the combined Liffey, but they need to coordinate twice in the independent Liffey.
4. In contrast, the number of records stored in situation picture is lower in the independent Liffey (Figure 6.18(e)). This is because vehicles that cross only one junction do not store records for vehicles crossing the other junction.

In conclusion, results in the Liffey scenario shows that Comheolaíocht-developed Liffey scenario protocols are safe (i.e., the safety constraints are respected) and scalable (i.e., there is an upper bound on message receives per second and records stored in situation picture).

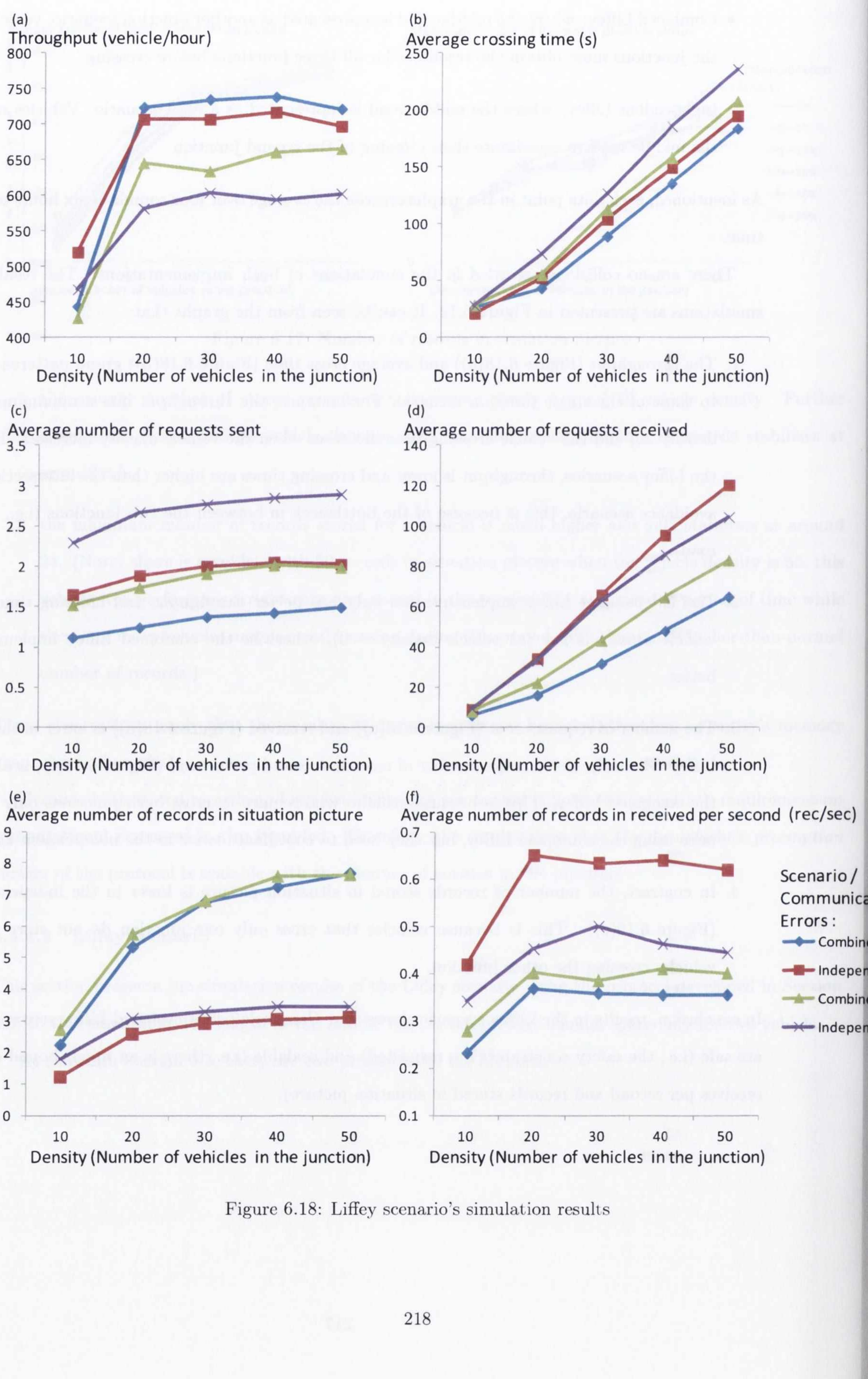


Figure 6.18: Liffey scenario's simulation results

6.4.4.3 Summary

The section showed that the two parameters: messages received per second and number of records stored dictate bandwidth, memory and computational power usage. In addition, it is shown that both intersection collision and Liffey scenarios have an upper bound on the messages received per second and number of records stored. These upper bounds can be translated to upper bounds on resources usage. As such, the results in this section confirm that Comheolaíocht-developed protocols are scalable to number of entities in the system.

6.5 Summary

This chapter has shown that protocols developed using Comheolaíocht are scalable and reliable. Comheolaíocht-developed protocols are scalable because the bandwidth, memory and computational power usage is bounded. Comheolaíocht-developed protocols are reliable because i) entities using these protocols will not violate system's safety constraints and ii) entities will not be in a live-lock or deadlock. In addition, the chapter demonstrated the use of Comheolaíocht to develop protocols for the intersection collision avoidance and Liffey scenario. These results have also been demonstrated through simulated experiments. These experiments have also shown that the protocols developed using Comheolaíocht are as efficient as specialized protocol for the intersection collision scenario.

Chapter 7

Conclusion and Future Work

This thesis presented Comheolaíocht, a systematic approach to the design of scalable and reliable protocols for multi-entity coordination. This chapter summarises the most significant achievements of the work described in this thesis and assesses its contribution to the state of the art. In addition, some suggestions for future work are outlined.

7.1 Achievements

The motivation for the work described in this thesis arose of the increasing number of multi-entity applications developed. Such applications may be safety critical (i.e., violation of these requirements might endanger human safety and possibly damage crucial or expensive infrastructure), and have progress requirements (i.e., some goals must be achieved). Protocols derived using Comheolaíocht ensure system's safety and allow progress despite imperfect perception, actuation and communication. In particular, derived protocols ensure that entities behavior leave enough allowances for possible inaccuracies (bounded) and breakdowns (with known behavior). Similarily, derived protocols are tolerant against communication delays and omission. In addition, some applications may be required to scale in the numbter of entities deployed (e.g., to support faster operation or for robot maintenance) or require support for dynamic participantion, therefore, these systems should be scalable to the number of entities.

A review of existing work has shown that there are no generic (steps for developing) coordination protocols that support dynamic participants, ensure system's safety in the presence of imperfect communication and entity failure, and ensure that there are no deadlocks or live-locks.

The design of coordination protocols that are scalable and reliable is particularly challenging because scalability in entity numbers is constrained by limited computational resources (i.e., bandwidth, memory and computational power), and system's safety and goals requirements must be satisfied despite component failures (e.g., communication, sensors and actuators). For this reason, this thesis handles scalability by building on the observation that entities in a mobile environment are typically interested in events produced by other entities within a certain geographical area, and uses local coordination to derive an upper bound for the number of entities involved in coordination. The work supports reliability by providing a systematic analysis of the application design, ensuring that entities have a strategy for acting safely even in the events of failures; protocols developed using Comheolaíocht ensure system safety by having entities act in accordance to the defined coordination strategy in the event of errors.

In order to provide a systematic approach for designing coordination protocols, this thesis started by presenting a coordination taxonomy and providing the scope of this work; developers can then check whether Comheolaíocht can be used to develop their required multi-entity coordination protocols. Utilizing a shared-resource abstraction, this work allows developers to model coordination problems (i.e., those that exhibit different-time event ordering constraints) as shared-resources which enable the use of scheduling and mutual exclusion techniques. In addition, the modeling of entities' states and their evolution as graphs allows the developers to derive a coordination strategy that specifies what an entity may or may not do in various situations to ensure safety. By applying this coordination strategy to Comheolaíocht's CwoRIS pattern (which defines a coordination scheme, a request/feedback protocol and a rescheduling method), developers can generate scalable and reliable coordination protocols for their applications.

The scalable and reliable properties of Comheolaíocht-developed protocols are shown using proofs and computational complexity calculations. In addition, the usefulness of Comheolaíocht's approach is demonstrated in the development of a complex two-intersection scenario. Simulation results of intersection scenarios are used to confirm the scalability and reliability properties of Comheolaíocht-developed protocols.

Comheolaíocht provided systematic steps for developing protocols for multi-entity systems that are both reliable and scalable; and that are not supported by existing coordination methods.

7.2 Future work

As is always the case in research, many issues are worthy of a more detailed investigation.

Comheolaíocht handles coordination problems defined within a limited scope (e.g., problems with different-time event ordering constraints, non-group based coordination, static resources). An extension to this work is to expand on the scope of problems that Comheolaíocht can derive coordination protocols for. For instance, while Comheolaíocht uses shared-resources for modeling different time event ordering constraints and uses scheduling and mutual exclusion for coordination, other classical representations like "production and consumption" and "read-write locks" can be used for the modeling of other event ordering constraints.

Comheolaíocht allows a design of coordination problems from scratch, however, there might be existing autonomous or non-autonomous (i.e., human controlled) entities in the system. A significant extension to this work is to investigate how to:

- Extend an existing system with additional entities or behaviour.
- Enable coordination between autonomous and non-autonomous entities.

In addition, most steps presented in Comheolaíocht are manual. Future work could investigate how to automate these steps so as to shorten development time and to ease programming of autonomous mobile applications even more.

7.3 Summary

This chapter summarised the motivations for, and the most signicant achievements of the work descrbed 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 the usage of scheduling methods for entities coordination. In addition, some suggestions for future work presented.

Bibliography

D. Agrawal and E.A. Abbadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Trans. on Comp. Systems (TOCS)*, 9(1):1–20, 1991. ISSN 0734-2071.

R. Alami, F. Ingrand, and S. Qutub. A scheme for coordinating multi-robot planning activities and plans execution. In *Proceedings of European Conference on Artificial Intelligence*. Citeseer, 1998.

MK Allouche and C Daigle. Tictac-a multi-agent framework for real-time coordination of temporal plans. Technical report, Report Number DRDC-VALCARTIER-TR-2003-325; Defence R&D Canada - Valcartier, Valcartier QUE (CAN), Sep 2006.

R. Aragues, J. Cortes, and C. Sagues. Distributed consensus algorithms for merging feature-based maps with limited communication. *Robotics and Autonomous Systems*, 2011.

H. Attiya, A. Kogan, and J.L. Welch. Efficient and robust local mutual exclusion in mobile ad hoc networks. In *IEEE Transactions on Mobile Comp.*, volume 9, pages 361–375, Mar 2010.

J.L. Azevedo, B. Cunha, and L. Almeida. Hierarchical distributed architectures for autonomous mobile robots: a case study. In *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, pages 973–980. IEEE, 2007.

J. Baber, J. Kolodko, T. Noel, M. Parent, and L. Vlacic. Cooperative autonomous driving: intelligent vehicles sharing city roads. *Robotics & Automation Magazine, IEEE*, 12(1):44–49, 2005.

G. Balan and S. Luke. History-based traffic control. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 616–621. ACM, 2006.

A.L.C. Bazzan. Opportunities for multiagent systems and multiagent reinforcement learning in traffic control. *Autonomous Agents and Multi-Agent Systems*, 18(3):342–375, 2009.

LB Becker, E. Nett, S. Schemmer, and M. Gergeleit. Robust scheduling in team-robotics. *Journal of Systems and Software*, 77(1):3–16, 2005.

M. Ben-Ari. *Principles of concurrent and distributed programming*. Addison-Wesley Longman, 2006.

Kenneth P Birman, Amr El Abbadi, Wally Dietrich, Thomas A Joseph, and Thomas Raeuchle. An overview of the isis project. *IEEE Distributed Processing Technical Committee Newsletter*, October 1984.

Kenneth P Birman, Thomas A Joseph, Thomas Raeuchle, and Amr El Abbadi. Implementing fault-tolerant distributed objects. *IEEE Transactions on Software Engineering*, (6):502–508, 1985.

E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford University Press, USA, 1999.

F. Borran, R. Prakash, and A. Schiper. Extending paxos/lastvoting with an adequate communication layer for wireless ad hoc networks. In *Proc. of the 27th Int'l Symp. on Reliable Distributed Systems (SRDS '08)*, pages 227–236, Oct 2008a.

F. Borran, R. Prakash, and A. Schiper. Consensus in wireless ad hoc networks. Technical report, Technical Report LSR-REPORT-2008-001, EPFL, 2008b.

M. Bouroche. *Real-Time Coordination of Mobile Autonomous Entities*. PhD thesis, University of Dublin, Trinity College, 2007.

Mélanie Bouroche, Barbara Hughes, and Vinny Cahill. Real-time coordination of autonomous vehicles. In *Intelligent Transportation Systems Conference, 2006. ITSC'06. IEEE*, pages 1232–1239. IEEE, 2006.

A. Broggi, A. Cappalunga, C. Caraffi, S. Cattani, S. Ghidoni, P. Grisleri, P.P. Porta, M. Posterli, P. Zani, and J. Beck. The passive sensing suite of the terramax autonomous vehicle. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 769–774. IEEE, 2008.

Matthew D Brown. Air traffic control using virtual stationary automata. Master's thesis, Massachusetts Institute of Technology, 2007.

Mark Campbell, Magnus Egerstedt, Jonathan P How, and Richard M Murray. Autonomous driving in urban environments: approaches, lessons and challenges. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1928):4649–4672, 2010.

Y Uny Cao, Alex S Fukunaga, Andrew B Kahng, and Frank Meng. Cooperative mobile robotics: Antecedents and directions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots'*, volume 1, pages 226–234. IEEE, 1995.

A. Casimiro and P. Verissimo. Using the timely computing base for dependable qos adaptation. In *Reliable Distributed Systems, 2001. Proceedings. 20th IEEE Symposium on*, pages 208–217. IEEE, 2001.

A. Casimiro and P. Verissimo. Generic timing fault tolerance using a timely computing base. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 27–36. IEEE, 2002.

Antonio Casimiro, Jose Rufino, Luis Marques, Mario Calha, and Paulo Verissimo. Applying architectural hybridization in networked embedded systems. In *Software Technologies for Embedded and Ubiquitous Systems*, pages 264–275. Springer, 2009.

L. Chaimowicz, B. Grocholsky, J.F. Keller, V. Kumar, and C.J. Taylor. Experiments in multirobot air-ground coordination. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 4053–4058. IEEE, 2004.

Y-I Chang, Mukesh Singhal, and Ming T Liu. A fault tolerant algorithm for distributed mutual exclusion. In *Reliable Distributed Systems, 1990. Proceedings., Ninth Symposium on*, pages 146–154. IEEE, 1990.

B. Charron-Bost and A. Schiper. Harmful dogmas in fault tolerant distributed computing. *ACM SIGACT News*, 38(1):53–61, 2007.

G. Chockler, M. Demirbas, S. Gilbert, N. Lynch, C. Newport, and T. Nolte. Reconciling the theory and practice of (un)reliable wireless broadcast. In *4th Int'l Workshop on Assurance in Distributed Systems and Networks (ADSN) (ICDCS'05)*, pages 42–48, Jun 2005.

G. Chockler, M. Demirbas, S. Gilbert, N. Lynch, C. Newport, and T. Nolte. Consensus and collision detectors in radio networks. *Distributed Computing*, 21(1):55–84, 2008.

G. Chung, K. Jeffay, and H. Abdel-Wahab. Dynamic participation in a computer-based conferencing system. *Computer communications*, 17(1):7–16, 1994.

Edward G Coffman, Melanie Elphick, and Arie Shoshani. System deadlocks. *ACM Computing Surveys (CSUR)*, 3(2):67–78, 1971.

T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms.* MIT press, 2001.

A. Cornejo, F. Kuhn, R. Ley-Wild, and N. Lynch. Keeping mobile robot swarms connected. *Distributed Computing*, pages 496–511, 2009.

P. Costa, L. Mottola, A.L. Murphy, and G.P. Picco. Teenylime: transiently shared tuple space middleware for wireless sensor networks. In *Proceedings of the international workshop on Middleware for sensor networks*, pages 43–48. ACM, 2006.

P. Costa, L. Mottola, A. Murphy, and G. Picco. Programming wireless sensor networks with the teeny lime middleware. *Middleware 2007*, pages 429–449, 2007.

R. Cunningham and V. Cahill. Time bounded medium access control for ad hoc networks. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 1–8. ACM, 2002.

C. Curino, M. Giani, M. Giorgetta, A. Giusti, A.L. Murphy, and G.P. Picco. Mobile data collection in sensor networks: The tinylime middleware. *Pervasive and Mobile Computing*, 1(4):446–469, 2005.

T.S. Dao. *A decentralized approach to dynamic collaborative driving coordination.* PhD thesis, University of Waterloo, 2008.

T.S. Dao, C.M. Clark, and J.P. Huissoon. Distributed platoon assignment and lane selection for traffic flow optimization. In *Intelligent Vehicles Symposium, IEEE*, pages 739–744. IEEE, 2008a.

T.S. Dao, L. Ng, C.M. Clark, and J.P. Huissoon. Realtime experiments in markov-based lane position estimation using wireless ad-hoc network. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 901–906. IEEE, 2008b.

D. de Oliveira, A.L.C. Bazzan, and V. Lesser. Using cooperative mediation to coordinate traffic lights: a case study. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 463–470. ACM, 2005.

M.B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, 2006.

S. Dolev, S. Gilbert, L. Lahiani, N. Lynch, and T. Nolte. Timed virtual stationary automata for mobile networks. *Principles of Distributed Systems*, pages 130–145, 2006.

K. Dresner. *Autonomous Intersection Management.* PhD thesis, The Univ. of Texas at Austin, Dec 2009.

K. Dresner and P. Stone. Multiagent traffic management: A reservation-based intersection control mechanism. In *Proc. of the 3rd Int'l Joint Conf. on Autonomous Agents and Multiagent Systems*, volume 2, pages 530–537. IEEE Computer Society, 2004. ISBN 1581138644.

K. Dresner and P. Stone. Multiagent traffic management: An improved intersection control mechanism. In *Proc. of the 4th int'l joint conf. on Autonomous agents and multiagent systems*, pages 471–477. ACM, 2005. ISBN 1595930930.

K. Dresner and P. Stone. Multiagent traffic management: Opportunities for multiagent learning. *Learning and Adaption in Multi-Agent Systems*, pages 129–138, 2006a.

K. Dresner and P. Stone. Human-usable and emergency vehicle-aware control policies for autonomous intersection management. In *Fourth International Workshop on Agents in Traffic and Transportation (ATT), Hakodate, Japan*, 2006b.

K. Dresner and P. Stone. Sharing the road: autonomous vehicles meet human drivers. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1263–1268. Morgan Kaufmann Publishers Inc., 2007.

K. Dresner and P. Stone. Mitigating catastrophic failure at intersections of autonomous vehicles. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1393–1396. International Foundation for Autonomous Agents and Multiagent Systems, 2008a.

K. Dresner and P. Stone. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research*, 31(1):591–656, 2008b.

G. Dudek, M.R.M. Jenkin, E. Milios, and D. Wilkes. A taxonomy for multi-agent robotics. *Journal of Autonomous Robots*, 3(4):375–397, 1996.

J. Elizondo-Leal, G. Ramírez-Torres, and G. Pulido. Multi-robot exploration and mapping using self biddings. *Advances in Artificial Intelligence–IBERAMIA 2008*, pages 392–401, 2008.

W. Emmerich. Software engineering and middleware: a roadmap. In *Proceedings of the Conference on The future of Software engineering*, pages 117–129. ACM, 2000.

M. Fallon, G. Papadopoulos, and J. Leonard. Cooperative auv navigation using a single surface craft. In *Field and Service Robotics*, pages 331–340. Springer, 2010.

A. Farinelli, L. Iocchi, and D. Nardi. Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(5):2015–2028, 2004.

J. Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 222. Addison-Wesley London, 1999.

D. Ferguson, M. Darms, C. Urmson, and S. Kolski. Detection, prediction, and avoidance of dynamic obstacles in urban environments. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 1149–1154. IEEE, 2008.

Robert A Ferlis. Infrastructure collision-avoidance concept for straight-crossing-path crashes at signalized intersections. *Transportation Research Record: Journal of the Transportation Research Board*, 1800(1): 85–91, 2002.

M.J. Fischer, N.A. Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985. ISSN 0004-5411.

B. Fletcher. Uuv master plan: a vision for navy uuv development. In *OCEANS 2000 MTS/IEEE Conference and Exhibition*, volume 1, pages 65–71. IEEE, 2000.

C.L. Fok, G.C. Roman, and G. Hackmann. A lightweight coordination middleware for mobile computing. In *Coordination Models and Languages*, pages 135–151. Springer, 2004.

I. Foster. *Designing and building parallel programs*, volume 95. Addison-Wesley Reading, MA, 1995.

C. Frese and J. Beyerer. A comparison of motion planning algorithms for cooperative collision avoidance of multiple cognitive automobiles. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 1156–1162. IEEE, 2011.

K.K. Fullam, T.B. Klos, G. Muller, J. Sabater, A. Schlosser, Z. Topol, K.S. Barber, J.S. Rosenschein, L. Vercouter, and M. Voss. A specification of the agent reputation and trust (art) testbed: experimentation and competition for trust in agent societies. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 512–518. ACM, 2005.

G. Gaertner and V. Cahill. Understanding link quality in 802.11 mobile ad hoc networks. *Internet Computing, IEEE*, 8(1):55–60, 2004.

R. Garud and A. Kumaraswamy. Changing competitive dynamics in network industries: An exploration of sun microsystems' open systems strategy. *Strategic Management Journal*, 14(5):351–369, 1993.

D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1):80–112, 1985.

B. Gerkey, R.T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, pages 317–323. Citeseer, 2003.

B.P. Gerkey and M.J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

R. Grabowski and A. Christiansen. A simplified taxonomy of command and control structures for robot teams. Technical report, DTIC Document, 2005.

C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored mdps. *Advances in neural information processing systems*, 14:1523–1530, 2001.

A. Gupta, X. Lin, and R. Srikant. Low-complexity distributed scheduling algorithms for wireless networks. *IEEE/ACM Transactions on Networking (TON)*, 17(6):1846–1859, 2009.

G. Hackmann, C. Gill, and G.C. Roman. Towards a real-time coordination model for mobile computing. In *Proceedings of the 12th Monterey conference on Reliable systems on unreliable networked platforms*, pages 184–202. Springer-Verlag, 2005.

S. Hadim, J. Al-Jaroodi, and N. Mohamed. Trends in middleware for mobile ad hoc networks. *Journal of Communications*, 1(4):11–21, 2006.

A. Ho, S. Smith, and S. Hand. On deadlock, livelock, and forward progress. *Technical Report, University of Cambridge, Computer Laboratory (May 2005)*, 2005.

B. Horling and V. Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(04):281–316, 2004.

J.P. How, C. Fraser, K.C. Kulling, L.F. Bertuccelli, O. Toupet, L. Brunet, A. Bachrach, and N. Roy. Increasing autonomy of uavs. *Robotics & Automation Magazine, IEEE*, 16(2):43–51, 2009.

J. Huang, Z. Han, M. Chiang, and H.V. Poor. Auction-based resource allocation for cooperative communications. *Selected Areas in Communications, IEEE Journal on*, 26(7):1226–1237, 2008.

B. Hughes. *Hard real-time communication for mobile ad hoc networks.* PhD thesis, University of Dublin, Trinity College, 2006.

B. Hughes and V. Cahill. Exploiting space/time trade-offs in real-time mobile ad hoc networks. *Int'l Journal of Mobile Network Design and Innovation*, 3(1):21–32, 2009. ISSN 1744-2869.

B. Hughes, R. Meier, R. Cunningham, and V. Cahill. Towards real-time middleware for vehicular ad hoc networks. In *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*, pages 95–96. ACM, 2004.

M. Hulsen, J.M. Zollner, and C. Weiss. Traffic intersection situation description ontology for advanced driver assistance. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 993–999. IEEE, 2011.

K. Ioannidis, G.C. Sirakoulis, and I. Andreadis. Cellular ants: A method to create collision free trajectories for a cooperative robot team. *Robotics and Autonomous Systems*, 59(2):113–127, 2011.

ISO. Information technology - syntactic metalanguage - extended bnf. *International Standards Organization and International Electrotechnical Commission (ISO/IEC)*, 1996.

GR Jagadeesh and T. Srikanthan. Route computation in large road networks: a hierarchical approach. *Intelligent Transport Systems, IET*, 2(3):219–227, 2008.

E.G. Jones, M.B. Dias, and A. Stentz. Time-extended multi-robot coordination for domains with intra-path constraints. *Autonomous Robots*, 30(1):41–56, 2011.

S. Kato, S. Tsugawa, K. Tokuda, T. Matsui, and H. Fujii. Vehicle control algorithms for cooperative driving with automated vehicles and intervehicle communications. *Intelligent Transportation Systems, IEEE Transactions on*, 3(3):155–161, 2002.

D.B. Kingston, W. Ren, and R.W. Beard. Consensus algorithms are input-to-state stable. In *American Control Conference, 2005. Proceedings of the 2005*, pages 1686–1690. IEEE, 2005.

H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. Robocup: A challenge problem for ai. *AI magazine*, 18(1):73, 1997.

J.R. Kok, M.T.J. Spaan, and N. Vlassis. Non-communicative multi-robot coordination in dynamic environments. *Robotics and Autonomous Systems*, 50(2):99–114, 2005.

H. Kopetz. *Real-time systems: design principles for distributed embedded applications*, volume 25. Springer-Verlag New York Inc, 2011.

J. Kramer and J. McGee. Concurrency: State models and java programs. *ISBN: 0-471-98710-7, John Wiley and Sons*, 1999.

J. Kramer and M. Scheutz. Development environments for autonomous mobile robots: A survey. *Journal of Autonomous Robots*, 22(2):101–132, 2007.

A.D. Kshemkalyani and M. Singhal. Necessary and sufficient conditions on information for causal message ordering and their optimal implementation. *Distributed Computing*, 11(2):91–111, 1998.

E. Kutanoglu and S.D. Wu. On combinatorial auction and Lagrangean relaxation for distributed resource scheduling. *IIE Transactions*, 31(9):813–826, 1999. ISSN 0740-817X.

L. Lamport. Paxos made simple. *ACM SIGACT News*, 32(4):18–25, 2001.

L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

N. Lau, L.S. Lopes, G. Corrente, and N. Filipe. Multi-robot team coordination through roles, positionings and coordinated procedures. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 5841–5848. IEEE, 2009.

T. Lemaire, R. Alami, and S. Lacroix. A distributed tasks allocation scheme in multi-uav context. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3622–3627. IEEE, 2004.

C.S. Li and R. Ramaswami. Automatic fault detection, isolation, and recovery in transparent all-optical networks. *Lightwave Technology, Journal of*, 15(10):1784–1793, 1997.

L. Li and F.Y. Wang. Cooperative driving at adjacent blind intersections. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 1, pages 847–852. IEEE, 2005.

C.H. Liu, A. Gkelias, Y. Hou, and K.K. Leung. A distributed scheduling algorithm with qos provisions in multi-hop wireless mesh networks. In *IEEE International Conference on Wireless & Mobile Computing, Networking & Communication*, pages 253–258. IEEE, 2008.

C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

N.A. Lynch. *Distributed algorithms.* Morgan Kaufmann, 1996. ISBN 1558603484.

R.P. Maccubbin, B.L. Staples, and M.R. Mercer. Intelligent transportation systems benefits and costs. Technical report, 2003.

D.C. MacKenzie. Collaborative tasking of tightly constrained multi-robot missions. In *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings of the 2003 International Workshop on Multi-Robot Systems*, volume 2, 2003.

M. Maekawa. An algorithm for mutual exclusion in decentralized systems. *ACM Trans. on Comp. Systems (TOCS)*, 3(2):145–159, 1985. ISSN 0734-2071.

G. Manimaran and C.S.R. Murthy. A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis. *Parallel and Distributed Systems, IEEE Transactions on*, 9(11): 1137–1152, 1998.

R. Meier and V. Cahill. Steam: Event-based middleware for wireless ad hoc networks. In *Proc. 22nd Int'l Conf. on Distributed Comp. Systems Workshops, 2002.*, pages 639–644. IEEE, 2002. ISBN 0769515886.

Luís Mota, Luís Paulo Reis, and Nuno Lau. Multi-robot coordination using setplays in the middle-size and simulation leagues. *Mechatronics*, 21(2):434–444, 2011.

U. Munz, A. Papachristodoulou, and F. Allgower. Delay-dependent rendezvous and flocking of large scale multi-agent systems with communication delays. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 2038–2043. IEEE, 2008.

A.L. Murphy, G.P. Picco, and G.C. Roman. Lime: A middleware for physical and logical mobility. In *Distributed Computing Systems, 2001. 21st International Conference on.*, pages 524–533. IEEE, 2001.

R. Naumann, R. Rasche, and J. Tacken. Managing autonomous vehicles at intersections. *Intelligent Systems and their Applications, IEEE*, 13(3):82–86, 2002. ISSN 1094-7167.

E. Nett and S. Schemmer. Reliable real-time communication in cooperative mobile applications. *Computers, IEEE Transactions on*, 52(2):166–180, 2003.

E. Nett, M. Gergeleit, and M. Mock. Mechanisms for a reliable cooperation of vehicles. In *High Assurance Systems Engineering, 2001. Sixth IEEE International Symposium on*, pages 75–81. IEEE, 2001.

R. Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory. *Automatic Control, IEEE Transactions on*, 51(3):401–420, 2006.

R. Olfati-Saber, J.A. Fax, and R.M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

A. Omicini and S. Ossowski. Objective versus subjective coordination in the engineering of agent systems. *Intelligent information agents*, pages 179–202, 2003.

D. Padua. *Encyclopedia of parallel computing.* Springer-Verlag New York Inc, 2011.

L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

Z. Papp, C. Brown, and C. Bartels. World modeling for cooperative intelligent vehicles. In *IEEE Intelligent Vehicles Symposium, 2008*, pages 1050–1055. IEEE, 2008.

C.A.C. Parker and H. Zhang. Cooperative decision-making in decentralized multiple-robot systems: The best-of-n problem. *IEEE/ASME Transactions on Mechatronics*, 14(2):240–251, 2009.

L.E. Parker. Distributed intelligence: Overview of the field and its application in multi-robot systems. *Journal of Physical Agents*, 2(1):5–14, 2008.

H.V.D. Parunak, S. Brueckner, M. Fleischer, and J. Odell. A design taxonomy of multi-agent interactions. *Agent-Oriented Software Engineering IV*, pages 123–137, 2003.

M. Pavone, K. Savla, and E. Frazzoli. Sharing the load. *Robotics & Automation Magazine, IEEE*, 16(2): 52–61, 2009.

D. Peleg. Distributed coordination algorithms for mobile robot swarms: New directions and challenges. *Distributed Computing–IWDC 2005*, pages 1–12, 2005.

M. Pinedo. *Scheduling: theory, algorithms, and systems.* Springer Verlag, 2008. ISBN 0387789340.

L.R. Pondy and I.I. Mitroff. Beyond open system models of organization. *Research in organizational behavior*, 1(1):3–39, 1979.

G. Pritschow, C. Daniel, G. Junghans, and W. Sperling. Open system controllers–a challenge for the future of the machine tool industry. *CIRP Annals-Manufacturing Technology*, 42(1):449–452, 1993.

F. Qu, F.Y. Wang, and L. Yang. Intelligent transportation spaces: vehicles, traffic, communications, and beyond. *Communications Magazine, IEEE*, 48(11):136–142, 2010.

B. Randell, P. Lee, and P.C. Treleaven. Reliability issues in computing system design. *ACM Computing Surveys (CSUR)*, 10(2):123–165, 1978.

W. Ren and N. Sorensen. Distributed coordination architecture for multi-robot formation control. *Robotics and Autonomous Systems*, 56(4):324–333, 2008.

W. Ren, R.W. Beard, and E.M. Atkins. A survey of consensus problems in multi-agent coordination. In *American Control Conference, 2005. Proceedings of the 2005*, pages 1859–1864. IEEE, 2005.

G. Ricart and A.K. Agrawala. An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM*, 24(1):9–17, 1981. ISSN 0001-0782.

G.C. Roman, R. Handorean, and R. Sen. Tuple space coordination across space and time. In *Coordination Models and Languages*, pages 266–280. Springer, 2006.

D.A. Roozemond. Using intelligent agents for urban traffic control systems. In *Proceedings of the international conference on artificial intelligence in transportation systems and science*, pages 69–79, 1999.

S.J. Russell and P. Norvig. *Artificial intelligence: a modern approach.* Prentice hall, 2010.

S. Sariel and T. Balch. A distributed multi-robot cooperation framework for real time task achievement. *Distributed Autonomous Robotic Systems 7*, pages 187–196, 2006.

S. Sariel, T. Balch, and J. Stack. Distributed multi-auv coordination in naval mine countermeasure missions. Technical report, 2006a.

S. Sariel, T. Balch, and J. Stack. Empirical evaluation of auction-based coordination of auvs in a realistic simulated mine countermeasure task. *Distributed Autonomous Robotic Systems 7*, pages 197–206, 2006b.

S. Schemmer. *A middleware for cooperating mobile embedded systems.* PhD thesis, Otto-von-Guericke-Universität Magdeburg, Universitätsbibliothek, 2004.

S. Schemmer and E. Nett. Achieving reliable and timely task execution in mobile embedded applications. In *Object-Oriented Real-Time Dependable Systems, 2003. WORDS 2003 Fall. The Ninth IEEE International Workshop on*, pages 61–61. IEEE, 2003a.

S. Schemmer and E. Nett. Managing dynamic groups of mobile systems. In *Autonomous Decentralized Systems, 2003. ISADS 2003. The Sixth International Symposium on*, pages 9–16. IEEE, 2003b.

S. Schemmer, E. Nett, and M. Mock. Reliable real-time cooperation of mobile autonomous systems. In *Reliable Distributed Systems, 2001. Proceedings. 20th IEEE Symposium on*, pages 238–246. IEEE, 2001.

P. Schermerhorn and M. Scheutz. Social coordination without communication in multi-agent territory exploration tasks. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 654–661. ACM, 2006.

M. Schulze, T. Makinen, J. Irion, M. Flament, and T. Kessel. Prevent-final report. *PReVENT Consortium, Sindelfingen, Germany*, 20, 2008.

A. Senart, M. Bouroche, and V. Cahill. Modelling an emergency vehicle early-warning system using real-time feedback. *International Journal of Intelligent Information and Database Systems*, 2(2):222–239, 2008.

G. Sharma, C. Joo, N.B. Shroff, and R.R. Mazumdar. Joint congestion control and distributed scheduling for throughput guarantees in wireless networks. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 21(1):5, 2010.

M.L. Sin, M. Bouroche, and V. Cahill. Scheduling of dynamic participants in real-time distributed systems. In *Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on*, pages 245–254. IEEE, 2011.

M. Slot, M. Bouroche, and V. Cahill. Membership service specifications for safety-critical geocast in vehicular networks. In *Proc. 7th Int'l Symp. on Comms. Systems Networks and Digital Signal Processing (CSNDSP)*, pages 422–426, Sep 2010.

R.G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *Computers, IEEE Transactions on*, 100(12):1104–1113, 1980.

C.C. Sotzing and D.M. Lane. Improving the coordination efficiency of limited-communication multi–autonomus underwater vehicle operations using a multiagent architecture. *Journal of Field Robotics*, 27(4):412–429, 2010.

C.C. Sotzing, J. Evans, and D.M. Lane. A multi-agent architecture to increase coordination efficiency in multi-auv operations. In *IEEE OCEANS 07*, pages 1–6. IEEE, 2007.

S.S. Stankovic, M.J. Stanojevic, and D.D. Siljak. Decentralized overlapping control of a platoon of vehicles. *Control Systems Technology, IEEE Transactions on*, 8(5):816–832, 2000.

E.R. Stollof and H. Kalla. National agenda for intersection safety. *ITE Journal*, 2002.

K.C. Tai. Definitions and detection of deadlock, livelock, and starvation in concurrent programs. In *Parallel Processing, 1994. ICPP 1994. International Conference on*, volume 2, pages 69–72. IEEE, 1994.

W.T. Tsai, X. Sun, Q. Huang, and H. Karatza. An ontology-based collaborative service-oriented simulation framework with microsoft robotics studio®. *Simulation Modelling Practice and Theory*, 16(9):1392–1414, 2008.

M. VanMiddlesworth, K. Dresner, and P. Stone. Replacing the stop sign: Unmanaged intersection control for autonomous vehicles. In *Proc. 7th int'l joint conf. on Autonomous agents and multiagent systems*, volume 3, pages 1413–1416, 2008.

P. Veríssimo and A. Casimiro. The timely computing base model and architecture. *Computers, IEEE Transactions on*, 51(8):916–930, 2002.

P. Verissimo, A. Casimiro, and C. Fetzer. The timely computing base: Timely actions in the presence of uncertain timeliness. In *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*, pages 533–542. IEEE, 2000.

P.E. Veríssimo. Travelling through wormholes: a new look at distributed systems models. *ACM SIGACT News*, 37(1):66–81, 2006.

L. Von Bertalanffy et al. The theory of open systems in physics and biology. *Science*, 111(2872):23–29, 1950.

J. Wawerla, G.S. Sukhatme, and M.J. Mataric. Collective construction with multiple robots. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2696–2701. IEEE, 2002.

J. Werfel and R. Nagpal. Extended stigmergy in collective construction. *IEEE Intelligent Systems*, pages 20–28, 2006.

J. Werfel, Y. Bar-Yam, and D. Ingber. Bioinspired environmental coordination in spatial computing systems. In *Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on*, pages 338–343. IEEE, 2008.

W. Wu, J. Cao, and J Yang. A fault tolerant mutual exclusion algorithm for mobile ad hoc networks. In *Pervasive and Mobile Comp.*, volume 4, pages 139–160, Feb 2008.

N. Xiong, J. He, Y. Yang, Y. He, T. Kim, and C. Lin. A survey on decentralized flocking schemes for a set of autonomous mobile robots. *Journal of Communications*, 5(1):31–38, 2010.

R. Yared, X. Défago, J. Iguchi-Cartigny, and M. Wiesmann. Collision prevention platform for a dynamic group of asynchronous cooperative mobile robots. *Journal of Networks*, 2(4):28–39, 2007.

M.M. Zavlanos and G.J. Pappas. Distributed formation control with permutation symmetries. In *Decision and Control, 2007 46th IEEE Conference on*, pages 2894–2899. IEEE, 2007.

X.F. Zhang, Y. Yang, M. Xie, H. Chen, and Z.P. Yu. Perception, planning and supervisory control of an unmanned vehicle for 2010 world expo. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 865–870. IEEE, 2008.

L. Zhou, X. Wang, W. Tu, G.M. Muntean, and B. Geller. Distributed scheduling scheme for video streaming over multi-channel multi-radio multi-hop wireless networks. *Selected Areas in Communications, IEEE Journal on*, 28(3):409–419, 2010.

H. Zimmermann. Osi reference model–the iso model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 28(4):425–432, 1980.

R. Zlot and A. Stentz. Market-based multirobot coordination for complex tasks. *The International Journal of Robotics Research*, 25(1):73–101, 2006.

Appendix

Part I

Code for finding Strategy

This part shows the pseudo code for searching finding a safe stragtegy from a mode transition diagram.

```
findStrategy(entrance, {}, initialized strategy, NULL)

findStrategy(curr, seenLLFSM, strategy, prev)
{
  if(curr is an exit mode)   // termination Case
     if(curr is a non-fail-safe mode)
        if(seenLLFSM is empty)
            strategy[curr].result = unsafe, RETURN;
        else
            strategy[curr].result = safe
            strategy[curr].condition = I:prev|seenLLFSM, RETURN;
      else
         strategy[curr].result = safe, RETURN;

  if(curr is a LLFSM)
     seenLLFSM += curr
  if(curr is non-fail-safe and seenLLFSM = ∅)
     strategy[curr].result = unsafe, RETURN;

  strategy[curr].result = (pending * seenLLFSM)
  boolean allSafe = false
  for each destination mode, x, of curr
  {  xDeter = findDeterministic(curr, x)
           // true if transition deterministic

     // recursive step, search depth
     if(strategy[x].result is Null)
        findStrategy(x, seenLLFSM, strategy, curr)

     // deals with loops
     if(strategy[x].result == (Pending * LLFSM_prev))
```

```
        if (LLFSM_prev is a subset of seenLLFSM)
            strategy[x].result = (Pending * seenLLFSM)
        findStrategy(x, seenLLFSM, strategy, NULL)

     // Recursive steps
     if(xDeter)
       if(strategy[x].result == Safe) // Case 1 & 2
           strategy[curr].result = Safe, RETURN;
       if(strategy[x].result == Unsafe AND seenLLFSM $\neq \emptyset$) // Case 5
           strategy[curr].result = Safe
           strategy[curr].condition = O:x|seenLLFSM, RETURN;

     if(strategy[x].result == Unsafe AND !xDeter AND
           seenLLFSM $= \emptyset$) // Case 8
       strategy[curr].result = Unsafe, RETURN;

     if(!xDeter AND (strategy[x].result == Safe OR  // Case: 3 & 4
       (strategy[x].result = Unsafe AND seenLLFSM $\neq \emptyset$))) // Case 7
           allSafe = true
  }

  If allSafe = true
     strategy[curr].result = Safe, RETURN;  // complete Case 3, 4, 7
  Else
     strategy[curr].result = Deadend, RETURN;  // Case 6, 9
}
```

Part II

Rescheduling (other methods)

.1 Earliest plan

The first strategy for getting out of a live-lock is the earliest plan method. In this method, entities request for the earliest available resource time slot; the earliest time slot is calculated from each node's internal situation picture.

Figure 1 shows a worst case based in the earliest plan model based on 3 entities, a, b and c. Compared to Figure 5.9, Figure 1 only shows the resource-time diagram; the reader can assume that the requests are delivered as in the sequence of Figure 5.9. In the figure, the small curly yellow arrows refers to an entity accepts a conflicting request when that request is delivered, the words describe the entity's new schedule on accepting those conflicting request.

Figure 1.A The scenario is the same as Figure 5.9.A where entity x's message causes entity a to backoff. Just after entity a's request is delivered, it recalculates the earliest time and send out its next request to be just after x.

Figure 1.B Entity b who have sent request b before entity a's request is delivered, heard a's delivered request and accepts it. b recalculates and sent out its next request to start just after a's first request.

Figure 1.C Entity c who have sent request c before entity b's request is delivered, heard b's delivered request and accepts it. c recalculates and sent out its next request to start just after b's first request.

Using CwoRIS's request/feedback protocol: if entity c has the highest priority at this point, it can ignores entity a's and entity b's second message and can access the shared resource after its second message is delivered.

Figure 1.D Entity a and b second request is delivered.

- If entity b has the highest priority, it can ignores a's second and c's first message. Therefore b can access the shared resource after its second message is delivered.
- Assuming that entity a has the highest priority. During the delivery of a's second message, it has heard entity$c$'s first message which is not in conflict with entity a's first message. Therefore, entity a must accepts entity c's first message and replan, a reschedules itself to after c.
  - If entity b's priority is higher than c, b ignores c first request and schedule its third request to be after a's second request.

Figure 1.E Entity c's second request is delivered. By our assumption above, entity c has the lowest priority, having heard entity b's second request, c schedule itself after b's second request.

Entity a's second request did conflict with entity c's second request, therefore, when entity a's cannot ignore entity c's second request, it sent a fourth schedule after c's second request.

Figure 1F On the delivery of entity a's fourth request, none of the other entities can ignore it, (since it has the highest priority) and entity a can access the shared resource.

Shortly after, entity b's fourth request is delivered and it acquired time slot before entity a. The final request for entity c is not shown, but since c has the lowest priority, it would accepts both a and b and reschedule itself after them.

The rational for the earliest plan reschedule method is to force the entities to request for the same resources. When the reservered resources are the same, entities can ignore requests with lower priority using the request/feedback protocol's inference and priority mechanism.

An evaluation on the worst case solve time for n nodes in a live-lock, happens when the nodes' priority, with respect to the message arrival sequence, is in a decending order. In this worst case, the entities have to spent n rounds to align their resources request, and in the $n+1$ round the highest priority node can ignores all other requests and breaks out of the live-lock. In this worst case scenario, after the highest priority node gain accesses to its required resources in the $n+1$ round, the rest of the entities competes for the next (same) set of available resources and the next highest priority node can ignores all other entities. Thereafter, an entity may acquires accesses to the shared resources every two rounds after the $n+1$ round. Therefore, in the worst case, it takes $3n-1$ rounds for all n entities to clear a live-lock.

In the first $n+1$ rounds, n entities sent a request per round, giving a total of $n(n+1)$ messages. Every two rounds after the $n+1$ round, an entity gets to access the share resources - there is one less request sent per two rounds; sum from $(n-1)$ to 1 gives $\frac{1}{2}n(n-1)$ and two rounds of it gives $n(n-1)$. Therefore the total number of request sent for all n entities to break out of a live-lock in the worst case is $n(n+1)+n(n-1)$ and in the order of $O(n^2)$.

.2 Other strategies; future work

This section presents serveral alternative strategies that can be compared in the future on the issue of breaking out of a live-lock.

The first strategy adopts the idea from the TCP/IP; CwoRIS can uses randomized or double backoff resend time so that the entities involved in the live-lock can avoid the same races in its resend. The staggering of resend time does not costs CPU and memory resources to implement, however, this solution does not guarantee that the entity may break out of a live-lock and its effectiveness depends on the number

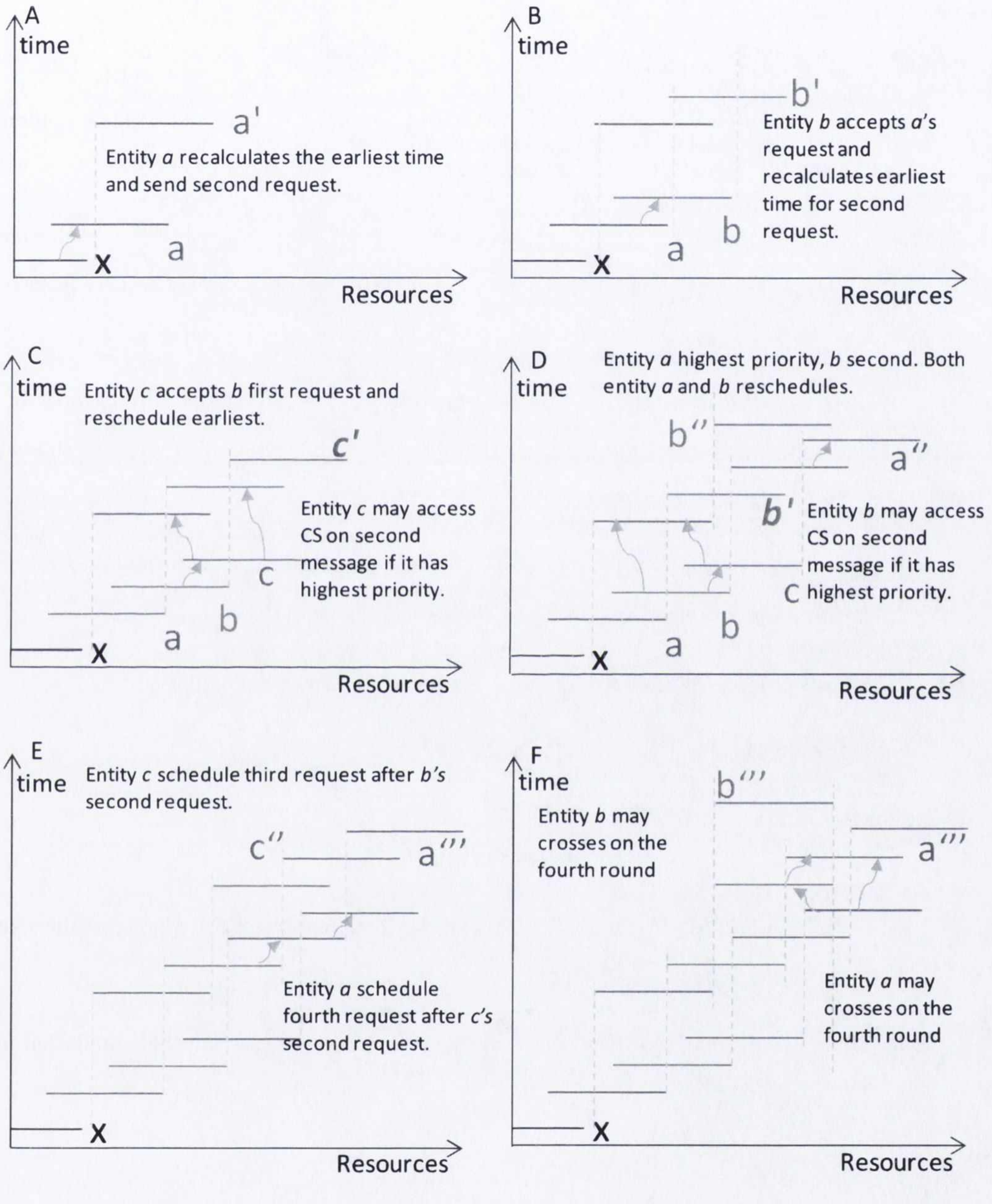

A
time
a'
Entity *a* recalculates the earliest time and send second request.
X
a
Resources
B
time
b'
Entity *b* accepts *a*'s request and recalculates earliest time for second request.
X
a
b
Resources
C
time
Entity *c* accepts *b* first request and reschedule earliest.
c'
Entity *c* may access CS on second message if it has highest priority.
c
X
a
b
Resources
D
time
Entity *a* highest priority, *b* second. Both entity *a* and *b* reschedules.
b''
a''
b'
Entity *b* may access CS on second message if it has highest priority.
C
X
a
b
Resources
E
time
Entity *c* schedule third request after *b's* second request.
c''
a'''
Entity *a* schedule fourth request after *c's* second request.
X
Resources
F
time
Entity *b* may crosses on the fourth round
b'''
a'''
Entity *a* may crosses on the fourth round
X
Resources

Figure 1: Earliest plan worst case

of entities involved in the live-lock. A future work is to optimize a good-enough timing strategy for a back-off and resend solution.

Another strategy is to have some entities back-off while others hold-wait-and-see during a conflict. Consider the example in Figure 5.9:

Figure 5.9.A Entity *c* accepts request from entity *b* and gets ready to back-off for a resend.

Figure 5.9.B When entity *c*'s request is delivered to everyone, entity *c* could have randomly decides to hold on to its request and wait (rather than resend).

Figure 5.9.D When entity *c* receives another request from entity *b* (request b'), entity *c* can infer that entity's *b* previous request - the request that prevented entity *c*'s first request from being valid - is in fact invalid. Therefore, entity *c* could actually acts on its first request.

This strategy is worth investigating in a future work because it is potentially cheaper in memory and CPU usage than the internal simulation method. However, the strategy is not investigated in this work because of two challenges:

1. Many permutations on decision to hold, wait and see. Besides using a random strategy for waiting, the strategy could include other decisions like

   (a) Wait only if < number of entities' request receives are in conflict

   (b) Wait only if < number of resources in conflict

   (c) Wait only if < percentage of time is in conflict

   (d) Wait until a double/random back-off time

2. The total number of entities involved in the live-lock may affact the quality of the solution, making this strategy a difficult to test for efficiency.

For this thesis, the evaluation of CwoRIS focuses on the deterministic internal-simulation method.