Design and Implementation of a Distributed Executive

by

Sabrina Romero

S.B. in Computer Science and Engineering
Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2023

Authored by: Sabrina Romero
Department of Electrical Engineering and Computer Science

Certified by: Brian Williams
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Design and Implementation of a Distributed Executive

by

Sabrina Romero

Submitted to the Department of Electrical Engineering and Computer Science
on , in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The deployment of autonomous robots has the potential to revolutionize high-risk missions, from rescue operations and outer-space exploration to the maintenance of underwater infrastructure. In many of these scenarios, such as the routine maintenance of a distant space station, collaboration between multiple robots becomes essential. However, the vastness of space or the depths of the oceans often impose communication constraints, making real-time coordination challenging. While centralized control of these missions is traditional and straightforward to implement, it often becomes impractical in these contexts because of communication delays and uncertainties. Given these challenges, a distributed approach is not just preferred but necessary, ensuring robots can operate independently when under limited communication conditions. Traditional strategies to coordinate multiple robots' schedules when communication is unreliable have been conservative, as they tend to favor prefixing the time for their actions, creating rigid and non-robust schedules. Such schedules can allocate excessive time to tasks as a safety measure, leaving potential resources underutilized. This over-caution not only results in inefficient execution but can also prevent the executive from identifying viable schedules for missions, even when they exist with a more flexible approach. The lack of adaptability, especially in the face of unexpected challenges, undermines the executive's robustness. To address these shortcomings, our aim is to craft a flexible and robust distributed executive adept at planning, scheduling, and executing multi-agent scenarios. We build upon the Kirk executive, a creation of the MERS group at CSAIL, enabling it to proficiently manage multi-agent scenarios without a guarantee of perfect communication during execution. Central to our methodology is the principle of temporal decoupling which allows agents to decouple any inter-dependencies in their schedule and operate independently. We integrate the state of the art algorithm in temporal decoupling, which decouples as necessary, leaving room for communication when it is available. This integration not only enhances the autonomy of the agents but also ensures they can leverage the benefits of communication when it is available, striking a balance between independence and collaborative efficiency. Building on this foundation, our work offers a practical perspective on autonomous robot coordination. By enhancing the Kirk executive with a temporal decoupling algorithm, expanding the Reactive Model-based Programming Language (RMPL) for multi-agent scenario representation, and showcasing Kirk's improved capability in multi-agent scenarios with communication constraints, we bridge the gap between theoretical foundations and practical applications.

Thesis Supervisor: Brian Williams
Title: Professor of Aeronautics and Astronautics

Acknowledgments

I would like to extend my sincere gratitude to those who have supported me throughout this journey.

To my advisor, Brian Williams, thank you for your guidance, understanding, and unwavering support. Your wisdom and patience not only helped me navigate through intense personal challenges but also shaped me to become a more resilient and thoughtful professional. Your belief in me during those critical moments has left an indelible mark on my academic journey, and I am profoundly grateful.

To my mentor, Yuening Zhang, whose steadfast presence has been a guiding light at every step of my journey, thank you. Your guidance, patience, and expertise have not only helped me achieve significant milestones but also shaped my growth as a researcher. Your mentorship has been instrumental in crafting this thesis, and your example continues to inspire me to strive for excellence. I am deeply grateful for your belief in me and the wisdom you've shared.

Special acknowledgment goes to my peers at the MERS lab: Cameron Pittman, Jake Olkin, Shashank Swaminathan, and Marlyse Reeves. Their brilliance and sense of community have not only made my master's journey much more enjoyable but have also served as a constant source of inspiration. I am grateful for their friendship, support, and the vibrant environment they create.

To Oscarito, my friend and collaborator, thank you for years of shared intellectual exploration. Your constant guidance, willingness to help when I stumble, and ability to teach new and intriguing concepts have fostered a nurturing environment for growth. Your friendship, more than just an academic connection, has been a source of joy. I'm grateful for your brilliance and your friendship.

To my boyfriend, Gaby, who has been my constant companion throughout this journey. Your encouragement, understanding, and love have been a source of strength, enabling me to face challenges and accomplish my goals. Thank you for being there every step of the way, for celebrating the triumphs and comforting me through the trials. I am profoundly grateful for everything you've done and for the person you are.

To my mom, to whom I owe the person I am today. Your nurturing guidance since childhood has ignited my intellectual curiosity and inspired me to continually seek the best

version of myself. You bravely supported the pursuit of my dreams, believing in me without hesitation. Our daily morning calls have been more than just conversations; they have been my strength and joy, a constant reminder of love, encouragement, and connection. Thank you for your unwavering faith, and your unconditional love.

Your collective wisdom, encouragement, and friendship have been the cornerstone of this thesis, and I carry your influence with me as I step into the future. Thank you all.

Contents

List of Figures

Chapter 1

Introduction

In the vast expanse of space, where the unknown beckons and challenges our understanding, autonomous robots emerge as the vanguard of exploration. Their resilience and adaptability make them ideal candidates for missions that are beyond human reach, be it the cold void of outer space or the mysterious depths of our oceans. These robotic agents, designed to operate in high-risk environments, have transformed the way we approach tasks like rescue missions, outer-space exploration, and the maintenance of underwater infrastructure.

In a scenario where a space station, orbiting a distant planet, requires routine maintenance, human intervention is not only risky but also logistically challenging. Instead, a fleet of autonomous robots, each specialized in a particular task, is deployed. These robots must collaborate, sharing resources and information, as well as coordinating their tasks, to ensure the station remains operational. However, the vast distances and the unpredictable nature of space mean that communication between these agents is sporadic at best.

Traditionally, the coordination of multi-agent missions has relied on centralized control, where a central entity oversees and directs the actions of all agents. While this approach is straightforward to implement and offers a cohesive control structure, it becomes impractical in scenarios with limited communication. The lack of autonomy of each agent prevents them from completing their tasks if they become disconnected from the base or coordinator that is sending the orders to execute the events. In the dynamic and often hostile environment of space, this lack of adaptability can be a severe limitation. Agents must be able to respond to unforeseen events, such as equipment malfunctions or sudden changes in the space station's status, without waiting for instructions from a central entity that may be delayed or entirely

unreachable.

Given these challenges, a distributed approach becomes essential, allowing robots to operate independently when required due to the mission's limited communication conditions. In the context of the space station maintenance scenario, this approach allows each robot to function autonomously. Unlike centralized control, where the failure or isolation of one agent could jeopardize the entire mission, a distributed system offers resilience. Each agent carries a portion of the overall mission knowledge, enabling it to continue its tasks even when communication with others is nonexistent. This independence, coupled with a coordinated understanding of the mission's goals, ensures that the robots can respond effectively to the dynamic challenges of space, maximizing efficiency and robustness. The distributed approach thus emerges as an essential paradigm for multi-agent missions in high-risk environments with unreliable communication.

However, the shift from centralized to distributed control isn't straightforward. Traditional methods of coordinating multiple robots' schedules in environments with unreliable communication often err on the side of caution. They lean towards creating more fixed and less flexible schedules, where the timing of actions is predetermined and unyielding so that no communication is required during execution. In the context of our space station maintenance scenario, imagine one of the robots encountering an unexpected obstacle or delay. A rigid schedule might cause the execution to fail, since the agent might be unable to adapt and continue its mission. The prefixed time for its actions would become a hindrance rather than a guide, as the robot would be constrained by the predetermined schedule that does not leave room for dynamically responding to the new situation. This conservative approach to scheduling can also lead to inefficiencies. If an overly generous time buffer is to be allocated to the task, the system may leave resources idle and underutilized. A robot that completes its task early might be forced to wait, adhering to the rigid schedule rather than seizing the opportunity to advance the mission. The challenge, therefore, lies in crafting a distributed approach that not only ensures independent operation but also retains the flexibility and adaptability necessary for the aforementioned environments.

Building on existing frameworks, we turn to the Kirk executive, a creation of the MERS group at CSAIL. The Kirk executive has already demonstrated its capabilities in managing multi-agent scenarios, but the unique challenges posed by unreliable communication require further refinement and innovation. Our goal is to enable the Kirk executive to adeptly

manage multi-agent scenarios even without a guarantee of perfect communication during execution.

In order to equip Kirk with the ability to tackle these scenarios in a distributed fashion, we leverage the principles of temporal decoupling. Temporal decoupling is a concept that allows agents to decouple the timing constraints between their actions, enabling them to operate independently while still maintaining the overall coherence of the mission. It provides a framework for agents to plan their actions in such a way that they can adapt to changes without needing to coordinate their schedules in real time.

Consider our space station maintenance scenario, where two robots are to perform interdependent tasks: one robot must repair a critical power module, and the other must recalibrate the station's communication array. The recalibration can only begin once the power module repair is completed, but the vast distances and communication constraints mean that the robots cannot coordinate in real time. Temporal decoupling allows these robots to plan their tasks with an understanding of this dependency. The robot assigned to repair the power module can operate independently, with the understanding that the recalibration robot has scheduled its task to begin at a specific time, such as 1:00 pm, after the repair is completed. This plan is in place even if direct communication between the robots is not possible. This approach ensures that the mission continues to progress smoothly, even when faced with communication challenges.

Our work entails integrating the state-of-the-art algorithm in temporal decoupling into the Kirk executive. This advanced approach adds a new dimension to the flexibility and robustness of the executive by decoupling only when necessary, and leaving space for communication when available. In the context of our scenario, if the robots know that communication will be possible at certain points, they can plan their tasks with this in mind. The recalibration robot might delay its task slightly, starting either at 1:15 pm or when it gets a notice that the repair has finished, whichever happens first. This allows for a more nuanced and flexible approach to scheduling, where the robots can coordinate more closely without sacrificing their ability to operate independently if no communication is available.

By enhancing the Kirk executive with the state-of-the-art temporal decoupling algorithm, we distribute its capabilities to handle scenarios with limited communication, enabling a dynamic reaction to contingencies. This work further includes the extension of the Reactive Model-based Programming Language (RMPL) to express the complexities of

multi-agent scenarios, allowing for a more robust representation of the complex dependencies involved. Moreover, we have undertaken the necessary transformations within Kirk to decouple the multi-agent plan input into dispatchable and independent agent plans. This ensures that each agent can operate autonomously while still working towards the collective mission goals. Through simultaneous execution of these agent plans, we demonstrate how Kirk can adeptly manage multi-agent scenarios, even in the face of communication constraints.

In bridging the gap between theoretical foundations and practical applications, this thesis contributes to the ongoing evolution of autonomous robot coordination. By expanding the RMPL for multi-agent scenario representation and showcasing Kirk's improved capability in multi-agent scenarios with communication constraints, we offer a practical perspective on autonomous robot coordination that is both innovative and applicable to real-world challenges.

Chapter 2

Background

In this chapter, we explore the foundational terminology that is used throughout our research. By providing a comprehensive overview, we hope to bridge any knowledge gaps and set the stage for the subsequent discussions.

2.1 Temporal Networks

Simple Temporal Networks with Uncertainty (STNUs) [9] and Multi-Agent Simple Temporal Networks with Uncertainty (MaSTNUs) [2] are frameworks for modeling and solving temporal problems. They are used to represent and manage the temporal constraints and uncertainties that exist in the tasks or events that agents need to perform.

2.1.1 Simple Temporal Networks with Uncertainty

An STNU is a generalized form of Simple Temporal Network (STN) that incorporates uncertainty into the duration of certain activities. In an STNU, events are modeled as nodes in a directed graph, and the temporal constraints between these events are represented as directed edges. Each edge is associated with a time interval that characterizes the duration of an activity, or the temporal restriction between two events. These constraints, or links, can be either contingent or required. While requirement links define required intervals of occurrence, contingent links capture the uncertainty in durations, indicating that the exact time of occurrence for one event cannot be controlled but is guaranteed to be within a time range from another event [9].

The primary objective in employing an STNU is to identify a feasible schedule for the

controllable events that satisfies all constraints, while also considering the uncertainty of the uncontrollable events whose time of occurrence cannot be predetermined. Therefore, what sets STNUs apart from regular STNs is the challenge of dynamic controllability. In the context of STNUs, dynamic controllability refers to determining whether there exists a strategy for executing controllable events. This strategy must take into account the observations of the uncontrollable events as they occur in real time, ensuring that all temporal constraints are satisfied [9].

Formally, an STNU can be described as:

1. **Events**: A set of events that need to be scheduled.
2. **Requirement Links**: Temporal constraints that specify a range, including a lower bound and an upper bound, within which the occurrence of two events is required.
3. **Contingent Links**: Temporal constraints that represent uncontrollable durations between events. These constraints specify a range within which the actual duration will lie.

The STNU framework serves as a powerful tool for modeling and reasoning about temporal problems where the exact durations between events are not deterministic but are constrained within known bounds. By addressing the uncertainties in temporal planning, STNUs provide a more realistic and adaptable approach to scheduling and coordination tasks in dynamic environments.

2.1.2 Multi-Agent Simple Temporal Networks with Uncertainty

A Multi-Agent Simple Temporal Network with Uncertainty (MaSTNU) extends the STNU framework to cater to environments with multiple interacting agents. In such environments, challenges like intermittent communications, varying agent capabilities, and the need for collaborative decision-making become paramount. MaSTNUs address these challenges by partitioning the STNU among a set of agents, with each agent possessing its own local network of tasks or events. These local networks interconnect through shared time constraints and dependencies, encapsulating the temporal aspects and uncertainties inherent to an STNU. A significant feature of MaSTNUs is the introduction of communication links between agents, represented by inter-agent or external contingent temporal constraints.

These constraints allow us to describe when the occurrence of an event can be observed or communicated to another agent, and with what delay. More expressive than simply assuming agents cannot communicate at all during execution, this framework allows opportunities for coordination even in scenarios characterized by limited communication [2].

Formally, a MaSTNU can be defined as follows:

1. **Local Events**: Each agent a has a set of local events V_a, partitioned into shared events V_a^S and private events V_a^P. Shared events include the reference event v_Z and events connected to external constraints. Private events are given by $V_a^P = V_a \setminus V_a^S$.

2. **Shared Events**: The set of all shared events across agents is denoted as $V_S = \bigcup_{a \in A} V_a^S$, where A is the set of all agents.

3. **Constraints**: MaSTNUs comprise two types of constraints:

   - **Local Constraints**: These are defined within each agent's local network and are analogous to the constraints in an STNU. They govern the temporal relationships between events within an agent's domain.
   - **External Constraints**: These represent interactions between different agents and include inter-agent requirement and contingent temporal constraints. External requirement constraints refer to enforceable temporal dependencies between events from different agents, while external contingent constraints, also known as communication links, model the permissible delays in communication between agents, reflecting the real-world limitations of multi-agent systems.

The mathematical structure of a MaSTNU captures the complexity of multi-agent coordination, encompassing not only the temporal aspects of task scheduling but also the nuances of inter-agent communication and collaboration. By integrating these elements, MaSTNUs provide a robust framework for modeling and solving distributed planning problems, where the interactions between agents and the uncertainties in duration of activities are central considerations.

2.2 Qualitative State Plans

While temporal networks excel in representing temporal constraints, they primarily focus on the relationships between events in terms of time. Real-world dynamic environments

often demand a more comprehensive approach, one that can capture not only the timing of events but also the evolving state of a system.

This is where state plans come into play. State plans specify a desired evolution of the plant state, defined by a set of events, coordination constraints imposing temporal constraints between events, and activities imposing constraints on the plant state [6]. Unlike STNUs, which primarily capture temporal relationships, state plans offer a more realistic representation of a system's evolving state space over time. They integrate both state variables and temporal constraints, providing a holistic view of a system's progression, and they are characterized by:

- **State Variables and Parameters:** These elements capture the dynamic properties of the system, defining its state space and potential changes over time.
- **Events:** Specific moments in time, events can be either controllable or uncontrollable, and they may encompass observations made at those moments.
- **Temporal Constraints:** These constraints dictate the temporal relationships between events, specifying the permissible sequence and time intervals for their occurrence.
- **Episodes:** Merging both temporal and state constraints, episodes connect two events with a state constraint, often linked to an activity or action an agent can undertake. They depict the progression of state over time.

2.3 Reactive Model-based Programming Language

In the quest for more adaptive and responsive planning tools, the Reactive Model-based Programming Language (RMPL) emerged as a specialized programming language tailored for planning and execution in dynamic environments. While state plans provide a comprehensive representation of a system's evolving state space over time, there's a need for a tool that allows the operator to easily specify the desired behavior of the system, especially in complex missions where adaptability and real-time responsiveness are paramount.

Developed to address the challenges of coordinating complex missions, especially in space exploration, RMPL provides a solution that is both adaptive and robust to uncertainties [4]. Traditional programming methods, with their deterministic nature, often fall short

in dynamic scenarios where adaptability is crucial. RMPL, on the other hand, combines the flexibility of embedded programming with the deliberative reasoning power of temporal planners since it only requires the operator to specify the desired high level behavior of the mission, leaving enough flexibility for the executive to reason about what it should do as situations unfold in real time, achieving greater robustness.

RMPL's strength lies in its ability to define an agent's expected behavior during execution, emphasizing expressive constraints. This design grants agents the autonomy to make decisions that adhere to these high-level specifications. The language incorporates features from both reactive programming and temporal planning, offering constructs to express concurrency, synchronization, metric constraints, and contingencies. Such a design facilitates the creation of executives of programs that can proactively reason through their execution, determining temporally consistent courses of action. This proactive reasoning is especially beneficial in real-world applications like search and rescue missions. For instance, when coordinating a group of vehicles flying from a rendezvous point to a target area, RMPL can specify alternative routes, timing constraints, and other contingencies, ensuring efficient and safe mission execution amidst uncertainties [5].

Over time, RMPL has seen several iterations, leading to its current form with a Lisp-like syntax, implemented as a Common Lisp library. This evolution has equipped RMPL to support a range of planning tools, from MERS planners to comprehensive generative planning, effectively addressing typical modeling challenges such as exceptions and observations. However, despite its advancements and capabilities, RMPL has certain limitations in expressing important details of multi-agent scenarios. Specifically, it lacks the ability to express arbitrary constraints between random events within the structure of the program, or to model when the mission does not occur under perfect communication conditions or agents cannot observe the execution of each other's events. The latter is the result of not associating agents and tasks explicitly, which leads to RMPL considering any multi-agent system as a single entity. These gaps in representation can hinder the full realization of multi-agent coordination, especially in scenarios where precise agent-task associations and inter-event constraints are crucial for successful mission execution.

In this thesis, we aim to address these limitations by introducing modifications to RMPL, enhancing its expressiveness and adaptability. By integrating features that allow for explicit task-agent associations, and the ability to define temporal requirements between arbitrary

actions of two agents, as well as the design of a communication model, we transform RMPL into a more comprehensive tool for multi-agent planning. These enhancements not only make RMPL more expressive but also ensure that it can capture the nuances and complexities inherent to multi-agent systems.

2.4 The Kirk Executive

The Reactive Model-based Programming Language (RMPL) was designed to address the complexities of planning in unpredictable and dynamic environments by allowing the operator to express high level programs that define the mission. In this context, the Kirk executive stands out as a pivotal component. Before mentioning it transforms RMPL to state plan. Kirk is an executive that can interpret and execute the RMPL program. It effectively transforms the high-level directives of RMPL into comprehensive state plans. These plans guide the behavior of autonomous agents in various real-world situations.

Upon receiving an RMPL program, the Kirk system compiles it into a state plan that it can reason with and actually execute. This plan encompasses all potential execution paths of the RMPL program, detailing every resource limitation and possible conflict between simultaneous activities. By presenting a consolidated view of the system's behavior, it ensures that the resulting plans are timely and proficient at navigating the challenges of dynamic systems.. The Kirk executive's ability to bridge RMPL's high-level directives with detailed state plans represents a notable progression in autonomous agent planning [5]. Originating from the MERS group at CSAIL, the Kirk executive not only oversees state transitions and task execution but also paves the way for rapid mission planning. It does this by conducting graph searches and utilizing symbolic constraints and decisions [10].

Our research seeks to enhance the foundational capabilities of the Kirk executive. While it was initially centralized, we have incorporated distributed functionalities to ensure that the Kirk executive can handle distributed execution of multi-agent missions under limited communication. This development highlights the Kirk executive's significance in managing and executing state plans derived from RMPL, solidifying its position in autonomous agent planning.

2.4.1 Main Components of the Kirk Executive

The Kirk executive consists of three primary components, each essential the system's operation:

- **Planner:** Tasked with plan generation, the planner takes a high-level goal specification to create a state plan. This state plan, representing a specific configuration of activities and events, is mapped to an execution schedule, defining the precise timing of events and ensuring temporal coherence during execution.
- **Scheduler:** Responsible for the timing and ordering of events, the scheduler manages the generated plans, tracking time window bounds, maintaining lists of enabled and blocking events, and overseeing event dispatching.
- **Dispatcher:** Serving as the coordinator for event execution, the dispatcher manages interactions between the executive and the child executive (a lower-level component executing the plans). It receives messages from the child executive and dispatches events, ensuring alignment with the overall mission and plan execution.

The dispatcher and scheduler are tightly integrated, collaboratively managing the timing and execution of events. The scheduler maintains scheduling information, such as time windows, bounds, and event dependencies, derived from the state plan generated by the planner.

Upon receiving messages from the executive, the dispatcher consults the scheduler to determine the next events to dispatch, considering the current time and scheduling information. The scheduler, in turn, identifies the next event by analyzing time windows and dependencies, communicating with the dispatcher to trigger the event's execution.

This coordinated interaction between the dispatcher, scheduler, and executive ensures a seamless flow of execution, aligning with the temporal constraints and dependencies defined in the plan. It exemplifies the Kirk executive's ability to provide a comprehensive solution for complex system control.

2.5 Conclusion

Throughout this chapter, we delved into the foundational concepts and tools that underpin our research. We began by exploring the intricacies of temporal networks, highlighting

their role in modeling and solving temporal problems. The introduction of STNUs and MaSTNUs showcased the evolution of these networks to accommodate uncertainties and multi-agent scenarios, respectively.

We then transitioned to state plans, emphasizing their ability to capture the evolving state of a system over time, offering a more comprehensive representation than temporal networks alone. The Reactive Model-based Programming Language (RMPL) was introduced, highlighting its pivotal role in allowing users to easily program for complex behavior of autonomous agents in dynamic and uncertain environments. In this context, the Kirk executive emerges as a key player, adeptly translating RMPL's high-level directives into actionable state plans, ensuring effective planning and execution in dynamic scenarios.

Each of these components, from temporal networks to the Kirk executive, serves as a building block in our quest to develop more adaptive and responsive planning tools for autonomous agents. Their interplay and integration form the backbone of our research, setting the stage for the innovations and enhancements we propose.

Chapter 3

Related Work

In a myriad of scenarios, from monitoring underwater infrastructure to exploring the vastness of space and executing critical rescue missions, the deployment of robotic swarms has emerged as a transformative solution. These swarms consist of individual robots or agents, each assigned specific tasks. When these agents are effectively coordinated, their combined efforts lead to the successful achievement of the overarching mission objectives.

However, the very environments where these swarms prove invaluable often present a significant challenge: unreliable or even nonexistent communication. In situations where precise coordination is of utmost importance, such communication challenges can jeopardize the entire mission. This raises a pivotal question: How can we orchestrate multiple autonomous agents to ensure cohesive operation in the face of unreliable communication pathways?

The Kirk executive developed by the MERS group used to approach multi-agent input problems in a centralized fashion. While centralized solutions offer the advantage of a unified control point, they come with inherent drawbacks, especially under limited communication. Centralized systems have a single point of failure; if the central entity faces a communication disruption, the entire coordination mechanism can collapse. Moreover, an agent could become isolated, and not be able to communicate with the rest. This underscores the pressing need to transition from centralized to distributed planning and execution model.

Historically, efforts to address this challenge have spanned a spectrum of strategies. Some of the earliest attempts, while pioneering, adopted a more conservative stance. This often translated into rigid schedules set in stone before execution. While this might be

apt for controlled environments, it becomes restrictive in dynamic settings characterized by unpredictable environmental factors and communication uncertainties. Relying solely on a fixed schedule could result in agents remaining idle, missing opportunities to engage in productive tasks. Moreover, there's a deeper concern: viability. In scenarios where tight synchronization is essential, especially amidst uncertainties, a rigid schedule might fall short. As highlighted by Stedl [8], in such situations, a dynamic execution strategy becomes indispensable, allowing agents the flexibility to adapt to unforeseen events.

In this chapter, we delve into the seminal and contemporary works that have shaped our understanding of multi-agent coordination under limited communication.

3.1 Hierarchical Distributed Coordination

A significant contribution in using a distributed approach to tackling multi-agent systems, particularly in scenarios demanding cooperation in uncontrollable environments, is the work of John Stedl [8]. He proposed a hierarchical distributed approach to coordination. At its core, this approach is characterized by a two-layer structure. The upper layer consists of co-leaders employing distributed algorithms for high-level decision-making, planning, and scheduling. Meanwhile, agents within each group can communicate reliably amongst themselves, albeit potentially facing communication barriers with agents outside their group. This design ensures that activities within each group are scheduled dynamically, while groups themselves are scheduled statically relative to each other. Attributing this structure to a multi-agent plan takes advantage of the fact that the reliability of communication increases with proximity in distance. Such a structure not only efficiently addresses communication limitations between agents but also ensures robust adaptation to temporal uncertainties during execution.

Stedl describes two main algorithms that conform his distributed executive: the Hierarchical Reformulation (HR), and the Fast Dynamic Controllability (FAST-DC) algorithms. The HR algorithm's primary objective is to transform a two-layer plan, which consists of a mission plan (representing high-level objectives) and a set of group plans (detailing the activities of individual agent groups), into a set of decoupled, minimally dispatchable group plans. A key feature of the HR algorithm is its ability to maintain synchronization between the mission and group plans. If any changes occur in the time constraints of the mission

plan, the corresponding group plans are promptly updated, and the reverse is also true. The HR algorithm employs a decoupling algorithm that operates primarily on the mission plan, and aims to assign a fixed schedule for the start of each group plan. It uses a series of transformations to build a constraint network containing only executable timepoints. Once the decoupling algorithm has determined the earliest execution times for each group activity, the start timepoints for each group plan are fixed. This results in a set of decoupled group plans, where each plan can be executed independently based on its fixed start time.

The Fast-DC Algorithm is a novel approach introduced to efficiently transform a plan constrained by an STNU into a dispatchable form. It is a significant improvement over the traditional dynamic controllability algorithm introduced by Morris in 2001 [7], as it is designed to achieve enhanced speed through several key improvements:

- Incremental Dispatchability Maintenance: The algorithm exploits the fact that a dispatchable plan can be incrementally executed during the reformulation phase. A local incremental algorithm is introduced for maintaining the dispatchability of a plan constrained by STNs. When an edge length changes in a dispatchable distance graph, only a subset of the constraints needs to be notified of this change. This change only needs to be back-propagated, similar to regression. This technique is then applied to plans constrained by STNUs by introducing a new property called pseudo-dispatchability. This removes the need to compute the APSP-graph every time a requirement edge changes.

- Integrated Constraint Propagation: The constraint propagations of requirement edges, contingent edges, and conditional edges required by dynamic controllability are combined into an efficient general framework. This framework allows different types of constraint propagation to be interleaved, reducing the number of propagations required. The idea is to apply the required tightening as soon as they can be deduced, ensuring that the next round of propagations has the most up-to-date constraint set.

- Edge Trimming: Before performing the integrated constraint propagation, the distance graph is trimmed of redundant constraints. This can significantly reduce the number of propagations required.

The Fast-DC algorithm achieves its efficiency by removing the dependence on repeated APSP (All-Pair Shortest-Path) calls, which were a bottleneck in the traditional dynamic

controllability algorithm. By combining the above improvements, the Fast-DC algorithm ensures that multi-agent plans can be executed more efficiently and adaptively, especially in the presence of uncertainties.

However, while the hierarchical distributed coordination offers numerous advantages, it's not without its limitations. The very nature of this approach, which relies on grouping agents with reliable intra-group communication, can be restrictive in modeling mission goals. There might be scenarios where no clear groups of agents with reliable communication exist. Moreover, there could be instances where the hierarchical approach doesn't suffice, necessitating a shift towards fully distributing the plans. The two-layer structure, though efficient, might not always be the most suitable for all scenarios, especially when communication dynamics are more complex or when mission objectives don't align with the proposed grouping. In addition, the strong controllability enforced among group plans might hinder the flexibility of the overall solution.

3.2 Delay Controllability

Delay controllability is a pivotal concept that models the delay in observing the occurrence of uncontrollable events. In multi-agent settings, events from other agents can be perceived as uncontrollable events. The delay in receiving observations for these uncontrollable events, or potentially never receiving them, can be used to model the fact that in environments with limited communication, agents might only be able to observe others' executed events with a delay or might never observe them at all. This perspective allows the modeling of a multi-agent scenario from a single agent's viewpoint. When multiple agents are involved, each agent must have their own view of the STNU to ensure that the delayed observation of others' events is modeled consistently.

A notable exploration into this domain is the work of Nikhil Bhargava [1]. In real-world multi-agent scenarios, agents often operate in environments where communication is unreliable or delayed. For instance, underwater robots might face significant communication lags due to the properties of water. In such situations, it's crucial to have a plan that remains valid even when agents receive information later than expected.

Under a fixed-delay controllability framework, we can model the expected delay in the exchange of information between agents. For example, if an agent knows that there's a

delay of 5 units for a specific event, and it receives the observation at time $t = 10$, it can infer that the event actually occurred at $t = 5$. However, real-world scenarios often present more complex communication patterns. Recognizing the limitations of the fixed-delay model, Bhargava introduced variable-delay controllability. This model considers a range of possible delays, acknowledging that communication might sometimes be faster or slower than expected. By modeling this uncertainty, variable-delay controllability offers a more realistic and adaptable framework for multi-agent coordination.

Cameron Pittman, as part of the MERS group, has been instrumental in integrating the ability to reason over plans where variable-delayed communication has been modeled into the Kirk executive. Pittman leveraged the variable-delay controllability framework to enhance the executive, enabling it to effectively schedule and dispatch tasks within a multi-agent network when faced with unpredictable communication delays.

As we navigate the complexities of multi-agent coordination, it's crucial to differentiate between the various methodologies in play. Both delay controllability and temporal decoupling target the challenges inherent to multi-agent systems, yet their approaches are distinct. Delay controllability offers a unique perspective by modeling the entire multi-agent scenario through the lens of a single agent. By categorizing events from other agents as 'uncontrollable events'—those that an agent cannot directly influence or predict—it adeptly captures the uncertainties tied to observing and reacting to peer actions, especially in environments riddled with communication delays. This perspective contrasts with temporal decoupling, which takes a holistic view of the multi-agent temporal plan and decouple them into individual agent networks to be executed independently. With this foundation set, let's further explore the nuances of temporal decoupling and its unique approach to multi-agent coordination.

3.3 Temporal Decoupling

Temporal decoupling emerges as a solution when a group of agents, collaborating on a set of temporally-dependent tasks, aim to coordinate their execution of these tasks. The primary goal is to apply additional temporal constraints that are sufficient to ensure that agents working on different tasks can operate independently, without the need for constant synchronization. This concept is particularly relevant in multi-agent systems where agents

must work together on tasks that have temporal dependencies, yet they seek to minimize the communication overhead and maintain their autonomy.

Consider a scenario where two agents, say Bill and Sue, are committed to tasks with specific temporal dependencies, such as Bill must finish writing a letter before Sue takes the letter to the post office before it closes. To avoid potential conflicts in their schedules, they could either add further constraints to decouple their tasks or Sue could wait for Bill to finish writing the letter. For instance, Bill could be constrained to finish the letter before noon, and Sue could take it to the post office at any point between noon and the post office's closing time. Temporal decoupling emphasizes adding more constraints, namely decoupling constraints, to avoid having to coordinate their schedules online, providing algorithms to generate additional constraints that ensure the agents' tasks remain temporally independent. The Temporal Decoupling Problem (TDP) can be described as: given a set of tasks with temporal dependencies partitioned into multiple subsets, the goal is to identify constraints for each subset such that they can be executed independently without violating any temporal dependencies [3].

Casanova et al. introduce a nuanced approach that emphasizes the challenges faced in multi-agent settings, especially when agents operate in environments characterized by intermittent communications. The primary contribution of their research is the development of distributed execution strategies that can dynamically control the MaSTNU. These strategies are derived using a centralized offline procedure that leverages a Mixed Integer Linear Programming (MILP) formulation [2]. This is more suitable if the process of solving the decoupling can be done offline at a centralized location, like at a mission center before the coordinated deployment of agents in the field.

However, while the MaSTNU framework and the approach by Casanova et al. provide a foundation for temporal decoupling in multi-agent systems, there remains a crucial aspect that has been overlooked by assuming the existence of an entity with centralized knowledge about the MaSTNU: the preservation of privacy in a multi-agent context. In practice, it is often the case that no one agent has the holistic view of the mission, or the details from other agents' networks might be kept private. Therefore, a distributed algorithm was proposed to coordinate the part of the schedule that is shared among agents, without exposing the details of local networks.

The core of Zhang's approach is the generate-and-test strategy [12], which involves

having a coordinator to generate candidate decoupling solutions using the shared constraints among the agents, then sending said solutions to each agent so that they can report any conflicts that might arise because of these new constraints. The algorithm is divided into two sub-routines, DECOUPLE and CHECKDECOUPLING.

The DECOUPLE sub-routine is pivotal in this process, encoding the problem as a MILP problem. A key insight, originally introduced by Casanova and later utilized by Zhang, is the concept of internalization. In the context of DECOUPLE, internalization refers to the introduction of local constraints, that, when enforced, entail the satisfaction of external requirement constraints. In addition, for agents to observe other agents' events, i.e. those specified by external contingent constraints, a local contingent constraint is added to capture the possible times of occurrence for the observation event. Following this, the CHECKDECOUPLING sub-routine allows agents to verify the controllability of their local network after incorporating the proposed decoupling constraints. Any conflicts identified are relayed back to the master, guiding the generation of the next decoupling solutions.

Zhang's approach fully distributes the multi-agent plans, giving agents the flexibility of independence to deal with contingencies, while allowing for the expression and utilization of communication within the swarm when it is available. This thesis focuses on integrating the temporal decoupling algorithm into the Kirk executive since its principles offer the most flexibility in distributing realistic multi-agent plans among the participating agents. Given its relevance for our work, we define the governing algorithm in the following section.

3.3.1 Privacy-Preserving Temporal Decoupling

The temporal decoupling problem aims to find a set of decoupling constraints that augment the local networks of the agents, such that when each network is executed on their own, the interdependent requirements are still satisfied under limited communication.

Mathematically, the algorithm's objective can be formulated as follows:

Given a set of agents A, shared events E, and external constraints C, information which is extracted from an MaSTNU, the goal is to find a set of local decoupling constraints $D = \{D_a\}$ for all $a \in A$, such that D_a that ensures all external constraints are satisfied without explicit coordination during execution, maintaining the dynamic controllability of all local networks and the overall solution [12].

3.4 Conclusion

The realm of multi-agent coordination, especially in environments with unreliable communication, is a complex and evolving field. The journey from the foundational Kirk executive's centralized approach to the more recent distributed strategies underscores the continuous quest for more adaptive, resilient, and efficient solutions. As we've explored in this chapter, the transition from centralized to distributed execution is not just a technological shift but a necessity driven by real-world challenges. The works of Stedl, Bhargava, Pittman, and Zhang, among others, have illuminated the path forward, each contributing unique perspectives and methodologies to address the intricate dance of multi-agent coordination. While the Kirk executive provided a robust starting point, the integration of contemporary approaches, especially temporal decoupling, promises a future where robotic swarms can operate seamlessly, even in the most challenging environments.

Chapter 4

Architecture Overview

In this chapter, we provide an overview of the architecture that facilitates multi-agent coordination under limited communication scenarios using the Kirk executive. Our goal is to give the reader a holistic understanding of the entire pipeline, from mission specification to individual agent execution, an approach that is based on the integration of the privacy-preserving algorithm for decoupling of MaSTNUs [12] into the Kirk executive.

The entire process, from mission specification to the execution of independent state plans, is visually represented in Figure 4-1. This flow diagram provides a succinct visual overview of how Kirk handles multi-agent scenarios, emphasizing the key stages and their interconnections, which we will expand on throughout this chapter.

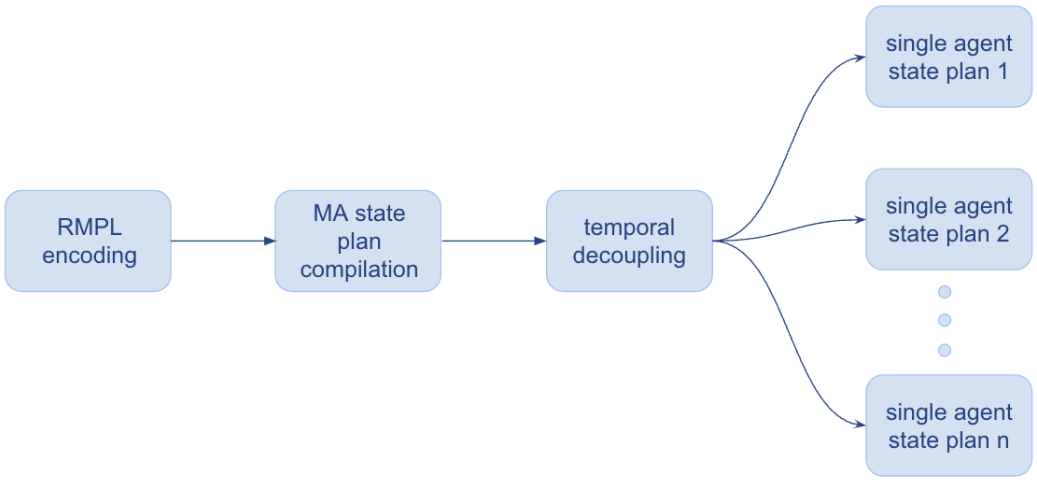

Figure 4-1: Flow Diagram Illustrating how Kirk handles multi-agent scenarios

4.1 Mission Specification in Multi-agent RMPL

A multi-agent scenario is composed of various independent agents that collaborate to achieve a common goal or mission. In our implementation, the process to execute it distributively on the Kirk executive, begins by encoding the mission goals using RMPL. To cater to the unique requirements of multi-agent scenarios, we extend RMPL to multi-agent RMPL (MaRMPL). These enhancements allow for the expression of critical information about MaSTNUs, such as agent-task association, a communication model, and the ability to express constraints between arbitrary events. For a comprehensive understanding of how mission goals are encoded using the enhanced RMPL, refer to Chapter 5.

4.2 Multi-agent State Plan Decomposition and Decoupling

Once the mission is defined in MaRMPL, it is handed over to the Kirk executive. Kirk's first task is to translate this high-level representation into a more actionable format known as the multi-agent state plan. This plan captures the temporal and state dynamics of the entire mission. During the compilation process, auxiliary events and constraints are introduced. Because these are not necessary for the execution, we simplify the plan at this stage in order to decrease the number of constraints that might need to be decoupled later on.

With the simplified multi-agent state plan in hand, the next step is its decomposition into two distinct types of temporal networks:

- Task Network: This captures the broader inter-agent temporal dynamics, highlighting the temporal interdependencies between agents.

- Agent Networks: These are localized representations focusing on individual agent dynamics.

These networks serve as the input to the privacy-preserving temporal decoupling algorithm, which we have integrated into our executive. Its output is the decoupling constraints that each agent requires for distributed execution.

4.3 Assembly of Independent State Plans

Once the decoupling constraints are obtained for each agent, we assemble individual state plans for each agent. These plans combine the original state and temporal information from the multi-agent state plan with the new decoupling constraints. Each agent's state plan becomes the goal specification for individual Kirk executive instances

4.4 Execution of Independent State Plans

Each of these single-agent state plans operates independently of the others, yet they all share the same mission starting point, which is typically a fixed reference time like 12pm. This common starting point ensures that the agents can execute their respective tasks simultaneously, maintaining alignment with the overall mission goals. Despite the lack of direct communication and coordination, the temporal decoupling ensures that the agents' actions are synchronized in a way that fulfills the mission's requirements.

While this chapter offers a high-level overview, the intricate details of each stage, including the algorithms, methodologies, and implementation nuances, will be elaborated upon in Chapter 6.

Chapter 5

RMPL modifications

Currently, RMPL is designed to specify single-agent problems. When full communication is guaranteed, RMPL can model multi-agent scenarios by treating each agent as a state variable, effectively centralizing control over all agents. However, in scenarios where communication between agents is not always guaranteed, a distributed approach to problem specification becomes necessary. The existing version of RMPL lacks the capability to express certain aspects of these distributed scenarios.

To address this limitation, this chapter introduces modifications we have made to RMPL. These changes allow RMPL to express two key ideas in the encoding of MaSTNUs: the association of an agent with primitive control programs and the establishment of arbitrary constraints and communication links between agents.

5.1 Agent Specification in RMPL

RMPL provides a structured approach to define agents as instances of classes, allowing for flexibility in adjusting to various scenarios. Agents can be tailored to fit the requirements of the scenario, which aids in reasoning over their state in primitive control programs.

5.1.1 Agent Class Definition

Consider the agent class `glider` as an example:

```
(defclass glider ()
  ((id
    :initarg :id
    :finalp t
    :type integer
    :reader id
    :documentation
    "The ID of this glider.")
   (deployed-p
    :initform nil
    :type boolean
    :accessor deployed-p
    :documentaiton
    "A boolean stating if the glider is deployed at any point in time.")
   (destination
    :initform nil
    :type (member nil "start" "end" "science-1" "science-2")
    :accessor destination
    :documentation
    "The location to which the glider is currently heading, or NIL if it is not
    in transit.")
   (location
    :initarg :location
    :initform "start"
    :type (member nil "start" "end" "science-1" "science-2")
    :accessor location
    :documentation
    "The location where the glider is currently located, or NIL if it is not at
    a location (in transit).")))
```

In RMPL, swarms can be composed of agents that belong to different classes, resulting in either homogeneous or heterogeneous configurations. This distinction is made when specifying various agent classes in the initial state of an RMPL document. These agents are

modeled as state variables. During the compilation of the state plan, these state variables take on a distinct interpretation as agents, primarily due to their subsequent association with control programs, as elaborated in the following section.

5.1.2 Primitive Control Programs

Primitive control programs represent atomic actions that can be performed by a single agent. In single-agent scenarios, tasks are implicitly performed by the agent. In contrast, multi-agent scenarios necessitate specifying which agent is responsible for each task. This is achieved by associating an agent with each primitive control program.

The general syntax for associating an agent with a control program in RMPL is:

```
(define-control-program control-program-name (agent)
  (declare (primitive agent)
           ...))
```

Where `agent` refers to the agent associated with the control program, and the `primitive` keyword indicates this is the control program of an atomic action that can be performed by the agent. Exactly one agent must be associated with each primitive control program.

For instance, consider the glider class, which has specific fields such as deployed-p to indicate if the glider is deployed and location to specify its current position. These fields can be adjusted or checked during the execution of a primitive control program.

In the `deploy` control program defined below, the `requires` clause checks the current state of the glider using its fields. Specifically, it requires that the glider is not deployed (deployed-p is nil) at the start of `deploy` and that its current location matches the specified location throughout `deploy`. The effect clause then updates the deployed-p field to indicate that the glider has been deployed. This highlights how the specifics of an agent class can be tied to the requirements of a primitive control program, and to transformations that the state space might undergo during it, allowing for dynamic and context-specific behaviors.

```
(define-control-program deploy (glider location)
  (declare (primitive glider)
           (requires (and (at :start (= (deployed-p glider) nil))
                          (over :all (= (location glider) location))))
           (effect (at :end (= (deployed-p glider) t)))
           (duration (simple :lower-bound 5 :upper-bound 5))))
```

Let's illustrate the association of agents to tasks in the following scenario where two rovers have to complete a space mission consisting of two tasks, A and B.

```
(define-control-program task-A (agent)
  (declare (primitive agent)
           (duration (simple :lower-bound 30 :upper-bound 60) :contingent t)))

(define-control-program task-B (agent)
  (declare (primitive agent)
             (duration (simple :lower-bound 45 :upper-bound 45))))

(define-control-program main (rover1 rover2)
  (with-temporal-constraint (simple-temporal :lower-bound 80 :upper-bound 90)
    (parallel (:slack t)
      (task-A rover1)
      (task-B rover2))))
```

The events and temporal constraints in the resulting state plan are represented in the figure 5-1.

In our current scenario, each episode is associated with an agent, and thus the start and end event of each episode is also associated with an agent. For instance, the start and end events of the episode `task-A` are associated with `rover-1`, and the start and end events of the episode `task-B` are associated with `rover-2`. These associations are crucial in multi-agent scenarios as they determine which agent is responsible for executing which task.

It is important to note that in order to capture all the complexity of the scenario, the resulting state plan may introduce auxiliary events that are redundant. We go through the

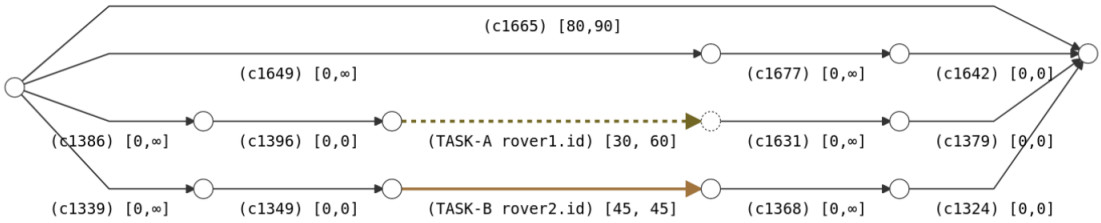

Figure 5-1: State Plan of simple parallel task execution by two agents

simplification process of the state plans in more detail in chapter 6.

5.1.3 Hierarchical Automaton in RMPL

In our scenario, the overall mission is represented by a hierarchical automaton defined by the `parallel` construct and that is composed of the tasks `task-A` and `task-B`. The hierarchy introduced here models how those two control programs are to be executed concurrently. In the state plan, the hierarchy is represented by additional auxiliary or intermediate events that manage the execution of the tasks. For instance, in figure 5-1, we have a common starting event and common ending event encapsulating the tasks, which ensures their overall execution is confined to a specific time range.

This arrangement naturally leads us to explore the underlying theoretical model that makes such concurrent execution possible: the concept of a hierarchical automaton. A hierarchical automaton is a model used to represent complex systems with hierarchical structures. In the context of RMPL, a hierarchical automaton is used internally to represent the structure of a control program that may be composed of other control programs, each defining transitions in the state in the automaton.

There are different types of hierarchical automata that can be used in RMPL, specifically "all" and "xor". The "all" type, used in the `parallel` construct, represents a situation where all child states (or tasks) of a parent state are active at the same time. In other words, all tasks within this construct are executed concurrently. On the other hand, the "xor" type represents a situation where exactly one child state of a parent state is active at a time. This type is used in the `sequence` construct, as seen in the following control program definition:

```
(define-control-program main (rover1 rover2)
  (with-temporal-constraint (simple-temporal :lower-bound 0 :upper-bound 105)
    (sequence (:slack t)
      (task-A rover1)
      (task-B rover2))))
```

Slack constraints are introduced within the `parallel` and `sequence` constructs to leave space for flexibility. Some of these introduced events are redundant and unnecessary to capture the mission. If we directly apply decoupling, these become additional events that have to be coordinated and potentially decoupled, when they are actually not necessary. Therefore, simplifying these constraints can reduce the number of external constraints that the temporal decoupling algorithm needs to internalize. Figures 5-2 and 5-3 show the resulting state plan from a simple sequence control program, and the simplified temporal relations of that state plan respectively. We expand on this simplification process in chapter 6.

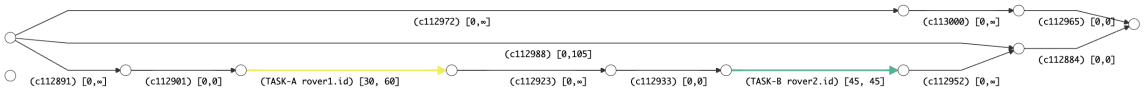

Figure 5-2: State plan of simple sequence task execution by two agents

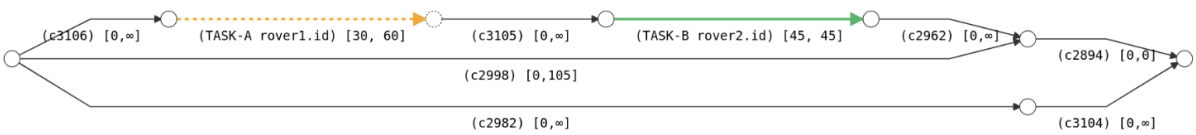

Figure 5-3: Simplified state plan of simple sequence task execution by two agents.

5.2 External Constraint Representation

In multi-agent scenarios, communication and coordination are essential. Hence, RMPL must be able to express any arbitrary temporal constraints between any two events of different agents that can be referenced, that is, any two endpoints of control programs.

5.2.1 Arbitrary Requirement Constraints

Consider a scenario where two rovers, rover1 and rover2, are performing tasks `task-A` and `task-B`, respectively. We may want to specify that rover2 should start sampling within a specific time range, say between 0 to 3 units of time, from when rover1 starts taking a picture. Additionally, there is a constraint that the parallel execution of tasks `task-A` and `task-B` must occur within a time range of 70 to 100 units.

The following code snippet illustrates how an arbitrary requirement constraint can now be added to the RMPL encoding of the scenario described above:

```
(define-control-program main (rover1 rover2)
  (let ((A (task-A rover1))
        (B (task-B rover2)))
    (with-temporal-constraint
        (simple-temporal :lower-bound 0 :upper-bound 3
                         :from (start-of B) :to (start-of A))
      (with-temporal-constraint (simple-temporal :lower-bound 70 :upper-bound 100)
        (parallel (:slack t) A B)))))
```

This goal specification results in the state plan seen in figure 5-4.

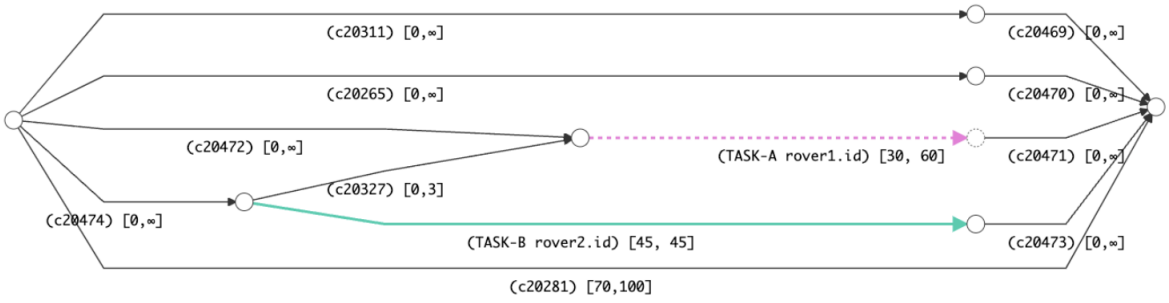

Figure 5-4: State Plan of simple parallel task execution by two agents with an arbitrary requirement constraint.

The construct `with-temporal-constraint` has been augmented to include references to the start and end points of the constraint to be introduced. These references allow for the specification of constraints between any two events in the automata being created.

A critical aspect of this approach is the way we handle instances of control programs. In

a complex scenario, we might have multiple instances of the same control program associated with the same agent. To ensure that a temporal constraint is applied to the correct instance of a control program, we store that specific instance in a variable.

For example, in the code snippet, we define variables A and B as instances of the control programs `task-A` and `task-B`, respectively. By doing so, we create a clear reference to those specific instances. Later, when we refer to `start-of A` and add A to the parallel construct, we are referring to that exact instance of the control program.

This approach provides precise control over the relationships between different parts of the program. It ensures that an arbitrary constraint is applied to the intended instance.

5.2.2 Arbitrary Contingent Constraints

Building on the ability to express arbitrary temporal constraints, we further extend our framework to handle contingent relationships between events. In our previous scenario, let's say that rover2 starts sampling once it has observed rover1 starts taking the picture. This observation could have a delay of up to 3 time units. In our RMPL framework, we have made it simple to include arbitrary contingent constraints such as those. By adding the `:contingent t` tag to an arbitrary constraint, we can define a contingent relationship between specific events. Here's how the syntax would change to make the previous constraint contingent:

```
(define-control-program main (rover1 rover2)
  (let ((A (task-A rover1))
        (B (task-B rover2)))
    (with-temporal-constraint
        (simple-temporal :lower-bound 0 :upper-bound 3
                         :from (start-of B) :to (start-of A)
                         :contingent t)
      (with-temporal-constraint (simple-temporal :lower-bound 70 :upper-bound 100)
        (parallel (:slack t) A B)))))
```

When a contingent constraint is imposed between events of two different agents, it becomes known as an external contingent constraint or a communication link [12]. In MaRMPL, we recognize two distinct types of external contingent constraints.

The first type is the arbitrary contingent constraints defined in this section. These constraints require the occurrence of an event to be communicated to another agent. Such constraints can model scenarios where explicit references of the observation events are needed, like when one rover wants to immediately start its task after receiving a signal from another rover.

The second type of external contingent constraint signifies available observation opportunities where an event can be observed by other agents. In this case, there are no temporal requirements directly associated with the observation of the event. Instead, these constraints represent opportunities for coordination that are available but not mandated. This second type is best modeled by the observation model described in section 5.3.

In the code snippet above, a contingent constraint is placed between the start of task B (executed by rover2) and the start of task A (executed by rover1). This constraint represents the scenario where rover1 must begin executing task A when it observes or receives the signal that rover2 started executing task B, and that said signal could come with a delay of 0 to 3 time units.

This contingent constraint provides a way to model the interaction between agents, where one agent's actions trigger specific behaviors in another agent, based on the observed events and the defined temporal relationships. Such contingent constraints enable more complex and coordinated multi-agent behaviors, reflecting the uncertainties and dependencies that often exist in real-world scenarios.

5.3 Observation Model

In multi-agent scenarios, the concept of observability plays a crucial role. Observability refers to an agent's ability to perceive a particular event performed by another agent, which provides additional opportunities for coordination online. For instance, an agent could take into account the observation of an event by another agent in its decoupled state plan, thus taking advantage of available opportunities for coordination. Although not yet implemented, we have designed this inter-agent observability in RMPL to be facilitated through the `observable` keyword, which is part of the declaration of a primitive event. This keyword allows certain control programs to be observed by other agents, enabling a form of inter-agent communication. The syntax for this construct is as follows:

```
(define-control-program observed-program (agentA agentB agentC)
  (declare (primitive agentA)
           (duration temporal-constraintA)
           (observable (at :start-and/or-end
                :delay (simple :lower-bound lb :upper-bound ub)
                :by (agentB agentC)))))
```

The `observable` keyword is used within the declaration of a control program to specify that the program can be observed by other agents. The event that can be observed may be the start of the control program (`at :start`), the end of the control program (`at :end`), or both the start and end of the control program (`at :start-and-end`).

The observation is subject to a delay, which is specified by a simple temporal constraint (`:delay (simple :lower-bound lb :upper-bound ub)`). This delay, which defaults to 0, represents the time window within which the observation can occur after the event has taken place.

Finally, the `:by` tag is used to specify the agents that are capable of observing the event. This is followed by a list of agents, allowing for the specification of a subset of agents that can observe the event.

Here's an example of how this might be used in practice:

```
(define-control-program collect-samples (rover1 rover2 base)
  (declare (primitive rover1)
           (duration (simple :lower-bound 30 :upper-bound 60) :contingent t)
           (observable (and (at :start
                                :delay (simple :lower-bound 0 :upper-bound 5)
                                :by (base))
                            (at :end
                                :delay (simple :lower-bound 5 :upper-bound 15)
                                :by (rover2))))))
```

The `collect-samples` control program is associated with `rover1`. This control program can be observed by other agents, specifically by agents `base` and `rover2`, at different points in time.

The `observable` declaration specifies two observable events:

1. The start of the `collect-samples` control program can be observed by `base`. The observation can occur after a delay of 0 to 5 units of time from the start of the control program.

2. The end of the `collect-samples` control program can be observed by `rover2`. The observation can occur after a delay of 5 to 15 units of time from the end of the control program.

The `duration` declaration specifies that the `collect-samples` control program has a duration of 30 to 60 units of time and is contingent, meaning that its exact duration is not under the control of `rover1`. This example demonstrates how the `observable` keyword can be used to specify multiple observable events associated with a single control program, allowing different agents to observe different events at different times.

Behind the scenes

The `(observable expression)` syntax in RMPL allows for the specification of inter-agent communication. The `expression` can either be a simple communication expression or a logical combination (conjunction/disjunction) of multiple such expressions. Each communication expression adheres to the syntax `(at :event :delay tc :by (agents))`, where:

- `:event` can be `:start`, `:end`, or `:start-and-end`, indicating the event that can be observed.

- `tc` is a temporal contingent constraint that specifies the delay bounds within which the observation can occur.

- `(agents)` is a list of agents that are capable of observing the event.

When compiling RMPL to state plan, for each agent specified in the `:by` list, an episode of zero duration is instantiated and assigned to that agent. This episode serves as a placeholder for the observation event.

Subsequently, contingent temporal constraints are created in accordance with the specified temporal constraint `tc`. These constraints become external contingent constraints linking the event specified in the `at` construct and the start event of the newly created episode. It's important to note that these external constraints have two endpoint events

that are considered shared. This is because they represent points of interaction between different agents' local networks.

The example in figure 5-5 shows a communication link representation, with A-prime event representing the observation of A.

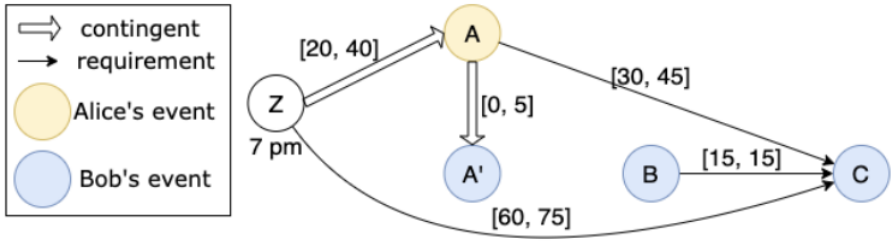

Figure 5-5: MaSTNU example with communication link from [12]

Since we do not currently count with the implementation of the `observable` construct, we solely rely on the definition of arbitrary constraints to model communication. Given a scenario as presented by the MaSTNU, where Alice's event A can be observed by Bob, we can model it using RMPL as follows:

```
(define-control-program task-A (agent observer)
  (declare (primitive agent)
           (duration (simple :lower-bound 20 :upper-bound 40)
                     :contingent t)
           (observable (at :end
                       :delay (simple :lower-bound 0
                                      :upper-bound 5)
                       :by (observer)))))

(define-control-program task-BC (agent)
  (declare (primitive agent)
           (duration (simple :lower-bound 15 :upper-bound 15))))

(define-control-program main (alice bob)
  (let ((A (task-A alice bob))
```

```
    (BC (task-BC bob)))
(with-temporal-constraint
    (simple-temporal :lower-bound 30 :upper-bound 45
                     :from (end-of A) :to (end-of BC))
    (with-temporal-constraint
        (simple-temporal :lower-bound 60 :upper-bound 75)
      (parallel (:slack nil)
        (sequence (:slack :end)
            A)
        (sequence (:slack :start)
            BC))))))
```

Chapter 6

Integrating Temporal Decoupling into the Kirk executive

In this chapter, we will embark on an exploration of how temporal details are extracted from a multi-agent state plan, forming the essential input for the temporal decoupling algorithm. The discussion will also encompass the innovative ways in which Kirk employs the decoupling constraints to craft individual state plans for each agent. These plans, though functioning independently, synchronize with the overarching mission goals, facilitating a unified execution. This chapter aims to provide an in-depth understanding of the processes and methodologies that transform a collective multi-agent plan into distinct, yet harmonized, single-agent plans.

6.1 Simplification of state plans

In the realm of RMPL, the state plan serves as a blueprint that outlines the temporal relationships between various events. However, due to the inherent flexibility and expressiveness of RMPL, the compiled state plan can sometimes contain redundant or superfluous constraints as can be seen in figure 6-1. These constraints, while useful during the compilation phase, can make the state plan more complex than necessary for decoupling, execution or human interpretation. Therefore, simplifications are introduced to streamline the state plan, making it more concise and easier to understand.

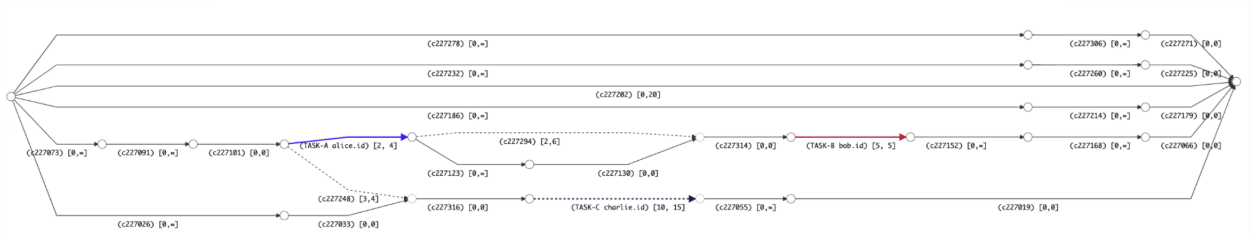

Figure 6-1: Three-agent State Plan before simplification.

Slack constraints

The state plan often includes slack constraints, which are temporal constraints that provide flexibility in the execution of events. These constraints are especially prevalent in sequence and parallel constructs, where they are introduced to allow for some leeway in the timing of events. For instance, in a sequence of events, slack might be added at the beginning, in-between events, or at the end to accommodate uncertainties or variations in execution times.

Arbitrary constraints

In addition to slack constraints, the state plan can also contain unnecessary constraints due to the introduction of arbitrary constraints in the control program. These are constraints that are not explicitly defined by the user but are added during the compilation process to ensure the feasibility of the plan. One common scenario where arbitrary constraints are introduced is when dealing with contingent constraints. For example, if there's an arbitrary contingent constraint pointing towards the start of an episode, a [0,0] constraint might be inserted to ensure said new contingent constraint ends in an uncontrollable event, which is not the case for the start event of any episode.

Simplification Process

The simplification process involves identifying and removing these redundant constraints from the state plan. By doing so, the state plan becomes more streamlined, retaining only the essential constraints that dictate the temporal relationships between events. This not only makes the state plan easier to interpret but also reduces the computational overhead during execution. In figure 6-2 we can see how our simplification process modifies the multi-agent state plan shown in 6-1.

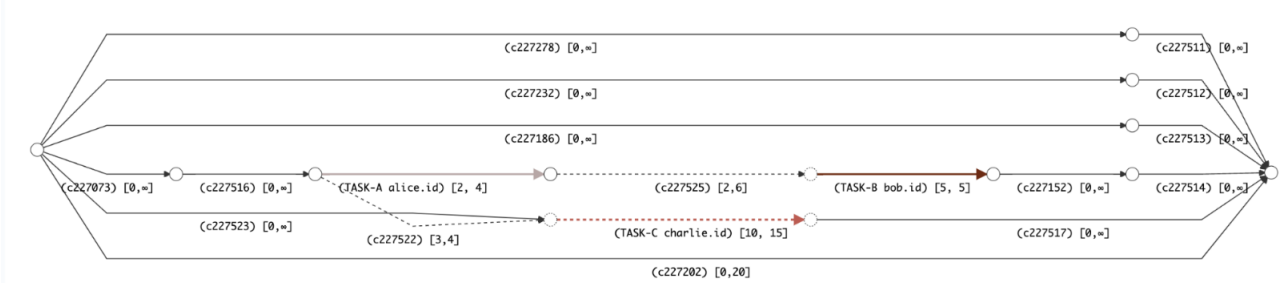

Figure 6-2: Three-agent State Plan after simplification.

Here's a brief overview of the main functions designed to identify and eliminate redundant constraints:

- eliminate-intermediate-events: This function targets intermediate events that are created by adding slack to sequence and parallel structures. It identifies these intermediate events and merges constraints surrounding them, effectively eliminating unnecessary complexity. The function also takes care of erasing repeated constraints that might arise due to the addition of arbitrary constraints.

- fix-contingent-constraints: This function focuses on handling contingent constraints within the state plan. It identifies constraints that are redundant due to the introduction of arbitrary constraints and replaces them with more streamlined versions. The function also creates new episodes and constraints as needed to maintain the integrity of the state plan.

6.2 From a multi-agent state plan to temporal networks

After the modifications explained in the previous section, we are left with a simplified multi-agent state plan, which can be considered as such because each episode is associated with an agent. In order to decouple its temporal constraints, the multi-agent state plan must be stripped of state information, that is, it must be transformed into two distinct types of temporal networks:

- **Agent Network:** The Agent Network is a localized representation that focuses on the individual temporal dynamics of each agent. It includes all internal constraints that govern the agent's behavior and shared events that the agent is willing to expose

to others. These shared events are endpoints of external constraints linking different agents, and only essential shared events are included to maintain a concise representation. The Agent Network is tailored to enable efficient detection of potential conflicts and application of privacy-preserving temporal decoupling algorithms. It comprises all local temporal information about an agent, segregating the broader inter-agent temporal dynamics from the individual agent temporal dynamics, thus aiding in conflict detection and resolution.

- **Task Network:** The Task Network encapsulates the broader inter-agent temporal relationships, providing a comprehensive view of the multi-agent mission's temporal landscape. It includes external contingent and requirement constraints that govern the interactions between agents, along with a mapping of shared events to their respective owners. These shared events, along with their corresponding agents, highlight interdependencies and constraints between agents. The Task Network also includes a reference event with only outgoing edges and no incoming edges, serving as a standardized point of reference for temporal relations. By encompassing requirement and contingent external constraints and shared events, the Task Network captures the global temporal dynamics of the scenario, making it essential for understanding the overall structure and flow of the multi-agent system.

Additionally, all networks require a reference event, identified as the event with only outgoing edges and no incoming edges. This event serves as the starting point of the mission.

The process of populating these networks would be straightforward if the state plan consisted solely of the reference event and the endpoints of episodes. In such a case, each endpoint would be mapped to its corresponding agent, and the reference event would be added to all networks. External constraints would be determined by checking if the endpoints are part of the same agent network.

However, the state plan also includes auxiliary events, introduced to maintain the hierarchy of the control program, as we have previously discussed. These events cannot be directly mapped to an agent, complicating the process.

To handle auxiliary events, they are included in a separate dummy agent network, treated as if managed by another agent. This approach allows the temporal decoupling

algorithm to find a decoupling constraint for auxiliary events in the same manner as for regular agent events. It ensures the maintenance of the control program's hierarchy and simplifies the handling of these additional events, integrating them seamlessly into the existing framework. Therefore, we treat these events as events being executed by a dummy agent.

6.3 From Decoupling Constraints to Independent State Plans

After obtaining the decoupling constraints for each agent through the decoupling algorithm, the next step is to integrate these constraints with the original state plan. This integration forms the independent state plans for each agent, including the dummy agent, ensuring that the individual plans are in harmony with the overall mission goals.

The process of adding decoupling constraints involves iterating through the decoupling constraints and creating instances of either simple or contingent temporal constraints based on the type and bounds of the constraint. The result is a mapping of agents to their corresponding networks, now enriched with the decoupling constraints.

The transformation of decoupled networks into individual state plans includes the construction of decoupled state plans for each agent, including the dummy agent. The process involves the following key steps:

- Identifying the goal episodes for each agent.
- Converting the networks into state plans by combining them with the information of the original state plan, namely its state space and constraints that are not part of an episode.
- Storing these assembled decoupled state plans in a mapping, associating each agent with its corresponding state plan.

The result is a set of independent state plans for each agent. These state plans retain the essential temporal relationships and constraints of the original state plan while being tailored to the individual dynamics and requirements of each agent. This transformation is pivotal in enabling the Kirk executive to craft individualized plans that synchronize with the overarching mission goals, facilitating a unified yet flexible execution.

Chapter 7

Demonstration and Future Work

7.1 Demonstration

The Kirk executive is now able to handle complex multi-agent scenarios distributively. We will show interesting examples of the pipeline and discuss their relevant details.

Simple Parallel Hierarchy

Let's start by showing the distributed behavior of Kirk when it receives the following simple problem definition from RMPL:

```
(define-control-program task-A (agent)
  (declare (primitive agent)
           (duration (simple :lower-bound 30 :upper-bound 60) :contingent t)))

(define-control-program task-B (agent)
  (declare (primitive agent)
             (duration (simple :lower-bound 45 :upper-bound 45))))

(define-control-program main (rover1 rover2)
  (with-temporal-constraint (simple-temporal :lower-bound 70 :upper-bound 90)
    (parallel (:slack t)
      (task-A rover1)
      (task-B rover2))))
```

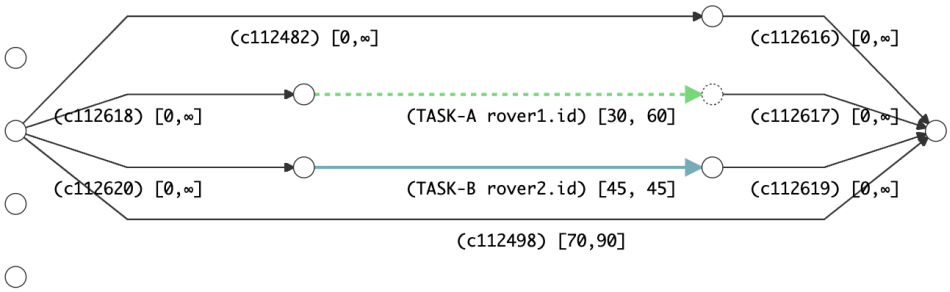

Figure 7-1: State plan of simple parallel task execution by two agents

After producing the corresponding simplified multi-agent state plan which can be visualized in figure 7-1, Kirk decomposes it into the task network and a set of three agent networks that includes the two agents of the problem and a dummy agent to input it into the temporal decoupling algorithm. The algorithm then outputs decoupling constraints appropriately, specifically constraint `c112868` and constraint `c112863` for rover1 and rover2 respectively. Once Kirk obtains the decoupling constraints, it assembles the resulting decoupled state plans for the two original agents as seen in figure 7-2.

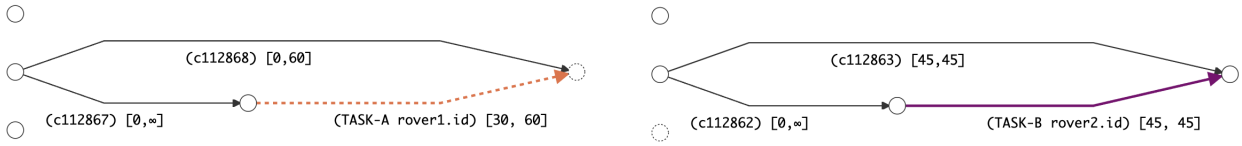

Figure 7-2: Decoupled state plan for rover1 and rover2 in simple parallel execution example

We have simulated the simultaneous execution of both agents' plans, collecting the resulting execution times in tables 7.1 and 7.2. Because the end time point of task-A is a contingent event, we have opted for simulating the observation of this event at the latest possible time. We will repeat this choice for all upcoming contingent events.

Event	Time (s)
EV112374	0
TASK-A:START	0
TASK-A:END	60

Table 7.1: Execution Times for rover1 in simple parallel execution example

Event	Time (s)
EV112374	0
TASK-B:START	0
TASK-B:END	45

Table 7.2: Execution Times for rover2 in simple parallel execution example

Parallel Sequences with Arbitrary External Contingent Constraint

A more interesting example uses a complex problem definition that includes an arbitrary internal requirement constraint and an external contingent constraint. The RMPL specification of this problem is as follows, where tasks A and B are defined as before:

```
(define-control-program main (rover1 rover2)
    (let ((A1 (task-A rover1))
          (A2 (task-A rover2))
          (B1 (task-B rover1))
          (B2 (task-B rover2)))
        (with-temporal-constraint
            (simple-temporal :lower-bound 1 :upper-bound 2
                             :from (start-of A1) :to (start-of A2)
                             :contingent t)
            (with-temporal-constraint
                (simple-temporal :lower-bound 3 :upper-bound 5
                                 :from (end-of A1) :to (start-of B1))
                (with-temporal-constraint
                    (simple-temporal :lower-bound 0 :upper-bound 115)
                    (parallel (:slack t)
                        (sequence (:slack t) A1 B1)
                        (sequence (:slack t) A2 B2)))))))
```

We are able to correctly compile its corresponding multi-agent state plan as seen in figure 7-3.

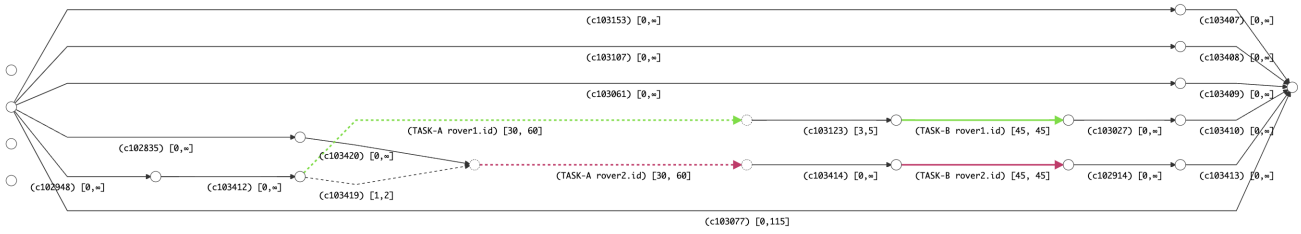

Figure 7-3: State plan of parallel sequences example.

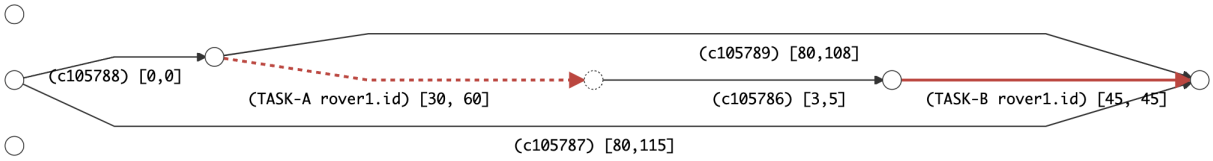

Figure 7-4: Decoupled state plan of rover1 in parallel sequences example.

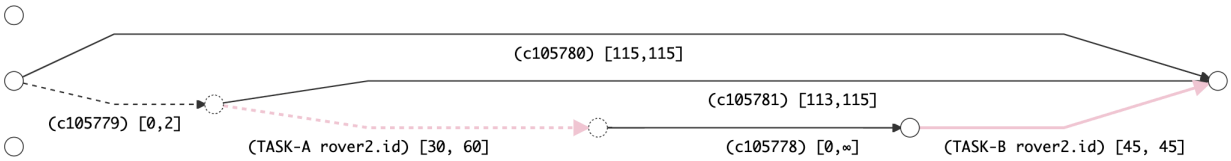

Figure 7-5: Decoupled state plan of rover2 in parallel sequences example.

When executed, the decoupled state plans seen in figures 7-4 and 7-5 produce the execution times for each event found in tables 7.3 and 7.4

Event	Time (s)
EV102813	0
TASK-A:START	0
TASK-A:END	60
TASK-B:START	63
TASK-B:END	108

Table 7.3: Execution Times for rover1

Event	Time (s)
EV102813	0
TASK-A:START	2
TASK-A:END	62
TASK-B:START	70
TASK-B:END	115

Table 7.4: Execution Times for rover2

These execution times maintain the overall constraints of the mission even though they are executed independently.

Zhang's example of MaSTNU

Finally, we can discuss the example we described in chapter 5, and shown in figure 5-5. Since we do not count with an implementation of the observation model, we model it using the addition of an arbitrary external contingent constraint as follows:

```
(define-control-program task-A (agent)
  (declare (primitive agent)
           (duration (simple :lower-bound 20 :upper-bound 40) :contingent t)))

(define-control-program task-A-prime (agent)
  (declare (primitive agent)
           (duration (simple :lower-bound 0 :upper-bound 0))))

(define-control-program task-BC (agent)
  (declare (primitive agent)
           (duration (simple :lower-bound 15 :upper-bound 15))))

(define-control-program main (alice bob)
  (let ((A (task-A alice))
        (A-prime (task-A-prime bob))
        (BC (task-BC bob)))
    (with-temporal-constraint
        (simple-temporal :lower-bound 30 :upper-bound 45
                         :from (end-of A) :to (end-of BC))
      (with-temporal-constraint
        (simple-temporal :lower-bound 0 :upper-bound 5
                         :from (end-of A) :to (start-of A-prime) :contingent t)
        (with-temporal-constraint
            (simple-temporal :lower-bound 60 :upper-bound 75)
          (parallel (:slack t)
            (sequence (:slack t)
              A A-prime)
            BC))))))
```

The multi-agent state plan compilation of this RMPL program can be seen represented in figure 7-6. The decoupled state plans are given below in figures 7-7 and 7-8.

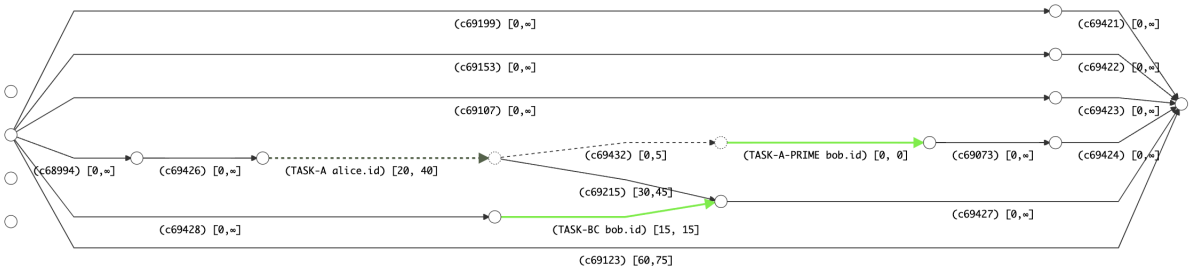

Figure 7-6: Multi-agent state plan representation of example MaSTNU [12]

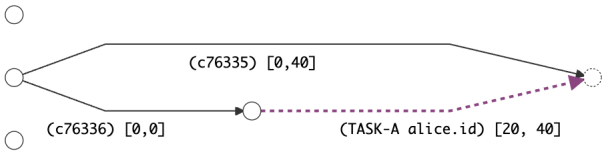

Figure 7-7: Decoupled state plan of agent Alice in example MaSTNU [12]

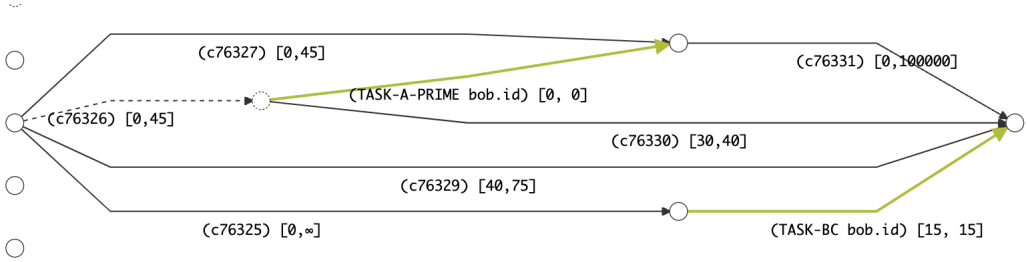

Figure 7-8: Decoupled state plan of agent Bob in example MaSTNU [12]

This results in the execution times found in tables 7.5 and 7.6.

The execution shows that even allocating all the time necessary for the contingent constraints, we are able to execute both plans that follow the original constraints of the multi-agent state plan. Here, this is possible because of the communication link that allows Bob to observe event A which is modeled as the episode A-prime. However, if we were to encode the same problem without the communication link, the state plan would look as seen in figure 7-9, and the decoupling algorithm would not be able to find appropriate decoupling constraints, which showcases the flexibility that we achieve when we model available communication.

Event	Time (s)
EV68925	0
TASK-A:START	0
TASK-A:END	40

Table 7.5: Execution Times for Alice

Event	Time (s)
EV68925	0
TASK-A-PRIME:START	45
TASK-A-PRIME:END	45
TASK-BC:START	60
TASK-BC:END	75

Table 7.6: Execution Times for Bob

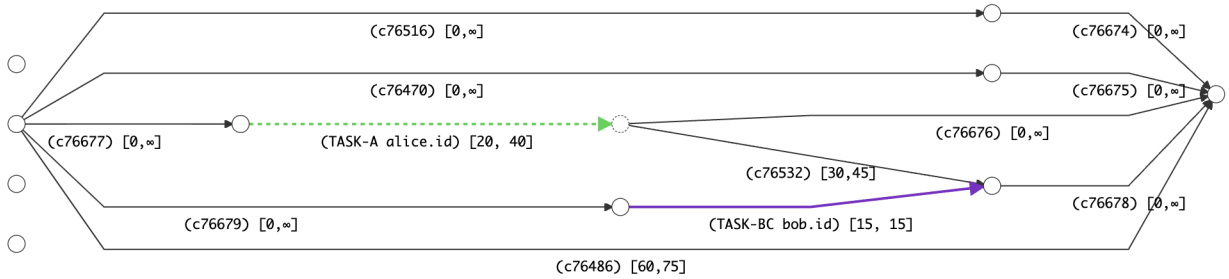

Figure 7-9: Multi-agent state plan resulting from MaSTNU without communication link [12]

7.2 Future Work

Although our executive portrays the desired behavior, there are still many aspects to improve. For instance, the implementation of the observable construct, as described in chapter 5, will facilitate modeling for the operator, as they will be able to easily capture observation opportunities within the mission. Now modeled as an external contingent constraint in RMPL, we saw from the last demonstrated example how these observation capabilities could offer the necessary flexibility for a decoupling to be feasible.

Furthermore, the Kirk executive could be improved by implementing a task assignment algorithm. In our RMPL programs, agent and task association are predetermined, which wouldn't be feasible for modeling problems with large and heterogenous swarms of agents. Thus, we could allow for program specifications that leave the task assignment up to the executive, where, ideally, Kirk would take into consideration the specific class and therefore capabilities or qualities of the agents to assign tasks such that the number of external constraints is minimized and less decoupling is necessary. We believe an addition like this will greatly enhanced the capabilities of the Kirk executive.

Moreover, this decoupling implementation should be extendable to include the decou-

pling of state variables in addition to the temporal constraints. The theory behind it was presented in [11]. Such implementation would significantly improve the ability of Kirk to handle realistic situations since there is often coupling between temporal and state constraints. For instance, in underwater missions, vehicles usually communicate at the surface, so in order to model a communication link we would need to take into consideration the current positions of the vehicles.

Chapter 8

Conclusion

This thesis has navigated through the realms of temporal networks, and multi-agent systems, to accomplish the integration of temporal decoupling into the Kirk executive. Next, we recapitulate the main insights, offering a concise summary of our work and a thoughtful contemplation of what lies ahead.

Our first two chapters aimed to set the context for the reader, highlighting the importance of our research and offering thorough references to the foundations we build upon. The significance of our work emerges from the alignment of theory and practice in integrating temporal decoupling into the Kirk executive. This integration expands the Kirk executive's capabilities to handle multi-agent scenarios in a distributed fashion, a critical approach for coordinating multiple agents in conditions of unreliable communication.

In addition to this foundational work, we conducted a thorough examination of related topics, including hierarchical distributed coordination, delay controllability, temporal decoupling, and privacy-preserving temporal decoupling. This review not only informed our research but also emphasized the unique contributions of our thesis, further underscoring the importance of distributed approaches in the field.

Then, we presented an overview of the architecture of the distributed Kirk, which uses a coordinator to process the input multi-agent state plan, decouple it, and assemble the resulting independent state plans. Since these share a common reference event as the starting point in their plans, we can simultaneously and independently execute them. In addition, we showcased how Kirk tackled specific examples.

The research presented in this thesis has not only contributed to the existing body of

knowledge but has also opened new avenues for exploration. The integration of temporal decoupling into the Kirk executive represents a significant advancement in the field, with potential applications in various domains.

The future may see further refinement of the methodologies and algorithms presented, along with the exploration of new challenges and opportunities in multi-agent systems and temporal planning.

Bibliography

[1] Nikhil Nikhil Gaurev Bhargava. *Multi-agent coordination under limited communication*. PhD thesis, Massachusetts Institute of Technology, 2020.

[2] G. Casanova, C. Pralet, C. Lesire, and T. Vidal. Solving dynamic controllability problem of multi-agent plans with uncertainty using mixed integer linear programming. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, pages 930–938. IOS Press, 2016.

[3] Luke Hunsberger. Algorithms for a temporal decoupling problem in multi-agent planning. In *AAAI/IAAI 2002*, pages 468–475, 2002.

[4] Michel Ingham, Robert Ragno, and Brian C. Williams. A reactive model-based programming language for robotic space explorers. In *Proceedings of ISAIRAS-01*, 2001.

[5] Phil Kim, Brian C. Williams, and Mark Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *IJCAI*, 2001.

[6] Thomas Léauté and Brian C. Williams. Coordinating agile systems through the model-based execution of temporal plans. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20. AAAI Press; MIT Press, 2005.

[7] Paul Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. 2001.

[8] John L. Stedl. Managing temporal uncertainty under limited communication: a formal model of tight and loose team coordination. Master's thesis, Massachusetts Institute of Technology, Dept. of Aeronautics and Astronautics, Massachusetts Institute of Technology, 2004. Thesis (S.M.).

[9] Thierry Vidal. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of experimental theoretical artificial intelligence*, 11(1):23–45, 1999.

[10] Brian C. Williams et al. Model-based reactive programming of cooperative vehicles for mars exploration. In *Int. Symp. Artif. Intell., Robotics, Automation Space (ISAIRAS-01), Montreal, QB, Canada*, volume 2001, 2001.

[11] Yuening Zhang, Jingkai Chen, Eric Timmons, Marlyse Reeves, and Brian Williams. State-temporal decoupling of multi-agent plans under limited communication. In *Proceedings of the AAAI Conference*, 2021.

[12] Yuening Zhang and Brian Williams. Privacy-Preserving Algorithm for Decoupling of Multi-Agent Plans with Uncertainty. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 426–434, 2021.