Assuring Agent Interaction through Run-time Monitoring and Control

Nora Ahmed Aldahash

Doctor of Philosophy

University of York
Computer Science

July 2023

Abstract

A multiagent system (MAS) is composed of autonomous agents that interact and exist in a shared environment. A fundamental aspect of multiagent systems is the communication that drives collaboration and cooperation among agents. Verification of agent communication is key to predictable and efficient interaction. Verification techniques of agent interaction have mostly been applied during design time. However, a multiagent system environment is inherently complex and presents challenges such as the heterogeneity of agents and the dynamic nature of the environment. To address such challenges, run-time approaches are needed to complement design time verification. This thesis presents a run-time monitoring and control approach for improving agent communication, where the environment takes a supervisory role through a Governing Agent (GA). The role of the governing agent is to minimise the negative effects of issues in interaction through monitoring and control. Run-time monitoring of interaction of agents is modeled with an Interaction Petri net (IPN). The proposed Petri net model allows the detection of common undesired scenarios such as protocol delay, lost messages, busy-wait, transmission delay, and agent termination. When scenarios are detected, control actions are taken by the governing agent. The proposed approach is evaluated with an experimental analysis and has been shown to minimise the negative effect of undesired interaction scenarios. The GA successfully detects issues in communication and apply control actions of interaction in two different classes of case studies. The first case study is a multiagent treasure hunt with collaborative interaction, while the second presents a competitive interaction demonstrated by an auction. Furthermore, a comparative analysis with a prior Petri net model of interaction is carried out to highlight the strengths and limitations of the new IPN model.

Contents

List of Tables

List of Figures

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Professor Steve King, for his invaluable support, guidance, and encouragement throughout the course of this doctoral research. His constructive feedback and assistance have been instrumental in the completion of this thesis.

I am also thankful to my examiners, Dr. Rob Alexander and Professor Brian Logan, whose insights and suggestions have contributed to strengthening this work.

I would like to thank Dr. Abir Najjar for her thoughtful observations and continuous encouragement throughout this journey. I also thank Dr. Daniel Kudenko for his guidance and support during the first year of my research.

I am endlessly grateful to my family. To my mother, Alanoud, your strength has always inspired me. To my sisters, Fahdah and Munira, I am truly grateful for your presence and for being there in all the ways that truly mattered. To my brothers Thamir, Fahad and Salman, thank you for your constant support. Salman, your advice has meant more than words can express.

To my partners in this journey, my husband Marwan, whose unwavering support and understanding have carried me through, my daughters Yara and Rund, and my sons Abdulrahman and Salman, thank you all for being your wonderful selves. This thesis would not have been possible without you.

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

Chapter 1

Introduction

Multiagent systems (MAS) are composed of multiple interacting intelligent agents that interact and make decisions to achieve their goals. Communication among agents is a fundamental aspect that drives coordination and cooperation. Currently, MAS have become more prevalent and there is a notable increase in systems that involve multiple agents interacting and coordinating to accomplish tasks, such as disaster area rescue, robotics and e-commerce.

The risks of an intelligent system are exposed by multiple challenges imposed by the autonomy of agents and their ability to make decisions independently. Therefore, there is uncertainty in agent behaviour that poses robustness and predictability issues. These challenges are further pronounced and more complex in the instances of multiagent coordination and cooperation.

Verification of agent communication has always been an important topic in the field of multiagent systems. More specifically, there has been interest in issues relating to faults in interaction that can effect the reliability of MAS [13]. Although numerous studies have been undertaken to investigate faults in agent communication, there is a need to further analyse runtime verification of the interactions of multi-agent systems [8]

A goal of verifying agent interaction is to ensure complete and correct

protocol enactment. A protocol is a sequence of interaction between different roles that agents can play with expected end points. A Successful protocol is when the messages exchanged follow the defined flow and reach an end point. However, complete interaction faces challenges such as dynamic open environments, asynchronous communication, and network delays.

A possible approach to resolving problems arising from agent interaction is through supervisory control for ensuring correct interaction. A supervisor role usually involves verification of requirements during run-time. This project aims to build upon this approach through a governing agent over a multi-agent system that is able to intervene in problematic situations. Another focus of this work is generating a model of multi-agent interaction using Petri nets for the governing agent.

This work proposes a model of multi-agent interaction using Petri nets to enable supervision of agent communication. Generating a model that captures multiple instances of a protocol but also models different protocols is a key objective for supervision. This objective helps in obtaining a holistic view on agent interactions in a multiagent system as agents are expected in multiagent environments to engage in different protocols and conversations simultaneously.

1.1 Assuring Agent Interaction

Run-time verification deals with techniques that allow checking whether an execution of a system satisfies or violates a given correctness property, as defined by [43]. An advantage here is that agents can be continuously verified while executing in their applied environment, which is key to fault containment and recovery. Existing run-time approaches to interaction verification vary in the properties they verify, and there has been work targeting communication languages [47], message structure [30], message sequence [34], and collaborative properties [44].

Faults in agent communication can have implications on the overall behaviour of agents and their goals. There are certain common scenarios in agent communication which effect multiple protocols simultaneously and consequently impact agents in completing their tasks. One approach to improve protocol enactment during runtime is to anticipate undesired scenarios that would cause faults of interaction.

A goal of this thesis is to detect undesired scenarios of interaction that happen in a MAS environment. To achieve this, the complete status of concurrent interaction should be monitored. This work builds on previous research on Petri net models of communication, and proposes the Interaction Petri net (IPN) for online monitoring to further explore Petri nets' efficiency during run-time [54] [57].

Previous Petri net models of interaction have focused on protocol-specific models which provide partial monitoring of communication [20] [35] [54]. In contrast, the proposed IPN defines a protocol as a Petri net marking to handle the monitoring of simultaneous conversations and protocols. In addition, marking analysis facilitates the detecting of undesired scenarios in interaction and gives access to message content being exchanged.

Monitoring and control hold potential if corrective actions can be taken. For this reason, we consider a governing agent equipped with a Petri net model of interaction to facilitate the process of detection of undesired scenarios. An advantage to online detection is that it presents a window to act if needed. A description of the proposed framework is presented in the next section.

1.2 Contributions

This thesis introduces an agent interaction monitoring and control framework for assured agent interactions. It comprises two components: the IPN model for capturing the state of interactions and the governing agent responsible

for detection and control. Figure 1.1 depicts the framework at a high-level.

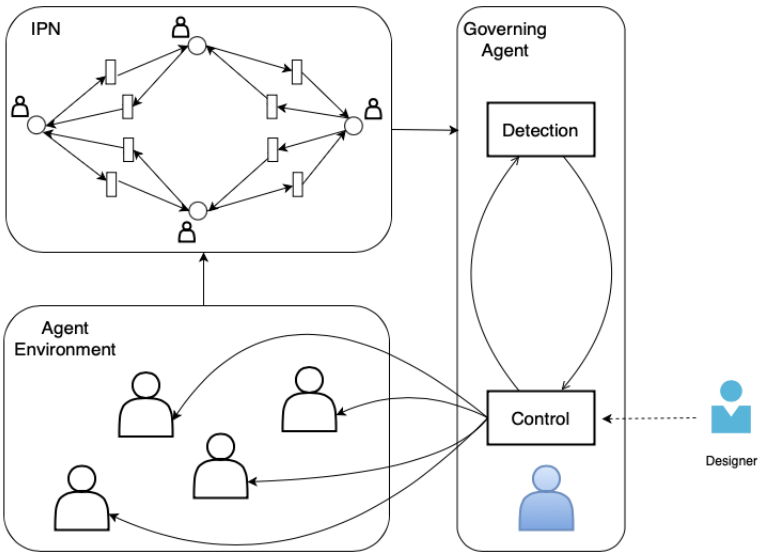

Figure 1.1: Monitoring and control cycle: 1. Detection takes the IPN marking as input. 2. Control action is based on detection output.

The proposed IPN model addresses the need to capture multiple interaction protocols at the same time. Previous models of interaction have modeled a single interaction protocol, while with the IPN, different interaction protocols can be modeled. The IPN can be considered agent-based, where places represents agents and tokens within a place represent messages. This approach enables the analysis of multiple conversations of different protocols an agent is engaged with in a single Petri net.

An agent-based approach to modelling protocols focuses on the status of interaction of agents rather than the protocol. An advantage is capturing an agent's observed and emitted messages. A Petri net marking would then represent an agent's message history and the protocols it is engaged in. The marking is the input to the governing agent analysis of interaction which not only provides protocol-specific status but any protocol within the

environment.

Additionally, common issues of communication are defined within the IPN for detection. Defining general issues enhances usability in which the issues are not dependent on context-specific protocols. The specifications of five undesired scenarios are introduced: Protocol Delay, Lost messages, Busy-wait, Transmission Delay, and Agent termination. Protocol delay refers to when an interaction protocol (IP) takes more than expected to be complete, lost messages are detected when an agent sends a message but it is not received by the recipient, busy-wait occurs when an agent receives a request while handling an ongoing request, transmission delay is when the delivery of a message is delayed, and termination of agents participating in ongoing protocols.

The second component of the framework described in this thesis, is the Governing Agent (GA), which handles detection and control. First, the agent performs a detection process in which it analyses the marking of the IPN and placement of tokens guided by the specification of interaction scenarios. When an undesired scenario is detected, the agent takes a control action. A control action is defined to correct and contain the negative effect of a scenario on communication. The control action determines how the environment and agents react when an issue of interaction is detected. Moreover, the governing entity is expected to be a part of the system design but can be imposed on an existing MAS. In that case, the design of the governing agent should be provided by the system designer as it requires knowledge of the environment.

The proposed approach is a supervisor entity that has the advantage to oversee the complete status of communication in MAS. The GA observes the local status of communication with IPN and reacts with governing action in the case of an unexpected or an undesired issue happening during runtime. The framework can be applied in a distributed MAS with replicated pairs of governing agents and IPNs. This allows the task of distributed monitoring to be carried by each GA placed within nodes of the MAS, but governing

actions of a GA are constrained to agents that operate within the same GA node. Complete status of interaction is available to GAs through a shared marking of IPN.

The contributions are summarised below:

1. The IPN model of interaction that enables monitoring of agent conversations during run-time. The model captures simultaneous conversations that can belong to different interaction protocols. Furthermore, the IPN model supports open environments where new agents can be added in run-time.

2. The formal definition of common undesired scenarios of communication. A specification of five types of interaction issues is given based on the definition of IPN.

3. A governing agent responsible for detection and control. It implements a process for detection within the IPN model and carries out appropriate control actions. It is applied in two MAS case studies to demonstrate how the governing agent successfully detects undesired interaction scenarios and minimises their effect on agents.

4. A comparative analysis of the IPN model with a similar Petri net model for validation.

5. An extension to the governing agent to be applied in a distributed environment. This includes a marking update process that entails the sharing of markings between governing agents.

1.3 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2 presents a background on multiagent systems, agent communication, and Petri net models. The chapter includes a review on the

interaction verification techniques applied over the software life cycle. It also presents a review on previous Petri net models of interaction and their properties and limitations.

Chapter 3 introduces the monitoring and control framework. First, a detailed specification of the monitoring and detection process is given, showing how the IPN captures the state of conversations and how undesired scenarios in communication are defined within the model. Next, the control process is described, showing the behaviour of the governing agent.

Chapter 4 evaluates the framework in an experimental analysis. The experiment involves two case studies. The first is a multi-agent case study with heterogeneous agents collaborating in a dynamic environment. The second case study is an auction with multiple participants. A comparison of IPN performance and capabilities against a related model of interaction is also applied to the case studies.

Chapter 5 details how an IPN can be applied in a distributed case study with multiple governing agents. The approach is used for maintaining the IPN across multiple instances of GA's. The approach is evaluated through two case studies, and its capabilities and limitations are discussed.

Chapter 6 summarises the contributions of this thesis, the limitation of the introduced run-time approach, and potential areas for future work.

Chapter 2

Literature Review

2.1 Introduction

Today with the rise of the robotics fields, the Internet of Things, and autonomous driving, MAS systems are more pervasive than ever and involve multiple agents that need to cooperate or coordinate to achieve a certain objective.

There are risks that arise from coordination mistakes. Safety concerns for autonomous agents operating in a multi-agent environment not only arise from the design of individual agents but also reflect the dynamics of interaction between agents.

A MAS of autonomous agents may exhibit different modes of coordination that need to be evaluated [13]. In certain scenarios, autonomous agents learn to coordinate, while in other cases, they are designed to communicate and coordinate efforts. In either case, unpredictable behaviour can emerge from multi-agent coordination that affects overall system execution. Moreover, a multi-agent environment is dynamic, as each agent takes an action towards its goal, thus altering its environment and presenting a challenge to safe coordination [48].

Run-time verification is considered an important approach for agent in-

teraction assurance, particularly in MAS, where an environment is typically open, dynamic, with asynchronous communication. However, run-time approaches are not as widely explored as design time approaches [8]. In this chapter, we present an investigation of previous work in the area of verifying agent interaction in collaborative settings.

The remainder of this chapter is organised as follows: Section 2.2 presents the background on agent communication and the characteristics of previous approaches to verifying agent interaction. Section 2.3 presents a review on MAS interaction verification techniques using Petri nets. Section 2.4 is the chapter summary.

2.2 Background and Related Work

This section introduces and defines the main concepts considered in this thesis. Section 2.2.1 gives an overview on agent communication and different specification approaches of interaction protocols. Section 2.2.2 presents related work on interaction verification, and section 2.2.3 is the discussion.

2.2.1 Agent Communication

An intelligent agent is the building block of multiagent systems. According to [74], an agent is:

> A computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to achieve its delegated objectives.

Autonomy of agents refers to their ability to make decisions on their own, meaning they rely on their internal state and behaviour to act. Moreover, in a multiagent system, agents are expected to have social ability, that is, they are also capable of interacting with other agents.

Table 2.1: Sample of performatives provided by KQML

Performative	Meaning
tell	S claims to R that C is in $S's$ VKB
forward	S wants R to forward a message to another agent

Interaction among agents is a core aspect of multiagent systems and facilitates coordination and negotiation among agents required in different types of systems. An agent interacts with other agents for different reasons depending on the environment, the roles of those agents, and the goals they were built to achieve. Voting, negotiating, and cooperation are among the shapes of communication that exist.

Communication between agents has been directly influenced by the speech acts theory. The theory of speech acts originates from the work of Austin [6] and was later extended by Searle [10]. In this theory, communication is treated as an action. The utterance of words alters the state of the world as an action does. In their work, a number of performative verbs were identified, such as request and inform, that correspond to different types of speech acts.

Significant efforts were made to support standard forms of communication for heterogeneous agents' message exchange through the development of agent communication language (ACL). The knowledge query and manipulation language (KQML) [27] was introduced first, and defines a group of performatives, a sample is shown in Table 2.1. A performative is described using the notion of a Virtual Knowledge Base (VKB), which represents the agent's beliefs. A message in KQML is an envelope structure with a set of parameters as depicted in Table 2.3.

Later, the Foundation for Intelligent Physical Agents introduced the FIPA ACL, known as FIPA [28]. FIPA shares a similar message structure with KQML, shown in Table 2.3, and also defines a set of performatives that represent the intention of a message, shown in Table 2.2. Moreover, FIPA produced a set of standard interaction protocols that define a structure for

Table 2.2: Sample of performatives provided by FIPA CL

Performative	Meaning
inform	The sender wants the recipient to believe this content
accept-proposal	An agent wants to state that it accepts a proposal made by another agent

Table 2.3: Message structure of KQML and FIPA CL

KQML Message Structure	FIPA Message Structure
(performative	(performative
:content	:sender
:receiver	:receiver
:language	:content
:ontology	:language
)	:ontology
	)

message flow for a number of common interactions, such as the Contract Net protocol.

FIPA tackles the lack of syntax in KQML and provides more detailed specifications that improve interoperability between agents. The semantics of both languages are based on cognitive concepts of beliefs and intentions of an agent. FIPA provides a formal specification of performative semantics using a formalisation of beliefs and intentions.

FIPA and KQML are considered operational ACLs with mentalist semantics. In contrast, many researchers have called for the adoption of interactions that are based on social semantics, not the internal state of agents [14]. Social-based interactions hold a business meaning, and interaction between agents is carried out by creating commitments to one other.

In commitment-based protocols, a commitment can be represented by the

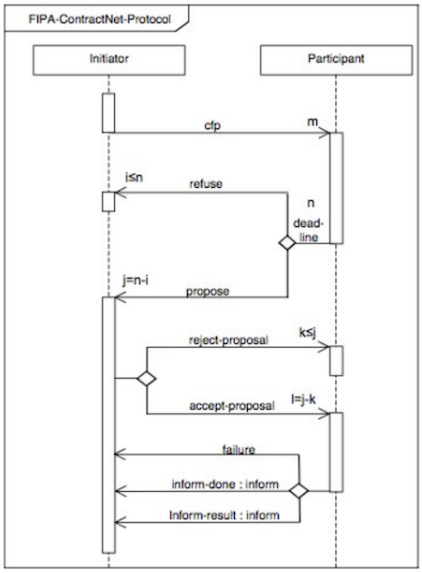

Figure 2.1: FIPA Contract Net Interaction protocol

expression C(debtor, creditor, antecedent, consequent), where the debtor is committed to the creditor to accomplish the consequent if the antecedent holds. Commitments are created and manipulated with operations like: create, cancel, and release. Abstraction at the business level enhances flexibility of operation for agents.

The type of interaction is shaped by the context and environment of agents. Interaction protocols are what facilitates communication among agents. There have been domain-specific protocols, for example, negotiating [59] and e-commerce [64]. There are also environments where the communication needed is simple message passing among agents.

An interaction protocol is a pattern of a conversation between agents [68]. Protocols define a set of message instances and a flow scenario between participants. Figure 2.1 shows the well known Contract-net protocol in AUML [50], which is one of the early notations to specify interaction protocols. The protocol defines two agent roles, initiator and participant, and the scenario of message exchange between participating agents.

Many approaches exist to specify protocols, and they vary in the abstractions they adopt. There are graphical notation such as state machines, Petri nets, and logic based approaches such as propositional dynamic logic, session types and trace-expressions. Generally, a protocol is specified through message structures and a control flow.

Petri nets and Finite State Machines (FSMs) provide a simple graphical notation to model interactions. Both are used to describe and analyze the behaviour of systems through states and state transitions. FSMs are considered single threaded where it can capture a single state, but Petri nets can model concurrency, as they are able to represent multiple states at the same time with tokens. Multiple interactions are allowed to happen at the same time and for FSMs to handle concurrency, additional techniques are needed.

HAPN [72] is a hierarchical FSM notation that implements hierarchy to model multiple protocols. A state represents the points in the flow of a protocol whereas a transition defines the sender, the receiver, guards that can test a condition and the effects on values of variables. A state can contain a number of sub-protocols, and can not transition unless all sub-protocols have reached their final state.

Among logic based approaches, session types is a formalism used to describe a structure of interactions between two parties. Scribble [76] is a protocol language based on Multiparty Session Types (MPST) that represents a conversation between two or more agents as a session. A conversation follows a protocol which is defined as a global type that involves agent roles and message signatures. Then a global type is projected into local protocols for each role.

Trace expressions are a formalism devised for the dynamic verification of agent interaction. First introduced in [3] as an evolution of global types, trace expressions define a set of operators to denote a trace of events. In [26], a protocol specification language is implemented based on trace expressions with protocol enactment checking. Enactability checking or protocol realization

is done with consideration of two aspects of interaction: the communication model and message ordering.

A communication model refers to the communication structure supported by the MAS, which can be synchronous, asynchronous, FIFO and others. In synchronous communication a receiver is required to be ready to receive before a sender can send out a message, asynchronous communication is when there is no order imposed on message delivery. In FIFO n-n communication, order on message delivery is applied where messages are delivered according to their emission order.

The second aspect is message ordering within a protocol and deals with the interpretation of the order of messages from the projected local protocol. The identified interpretations based on the work in [23] are SS, SR, RS and RR. Take the protocol: $a \xrightarrow{M1} b, c \xrightarrow{M2} d$. In the send before send (SS) a must send M1 before c sends M2. In the send before receive (SR), a must send M1 before d receives M2. In the receive before send (RS), b must receive M1 before c sends M2. In the receive before receive (RR), b must receive M1 before d receives M2.

Another approach that is based on global types and trace-based semantics is presented in [31]. An algorithm is introduced for projecting the global type into a set of sessions of its participants. For each session a trace is defined as the set of sequences of interactions that can occur.

There is also the Blindingly Simple Protocol Language BSPL [65], a declarative language that specifies a protocol in terms of the flow of information rather than the order of messages. BSPL emphasizes the history of messages, that is the information observed by an agent which enables it to proceed in a protocol.

The aforementioned languages represent different approaches for the design and specification of communication. In a comparative study found in [15], criteria for evaluation are presented to help distinguish the protocol languages. The criteria include concurrency, extensibility, protocol instances,

integrity and social meaning.

Concurrency refers to when messages may be sent or received concurrently, which reflects flexibility in protocol specification. Extensibility of a language is when agents can participate in different protocols. Instances of a protocol is a criterion for how a language can model and manage multiple instances of a protocol. Integrity and social meaning relate to the information within a message, information belonging to a specific protocol instance must hold for every message. Social meaning specifies the life-cycle of a commitment protocol and requires language support for protocol instances and data integrity.

Based on their evaluation criteria, the BSPL outperformed other languages. A main factor is that BSPL does not require message ordering guarantees while Scribble and trace expressions assumes FIFO communication in which message delivery is ordered. The HAPN language falls short in the representation of concurrency and extensibility because of state machines synchronization.

In another evaluation study that includes BSPL, HAPN and trace expressions [4], a different set of criteria is examined. The selected criteria relate to the support of fault tolerance and prevention such as static and dynamic verification. The criteria includes: modeling approach, IDE support, code generation, testing/simulation, a-priori verification and run time verification. BSPL falls behind HAPN and Trace expression in not having IDE support.

A communication modelling approach is important for specifying correct and complete interaction models. It facilitates the creation and design of how agents interact and ensures agent compliance of protocols. A complete enactment of a protocol is considered the criterion for success, and any deviation from it is seen as a failed interaction. Factors such as communication latency effect the successful completion of a protocol and reaching the end point for each role.

Environments and Agent Interaction

A multi agent system is defined in [24] as the set of autonomous agents that perceive its surrounding and capable of interacting. The space in which these agents sense and act on is the environment. It is agreed that an environment is a primary entity in the design and engineering of a MAS [71] [61]. The structure of an environment can be different from simply a facilitator of communication to a more complex and broader role.

When selecting a MAS environment for evaluation, a first consideration is that it would hold properties of MAS such as scalability and openness [70]. For agent interaction design and verification, the interaction model would then vary based on the applied environment from having complex protocols to simple message passing. In certain environments, indirect communication exists where agents interact through modifications to their shared environment.

Among the MAS environments is Packet-World, a test-bed for investigating situated agents that contains indirect and direct agent communication [69]. A situated MAS is when agents are explicitly placed in that environment. The grid-environment involves a number of coloured packets that agents need to move to their corresponding coloured tiles. Agents can communicate with each other to collaborate through a collaboration protocol. Indirect communication is applied by means of environmental markers such as flags, where agents can pass information indirectly. The Packet-world environment can be seen as related to unmanned vehicle transportation and automated warehouse transportation system.

Another test-bed is the Dedale environment which aims to study decentralised multi-agents coordination and decision-making [37]. The environment portrays a treasure hunt for teams of agents in an open and dynamic environment. The creators of Dedale present a comparison between several case studies found in the literature based on a number of MAS properties including asynchronous agent action, heterogeneity, partially observable, team

size and asynchronous communication. Notably only two out of the eight reviewed platforms have asynchronous communication.

An important aspect to consider in the evaluation of agent interaction is the protocols specified. A range of interaction protocols have been applied in different case studies for the study of agent communication design such as the Net-Bill protocol [66], a Play date protocol, Auction, collaboration protocol in Holonic manufacturing [72], Seller buyer protocol [11]. Typically a communication protocol contain a number of elements of interaction like choice, loop and exception.

Additional protocol properties are examined in [72] [15]. For the evaluation of protocol notation in [72], protocols that were selected hold one of four interaction types: First is parallelism where parts of an interaction take part at the same time, and synchronization where parts of interaction should not interleave. Second is exceptions in a protocol in which a protocol can be aborted at certain points. Third is information driven interactions which specifies what information needs to be collected within a protocol. Lastly, interactions that involve multiple role instances such as multiple bidder roles in an auction. Some of the criteria used for the evaluation of protocol languages in [15], are somewhat similar. The Concurrency criteria addresses parallelism in interaction. The Integrity criteria entails the information driven criteria but holds a stronger definition on the consistency of values and parameter bindings.

When evaluating agent interaction, the environment should not only exhibit characteristics of MAS but should entail interaction scenarios of interaction that cover the main criteria presented. A first scenario is extensibility, agents within the environment should be able to participate in different protocols. Second is a scenario where multiple instances of a protocol exist, which is when more one conversation of a protocol take place simultaneously. Third is protocol integrity. Fourth is including a protocol that involves multiple roles such as the Contract Net protocol. Lastly, a protocol that has

exception points.

2.2.2 Verifying Agent Interaction

Verifying agent interaction is a complex task and there is no unified approach to ensuring correct interaction. Different verification approaches exist throughout a software life-cycle, from design to development to run-time. A key distinction in verification is whether to verify the correctness of a specified property or to detect a violation of it.

A specification of how agents interact is the initial step for verification and varies depending on the formal specification language or model. The next step is to check the correctness of the property through formal approaches, such as model checking and theorem proving.

In model checking, a model of the system is checked to determine whether it meets a specification, and in theorem proving, a mathematical approach is followed to prove the correctness of a programme specification. Such methods are mostly applied at design time.

Design-time verification One of the early works on automatic design time verification is the language proposed by [73] called Mable. The language is used for the design and verification of a MAS, where agent communication is specified using two primitives aligned with FIPA: request and inform. The language makes use of the SPIN model checker to verify claims about the system. Another SPIN-based approach is found in [40]. The authors presents a tool that transforms protocols modeled with state-charts to the formal language Promela. Through message sequences, they verify the absence of deadlock and non-progress loops.

There are also approaches that target a specific aspect of communication. Epistemic properties are verified in [7], while in [1], they focus on social semantics of interaction. Work in [7] present an extension to the Action Computation Tree Logic for epistemic reasoning. They specify an interac-

tion protocol based on information exchange and verify the epistemic properties of participating agents. The work in [1] relies on the Social Integrity Constraints, a logic-based formalism, to specify social properties of an interaction protocol. They verify desired properties of interacting agents through proof-procedure.

In [52], authors address the issue of the incompleteness and ambiguity of FIPA IP specification. They propose to extend the semantics of an IP through a combination of state charts and propositional dynamic logic to formally verify its termination. A more general approach is found in [47], where the verification process is done at the ACL level. They introduce an approach to translate KQML into the input language of the MCMAS [46] model checker.

Run-time verification The second group of approaches is applied during run-time. Such approaches aim to verify interactions during execution where unexpected agent behaviour may arise. Agents typically interact in a dynamic, open environment with asynchronous communication.

Run-time approaches usually target violations of a specific property or find faults through testing and debugging approaches. A monitoring mechanism is used for the debugging and testing of agent behaviour [43]. It provides run-time visibility, which helps developers and designers in understanding how faults occur. It essentially consists of collecting data from agents and making it available visually or through logs.

Debugging agent interaction is usually done through checking message sequences. A tracing tool is introduced in [41] that logs agent behaviour and goes through an interpretation process using agent concepts of beliefs. The process outputs the agent interactions modeled with state-charts, which are verified through comparison with expected message sequences. Similarly in [32], interactions are captured with state machines for testing and debugging within the INGENIAS development platform.

There have also been efforts to develop tools for test automation for agent interaction. JAT [19] is a test automation framework that defines a mock agent that interacts with agents and carries out different test scenarios. Interaction protocols are monitored where faults of incomplete and unexpected messages are detected.

The authors of [51] apply model checking during run-time, and they focus on social constraints and aim to ensure an interaction satisfies those constraints. A combination of the system interaction model with an agent internal deontic model is fed into a model checker during run-time to verify the correctness of interactions.

Another form of verification is carried through protocol design and specification languages. An objective of protocol languages, discussed in Section 2.2.1, is to facilitate the process of compliance or enactment checking.

Scribble, BSPL and Trace expressions follow a distributed approach in which interaction is verified locally from the agent perspective. Verification techniques vary between protocol languages based on the constructs they adopt. For verification with Scribble, an FSM is generated for each role in the protocol [22]. This is through projections of the global type protocol into multiparty session types for each participant role. Projections enable distributed monitoring where a local FSM captures the progress of a participant communication. They showed low overhead of monitoring verification with FSM generation.

The KIKO approach extends BSPL and aims to bridge the agent decision logic with interactions that are protocol-compliant [18]. Interaction is verified through the agent decision making process. The approach involves Decision makers and a protocol adapter. The agent programmer provides the agent Decision maker, plugged into the protocol adapter, which holds the business logic of an agent and decides what messages to send. As BSPL specifies protocols with information constraints, the protocol adapter deter-

mines which messages are viable based on the message history thus ensuring correct protocol enactment.

As an alternative to verifying specified properties, the work of [34] introduces a formalism to specify both desired and undesired patterns of communication. The formal language, called the Robust Communication Language, has simple notation to specify undesired sequences, such as deadlock and initiation, and desired sequences, which depend on the particular protocol specified. Detection of patterns is undertaken after execution through queries on logs of recorded messages. An advantage here is the specification of undesired patterns is not tied to specific protocols.

The authors of [16] have presented valuable work with the aim of achieving fault tolerance communication in decentralised MAS with asynchronous message exchange. Their approach adopts handling faults such as message loss at the application level. Capturing faults is based on the information flow of a protocol and what messages are expected, and handling a fault is carried by the agent through following policies. The policies guide the agent in how and when to transmit information in order to recover and continue with the protocol. Two types of policies are defined, one that is protocol-specific which is defined as an extension to a protocol, and the other can apply to any protocol such a resend policy. The authors later introduce Mandrake [17], which is an agent programming model designed to address application-level fault tolerance built on concepts of their earlier work in [16].

2.2.3 Discussion

The application of formal methods relies on mathematical design and proof for verification, which makes it a viable option for agent interaction assurance. However, applying formal methods involves many challenges [62], such as the complex task of modeling the large open environments in which MAS systems are typically applied.

There is also the problem of formally specifying properties of intelligent

agents given their autonomous nature and the fact that they are modelled to make decisions. Despite these challenges, formal specification and verification is an important step towards correct interaction within MAS.

Design time verification of communication provides assurance, which is desirable before system execution, but complex, stochastic, and dynamic MAS environments require a combination of design and run-time approaches. Run-time methods offer designers visibility and enhance explainability of failures in dynamic environemnts. The authors of [8] have found, in their review of verification approaches to interaction, that only 25% of verification is done during run-time.

Formal runtime verification is concerned with verifying interaction based on a specification of a protocol. A trace of interaction is checked against expected message order at run-time. For the most part, protocol verification approaches have not dealt with the correction of a detected violation. For MAS environments, run-time approaches equipped with corrective or fault containment behaviour is desirable.

It is important to consider approaches such as fault detection and management that address emergent risk from mistakes of interacting [16]. One form of fault management is to specify faults or incorrect interaction instead of the expected and correct one. This approach anticipates incorrect scenarios of execution that consequently produce interaction faults. This is similar to the approach found in [34] in which undesired interactions are identified for detection. Moreover, an advantage here is that it enables the developer to implement a corrective action in response to the detected scenario at run-time.

Work that targets fault tolerance of communication has been shown to handle faults at the agent level where an agent is equipped with corrective policies [16]. However, another direction is to handle faults or unexpected issues in communication at the environment level. Detecting protocol faults from an agent perspective is subjective and concerned with the protocol con-

versation an agent is engaged with, but an environment perspective presents an objective view of all ongoing protocols. An objective view enables analysis of any protocol incompleteness or issues and allows a corrective action to be taken on the system level.

An environment entity with an objective view of communication would respond to issues of interaction at the environment level. A common issue of communication is message loss, for example a reaction would not only forward the message but also ensure the recipient is still available. Another possible issue is the termination of agents which results in incomplete protocols; if detected at the environment level, a replicated agent can be set to continue the protocol, and it may also be possible for the cause of termination to be examined by a supervisor.

Creating an expressive model of agents' communications is key to their analysis. Such a model would allow the detection of interaction issues within multiple protocols of the environment. The run-time approaches above vary in their mechanisms for formally capturing interactions, such as state charts, Petri nets, trace expressions and propositional dynamic logic.

The protocol specification languages present different formalisms for monitoring protocols at run-time. However, there are concerns when the objective is to monitor multiple protocols. The verification process with specification languages is applied for the specified protocol. Languages that specify protocols based on the order of messages, such as trace expressions, suffer from limited flexibility as they require message order in their verification technique. BSPL is a declarative language concerned with specification of protocols and offers an agent-centric perspective to interaction verification. Petri nets are similar to state charts, however Petri nets have the capability of modeling concurrent conversations which is a limitation for state charts.

Overall, a verification approach that reacts to issues in communication would benefit from a holistic point of view. Monitoring the global state

of interaction provides insight into the concurrent interactions taking place. We consider Petri nets as a monitoring formalism that enables detection of communication faults not just based on the structure of a specific protocol but rather any protocol.

In the next section we will review the different Petri net models of MAS in more detail. As Petri nets are considered a popular method for modelling multi-agent interaction [20, 21, 54, 35, 57], we explore the different Petri net models proposed for MAS.

2.3 Petri Net Models of MAS

Petri nets were invented by Carl Petri in 1962 and have been successfully used to model concurrent, distributed systems. A multiagent system is considered a discrete-event dynamic system, also suitable to be modeled using Petri nets as shown in [12]. In addition to modelling concurrent processes, Petri nets have a strong mathematical foundation underlying the graphical notation, which provides for both simulation and verification analysis techniques.

A Petri net is a graphical model consisting of places represented by circles, and transitions represented by bars and arcs, which can be directed from places to transitions or from transitions to places. A place within a net can be marked with one or more tokens, and the distribution of tokens among places is called a marking of a net. The movement of tokens between places is caused by firing a transition.

Definition 2: *A Petri net is a five-tuple* $\langle P, T, A, W, M_0 \rangle$ *where* P *is finite set of places.* T *is a finite set of transitions.* $A \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs.* $W : A \mapsto \{1, 2, 3, ..\}$ *is a weight function.* $M_0 : P \mapsto \mathbb{Z}$ *is the initial marking.*

An example of a simple Petri net is shown in Figure 2.2, illustrating the chemical reaction [49]: here, places H2 and O2 are input places to transition

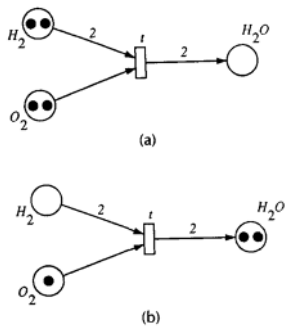

Figure 2.2: An illustration of a transition: (a) The marking before transition t (b) The marking after firing t. The number over an arc presents the weight of the arc. An empty arc indicates a weight of 1.

t, while place H2O is an output place of t. The initial marking in Figure 2.2 (a) shows two tokens in each input place. Transition t can fire only when it is enabled, that is, when all the input places of t contain at least the number of tokens equal to the weight of the arc connecting H2 and O2 to t. Figure 2.2 (b) shows the marking after transition t is fired, two tokens moves from the input place into the output place based on the arc weight from transition 2 to its output place H2O.

A marking of the Petri net defines the number and assignment of tokens in places; in other words, it represents a Petri net state. For example, Figure 2.3 (a) shows the initial marking of petri net N where M0 = [2,0,0]. An important concept is the reachability graph of markings where each marking is a node as shown in Figure 2.3 (b). Marking M0 = [2,0,0] is joined with M1 = [1,1,0] with an arc that represents the transition when firing t1 takes the Petri net from state M0 to M1. The behavioural properties of Petri nets can be applied to evaluate multi-agent systems, such as liveness and boundedness. The liveness of a Petri net refers to its ability to avoid deadlock. It indicates that a transition within the net is enabled and can be fired. In other words the net can progress and will not reach a state where no transition can be fired. A net is live when it is possible to ultimately fire any transition in the

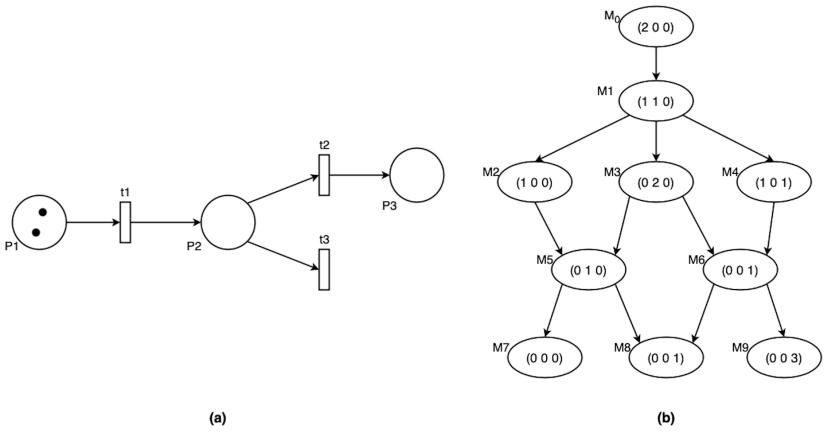

Figure 2.3: Petri net: (a) Initial marking (b) Reachability graph

net by progressing through some fire sequence for any marking reached from M0; thus, it guarantees a deadlock-free execution.

Looking back at Petri net N in Figure 2.3, we can see that is not live and after reaching marking M3, it is no longer possible to fire transition t1. Boundedness reflects the number of tokens a place can have; places often represent buffers for storing data, and verifying boundedness helps ensure there will be no overflows. A Petri net is said to be K-bounded if the number of tokens in each place does not exceed a finite number K for any marking reachable from M0. Moreover, a number of extended versions have been introduced that are referred to as high-level Petri nets, in contrast to original low-level Petri net. Examples of high-level nets include Timed Petri nets for defining time specification and Coloured Petri net (CPN), where to a data type can be attached to each token. Overall, the application of Petri nets to model the interaction of multi-agents could be useful in the analysis of safety properties of interaction. In particular, key properties of behaviour in multiagent systems, such as concurrency and conflict, can be modelled with Petri nets [67].

There has been numerous work on Petri net-based modelling of multi-agent systems for the purposes of design, verification, and validation. This indicates the ability of Petri nets to model multiagent systems and represent their characteristics as dynamic, concurrent systems. Previous Petri net models have been created for verification during two different phases of a system life-cycle: design time and run-time. In the analysis and design of MAS, Petri nets have been shown to naturally model agent architecture and produce feasible models of both agent architecture [12] and agent communication and interaction [20]. Petri net-based analysis techniques and simulations facilitate early detection of faults and assessment of liveness and dead-lock properties. Table 2.4 summarizes a collection of relevant Petri net work, highlighting the aspects of MAS verified and the analysis techniques used. In some work, a model is presented with no specification on how the model can analyzed.

2.3.1 Design Time Models

In the design phase, a developer's goal is to verify the design and identify problems early as possible. The reviewed work may be classified according to which aspects of a MAS are being verified. These main components are: an agent architecture, agent plans, and agent interactions.

The Petri net model of agent architecture in [12] presents places as the action a particular agent is performing. Having a token in a place that represents a specific action means that the agent is currently performing such action. Firing a transition indicates that the agent finished performing the action indicated by the input place. Through reachability graph analysis, the authors were able to prove that the system is deadlock free. In terms of agent interaction, the model only considers indirect interactions, where one agent changes the environment in a way that communicates new information to other agents. If communication were to be explicitly modeled, it would

Table 2.4: Summary of Petri net models of Multi-Agent Systems: 1. MAS aspect refers to target of verification. 2. Phase of the life cycle: Analysis and Design (A&D), testing, run-time. 3. Petri net type. 4. Petri net-based technique for verification. NA refers to not applied

ref	**MAS aspect**	**Phase**	**Petri net**	**Analysis Technique**
[12]	Agent Architecture	A&D	Low-level	Reachability analysis
[45]	Agent behaviour	A&D	Low-level	NA
[75]	Agent plans	A&D	Predicate/Transition nets	Reachability analysis
[2]	Agent plans	A&D	Hierarchical Coloured Petri Net	State space
[20]	Interaction protocols	A&D	CPN	NA
[63]	Coordination protocol	A&D	Stochastic PN	Simulation
[77]	Agent decision	A&D	Condition/event PN	NA
[9]	Temporal properties	Run-time	Time Basic Petri net	Reachability analysis
[55]	Agent movement	Run-time	Low-level	Transition firing rule
[39]	Agent behaviour	Run-time	Distributed PN Agent-oriented	Transition firing rule
[60]	Test case coverage	Testing	CPN	Reachability analysis

allow for more thorough analysis of agent interaction and behavior in design time. A similar, older model in [45] also focuses on robotic tasks. A Petri net place represents robot action while a transition model required conditions of those actions. A challenge in modelling an agent architecture is evident when considering more complex agents or environments. It would be a difficult task to capture all the possible states and actions an agent can assume.

A second area of work aims at verifying agent plans. An agent plan is the sequence of actions that would lead an agent to reach its goal. A methodology for multi-agent plan verification based on Predicate/Transition nets is proposed in [75]. The approach depends on the reachability analysis of

planning graphs through verifying the dependency between actions to reach a plan goal. A different method relies on generating the state space of a Petri net to capture all possible behaviours of an agent [2]. This is then used for model checking a temporal logic specification of a correct agent plan.

A third body of work examines interaction of agents, such as cooperation and interaction protocols. Agent communication languages such as KQML and FIPA define message exchange between agents that hold speech acts. Interaction protocols are used to structure the sequence of messages and define distinct conversations. The Petri net models of interaction protocols typically map the event of sending a message to a transition and the state of a conversation to a place. Furthermore, there are roles an agent assumes when engaged in an IP, where it would assume different states as the conversation progresses. A participating agent is modeled by its role within an interaction protocol and the states it can assume within a conversation.

An early model [20] explored CPN in modelling agent communicative interactions as conversations, defined as a pattern of message exchange that agents agree to follow in communicating. A place is the state of an agent, a transition models the message sending, and transition guards represent constraints on communications. Guards are boolean expressions implemented to ensure that the sender, receiver, and ID fields in a message correctly correspond to those in the initiating message of the conversation. This ensures the correct sequence within the conversation. The authors created a useful CPN model for protocol specification, but the report makes no attempt to analyse PN properties of interaction protocols.

Additional models were introduced in the past [21] [56] showing minor variations and objectives. One effort by [35] aims to compare models and analyse their scalabiltiy. The authors differentiate Petri net models in terms of what a place represents, and whether it is state-based or message-based. In state-based models, a place is mapped to the state of the conversation, while in message-based places, the message meaning is captured. Based on the

number of roles, messages, and conversations of an IP, a scalability analysis is presented.

An alternative approach to modelling communication is coordination of an agent's behaviour. The work in [63] and [77] both rely on a Petri net coordinator agent. In [63], the goal is to analyse and verify coordination protocols based on a peer-to-peer bidding algorithm of robots for the purpose of exploring an unknown environment. A Stochastic Petri net is adopted to design the coordination algorithm, where each robot is modeled as a Petri sub-net representing the process of exploring the next target cell. A different scenario is found in [77], where the decision process of an agent is modelled. A Condition/event net is used, in which a transition denotes the event of making a decision, and places connected to it represent the conditions enabling that decision. Agent decision nets are incorporated into a cooperation Petri net as sub-nets, which handles the allocation of roles and communicating commands to agents.

2.3.2 Run-time Models

A different line of research focuses on Petri-net techniques to verify and control agent behaviour during run-time. The authors of [9] focus on temporal properties of a real-time system and present a run-time verification method by comparing observed behaviour with a Time Basic Petri net model of expected behaviour and specified time constraints. Moreover, the idea of supervisory control during run-time is explored in [55] and [39]. A model of a supervisor for collision avoidance is created in [55], which enforces constraints on the mobility of agents in a shared space. A Petri net place represents an actual room or place within the environment, and a transition reflects robot movement from one space to other. The authors propose to restrict entry to a room by introducing a control place, defining the allowed quantity of robots. The work of [39] has implemented run-time control through a coordination agent and transition firing rules. A Distributed Agent-oriented Petri

net model controls the behaviour of a robot soccer team of eleven agents. Three agent models were designed to reflect the behaviour of each type of robot, where the current state of an agent is based on Petri net markings.

Petri net Models of communication during run-time are implemented for monitoring and debugging interaction protocols. Previous work, whether state-based or message-based, constructed a single Petri net to describe the flow of an interaction protocol. In this case, it is possible to model one or more than one conversation between two agents at the same time using token colour types. To handle different protocols, [5] indicates the possibility of linking Petri nets using connector places.

As an alternative to the design and run-time phases, the work in [60] focuses on model-based design and testing. The authors present a CPN model that captures the behaviour of cooperative robots to generate test cases. They report on the applicability of test case generation algorithms supporting the coverage of the underlying CPN models with respect to different testing criteria.

2.3.3 Discussion

Numerous Petri net models have been proposed for MAS that contribute to efforts to analysis and verify agent behaviour and interactions. In modelling agent behaviour and plans, reachability analysis has been shown to be useful in predicting possible agent actions. This has been implemented to verify accepted and correct behaviour [2]. This approach would be suitable for cases of simple behaviour, but in the case of a large number of agents with a large set of actions, reachability analysis would suffer from high state space complexity.

Petri net models of interaction have been successful in representing the flow and message exchange between agents. Past work has shown similarities in how they capture agent communication. For the most part models adopted a state-based approach in which places represent the state of interaction. This

approach then would follow the order of message flow. This can present a limitation in monitoring interaction with flexible arrival of messages.

For run-time monitoring only few [54] [57] authors have presented techniques for verifying interaction. A drawback in previous models is the inability to capture multiple interaction protocols in a single model. This is important because an agent is expected to engage not in a single IP, but in several.

A Petri net model that can capture multiple protocols provides a complete state of ongoing conversation within a MAS. This is useful in environments that include different protocols of interaction where an agent may participat in more than one distinct protocol. Monitoring different protocols can allow the analysis the overall state of conversations.

2.4 Summary

In conclusion, we have presented a review of the current verification approaches to ensure correct and predictable agent communication. We have considered the life-cycle phase where the verification is applied: at design time, in development and at run-time, the last of which is the focus of this project.

Verification of interaction during the design phase ensures the correctness of the interaction model. Applied approaches have mostly focused on formally verifying the satisfaction of a specific property. A drawback of formal verification is that knowledge of the specification language is required which makes it more suitable for design time.

Verifying during run-time is applied during development for testing and debugging. Run-time approaches for agent interaction are not as widely explored as formal and logic-based methods. They can help designers and developers understand errors and unexpected behaviour.

A key component of run-time verification is the monitoring mechanisms

of agent communication. What information will be collected and how they are presented determine the scope and efficiency of verification. Petri net-based methods were found to be one of the most commonly used approaches for capturing the trace of interaction.

Over the years, a number of Petri net models have been proposed for verifying correct interaction. Mostly, models of interaction have had Petri net places represent a state of interaction within a protocol. The flow of messages would transition the conversation from one state to the next. There are also models in which a place represents a message within the modeled protocol. Whether it is message-based or state-based, the Petri net models a single interaction protocol. For monitoring tasks, this would present a partial view of agent conversations in which agents are expected to engage in different protocols simultaneously.

This research considers run-time monitoring of agent interaction protocols. The proposed IPN defines a protocol as a Petri net marking to handle monitoring simultaneous protocols. In addition, an agent-based analysis facilitates the detection of errors in an interaction and gives access to message content being exchanged.

Chapter 3

Agent Interaction Monitoring and Control

The proposed framework is based on run-time monitoring and detection. It comprises two components: a Petri net model for monitoring and a governing agent for interaction management. The architecture of the monitor and control framework entails capturing a trace of execution through the Interaction Petri net, which then is analysed by a governing agent to determine whether undesired scenarios in communication are detected. The result of analysis determines what action is required to ensure correct interactions among agents.

The first process is to monitor and analyse the interaction to identify any undesired scenarios. A key property of an interaction protocol to examine is message sequence. This can be achieved through a simple Petri net analysis technique of marking analysis. A marking describes the state of the Petri net in terms of the number and location of tokens. In the proposed IPN, the components involved in an interaction protocol are: participants as places, messages as tokens, and message sequence as a Petri net marking. Defined as a colour set, a message token holds a number of values: sender, receiver, IP, sequence ID, body, and time of emission and arrival. This structure

of variables enables the identification of messages along the timeline of a particular IP sequence.

The second process is to take a control action depending on the analysis output. It is important to note, this step depends on the nature of the environment, as it entails altering elements within the space where agents reside and execute. One must examine what a designer can manipulate during execution. Our monitoring and control technique aims to minimise the outcomes of interaction errors. This makes the control method dependent on the nature of the interaction.

After presenting the proposed framework, a description of the selected case studies is given. A case study should reflect MAS properties to test and evaluate the framework. Such an environment should display challenges of communication among agents in an open, asynchronous system. This will provide an opportunity to examine Petri nets in run-time monitoring and the applicability of a governing entity executing at the environment level.

The remainder of this chapter is organised as follows: Section 3.1 presents the definition of the Interaction Petri net. Section 3.2 presents the governing agent design and components. The selected case studies are presented in Section 3.3, and lastly Section 3.4 is the chapter summary.

3.1 Interaction Petri Net (IPN)

The goal of the proposed Interaction Petri net is to trace message exchanges between agents for the objective of monitoring and control. IPN is focused on monitoring multiple concurrent interaction protocols.

Previous work [20][35] has built a net to model a specific interaction protocol; in contrast, the aim of IPN is to model a collection of interactions. The objective is to provide an all-inclusive view to enable the analysis of the state of communication, such as the status of send messages, exchange of environment information, and open requests.

The IPN is based on Coloured Petri nets, a high-level Petri net that differs from low-level Petri nets in terms of tokens. Here, tokens hold a data value and belong to types. A place in the IPN represents an agent, and a transition represents the action of sending and responding to a message. Tokens are defined to feature messages available to agents with a structure that supports metadata, as given in Table 2.2.

The IPN captures communication between agents in two events; sending and receiving a message. A token is a message instance that the agent intends to send to another agent. The event of firing a transition results in consuming a token from an input place and producing it in an output place. Hence, the movement of tokens replicates the dynamic interaction taking place between agents. Based on [38], we define the IPN as a seven-tuple = (P,T,A,Σ,C,G,E):

1. P is finite set of places. A place p is added for each agent.

2. T is a finite set of transitions t such that $P \cap T = \phi$. A transition t is created for each pair of places to model the event of receive message.

3. $A \subseteq P \times T \cup T \times P$ is a set of directed arcs. The set of directed arcs represents a link between an agent *place* and the event of send message *transition*.

4. Σ is a finite set of non-empty colour sets. In the IPN, a message colour set is defined as a record colour set [1] of other colour sets and holds values such as sender, receiver, message sequence ID, message body, time of sending and time of receiving.

5. $C : P \rightarrow \Sigma$ is a colour set function that assigns a colour set to each place.

[1] A record colour set is a term based on the CPN tools [58] described as "A fixed-length colour set whose set of values is identical to the Cartesian product of the values in previously declared color sets. Each of the component colour sets may be a different type and each is identified by a unique label so that each field is position-independent".

6. $G : T \rightarrow EXPR$ is a guard function that assigns a guard, a Boolean expression to each transition t. The guard defines a constraint on firing a transition based on the recipient of the message.

7. $E : A \rightarrow EXPR$ is an arc expression function that assigns an arc expression to each arc a. The expression binds the value of a token message to a variable with the colour set of the place connected to the arc.

The definition of IPN provides a new class of Petri nets for modelling interaction protocols. An interaction protocol can be represented by the Petri net marking. We can consider a protocol definition in terms of roles and messages. Suppose that protocol P contains the set of roles $R = \{r1, r2\}$, the set of messages $M = \{m1, m2, m3\}$, and agents $A1$ and $A2$ that would participate as roles $r1$ and $r2$ respectively. Then the structure of IPN will consist of the set of places $P = \{p1, p2\}$, where place $p1$ represents $A1$ and place $p2$ represents $A2$ as shown in Figure 3.1.

A place in IPN is mapped to the agent rather than a role in a protocol. This allows IPN to model an agent in different protocols and for the agent to play more than one role. An agent place encompasses the set of messages received from other agents. It also contains messages created by the agent that are intended to be sent to other agents but have not yet been fired through the input transition. In other words, token messages are mapped to the actual messages that an agent creates and receives. For each message created by an agent, IPN generates a corresponding token message within the agent's place. When a message is received by an agent, IPN fires the appropriate transition to produce the corresponding token message in the recipient agent's place.

Next is the set of transitions $T = \{t1, t2\}$, which capture the event of firing a message from place $A1$ to place $A2$ and from place $A2$ to place $A1$. A transition in IPN represents the event of an agent receiving a message. When

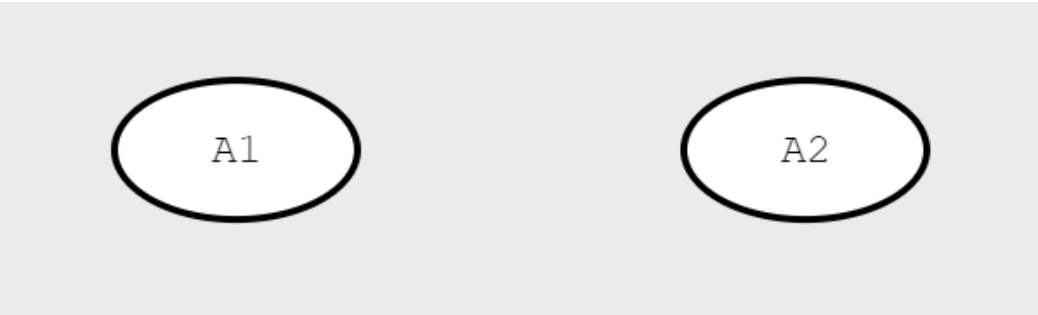

Figure 3.1: The first step is creating places A1 and A2.

a transition is fired, the received message token moves from the sender's place to the recipient's place, indicating that the message has been received.

Transitions are connected to places within the Petri net through arcs. An arc directed from a transition to a place indicates the direction of a token when fired. The set of arcs in IPN define a pair of arcs for each transition and its input and output places. The IPN structure with transitions and connecting arcs is shown in Figure 3.2.

Next is the representation of the message structure, which corresponds to the colour set defined in IPN. Typically, messages contain metadata as found in Table 2.3. The colour set defined here for IPN is MSG, shown in figure 3.3, and contains message information including: sender, receiver, IP, sequence ID, message body, time sent and time received. An important data value is the sequence ID which is an identification of the protocol conversation to support multiple protocol instances. The message structure can be supplemented with additional values by updating the MSG colour set. Given that tokens in Coloured Petri nets enable structured data definition, metadata can be added.

The next element of IPN is the guard function. A transition guard function is a Boolean expression applied to check some condition for the transition to be enabled. This provides a way to apply constraints on the firing of transitions and validate the value of tokens. In IPN, the guard function enforces

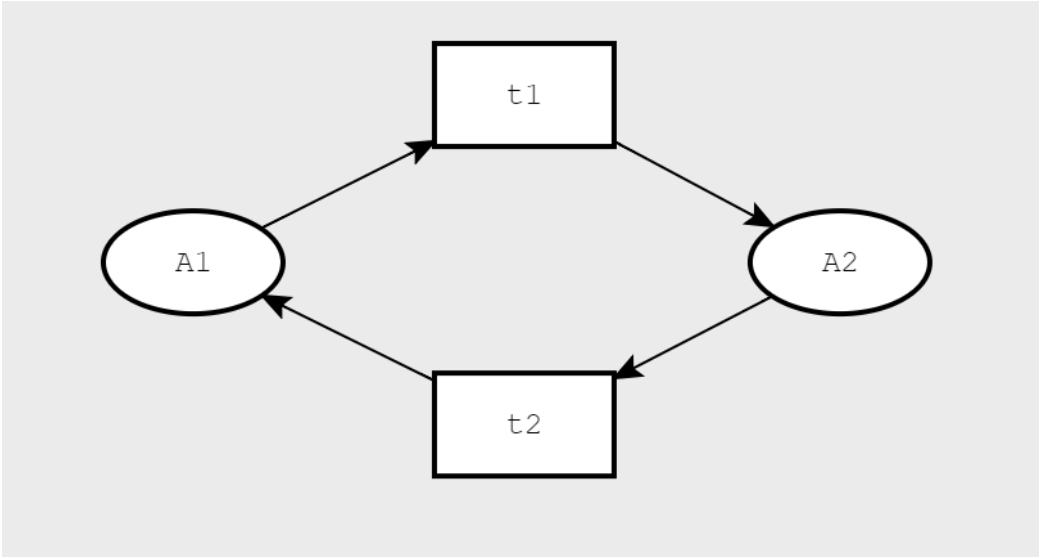

Figure 3.2: The second step is to add transitions t1, t2 and the connecting arcs.

```
colset MSG = record sender: STRING * receiver: STRING *
                    ip: STRING * seq_id: INT *
                    body: STRING * time_s:Time *
                    time_r: Time;
```

Figure 3.3: The MSG colour set.

a check on the receiver name field of message tokens. The guard function is defined as $G(t) = m.receiver == P_o.name$, where $m.receiver$ is the receiver field of a message token m and P_o is the output place connected to transition t.

The guard expression is applied to ensure that a transition is enabled only when the input agent place contains message tokens that are expected to be produced at the output place. A transition is fired when the recipient agent receives the actual message.

The last element is the arc expression. An arc expression of a Coloured Petri net determines how tokens are consumed from an input place and how they are produced in an output place. The expression defines the binding of a token value and how it is processed when a transition is fired. In IPN, input and output arc expressions are simply defined to bind the value of a single token message. An arc expression a is defined as $E(a) = x$ where x is a variable of type MSG.

An example of the complete IPN structure is shown in Figure 3.4. The example IPN contains:

- Places that represent agents.
- Transitions connected to places with input and output arcs.
- A colour set with a message data structure determined by the required metadata of protocols used in the environment.
- A guard function in each transition for testing token messages to be fired.
- An input and output arc expression that binds the value of a token message to a variable when fired.

After the IPN structure is complete, the protocol can now be represented through the placement of message tokens within the places of IPN. The

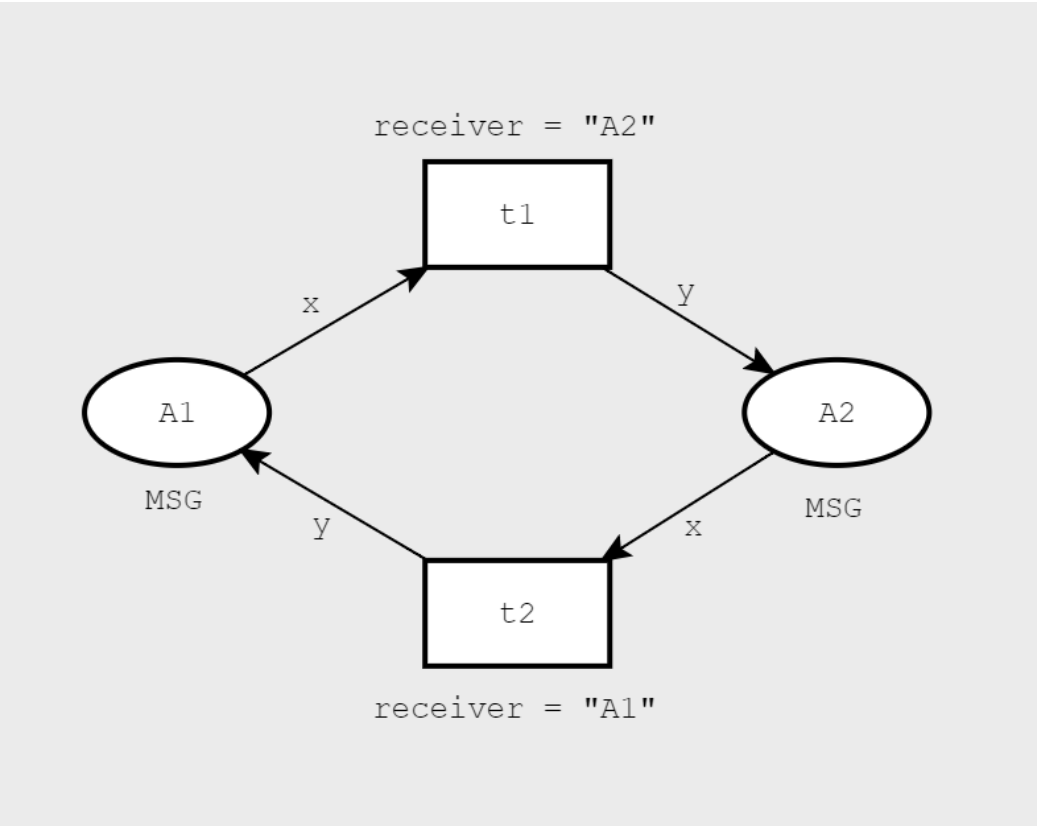

Figure 3.4: Complete IPN with places,transitions, arcs, guards, and an assigned colour set.

marking $A1 = [m2]$ indicates that place $A1$ contains the token m2, and the marking $A2 = [m1, m3]$ means that place $A2$ contains the tokens m1 and m3. According to the message token information; agent A1 has received m2 from agent A2, messages m2 and m3 are emitted by agent A2 and received by agent A1.

To demonstrate the IPN representation of multiple instances of an interaction protocol and multiple protocols, we present two examples. First, we consider the Contract Net protocol (CNP) shown in the previous chapter, Figure 2.1. The simple interaction includes two roles, an initiator and a participant, and the expected message exchange. Second, we consider a scenario

where the FIPA request protocol is used alongside the CNP protocol. This example shows how IPN models more than the protocol and enables agents to play more than one role.

Algorithm 1 Add an Agent to IPN

Input: new_agent
Procedure:
    $cpn \leftarrow ColouredPetrinet$
    $p \leftarrow cpn.add\_place(new\_agent.name)$
    $all\_places = cpn.place()$
    If $LENGTH(all\_places) > 1$ **then:**
        For Each $p \in all\_places$:
            If $p.name \neq new\_agent.name$ **then:**
                Create Transition from p **to** new_agent:
                $transition\_name = p.name + "to" + new\_agent.name$
                $expression = message['receiver'] == new\_agent.name$
                $t \leftarrow Transition(transition\_name, guard = expression)$
                $cpn.add\_transition(t)$
                $cpn.add\_input(p.name, t)$
                $cpn.add\_output(new\_agent.name, t)$
                Create Transition from new_agent **to** p:
                $transition\_name = new\_agent.name + "to" + p.name$
                $expression = message['receiver'] == p.name$
                $t \leftarrow Transition(transition\_name, guard = expression)$
                $cpn.add\_transition(t)$
                $cpn.add\_input(new\_agent.name, t)$
                $cpn.add\_output(p.name, t)$

To create an IPN, a process is provided in Algorithm 1, we follow the steps to create the IPN. Initially, we start with an empty Coloured Petri net, and the procedure describes how an agent is added to the IPN. The input is an agent for which we need to monitor its communication in the environment, which here is the agent that plays the initiator role. First step is to add the place $agent\_A$. Next step is to retrieve all places of the Petri net and check to see whether there is more than one agent place. Since we started with an empty net, $agent\_A$ is the only place added, the procedure stops.

To add the second agent that plays the participant role, we restart the procedure with input *agent_B*. This time, the condition at line 6 is true and we move to the loop in the next line. For every agent place different than the newly added agent within the IPN we create two transitions: a transition that connects from the new agent place to the place p and vice versa. Here the place *agent_A* is the only place that evaluates to true at line 8. The first transition is named *A_to_B* which indicates that *agent_A* is the input place and *agent_B* is the output place. The second transition is *B_to_A* where *agent_B* is the input place and *agent_A* is the output place. The guard on each transition is defined to ensure firing tokens that are received by the output place agent. The resulting IPN is illustrated in Figure 3.5.

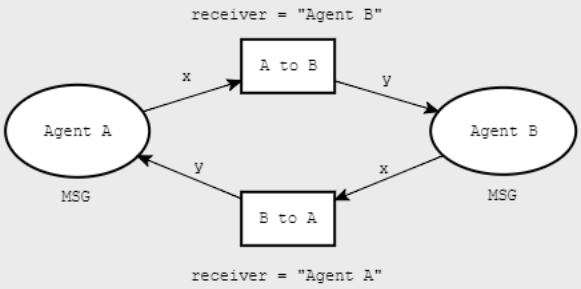

Figure 3.5: IPN of agent A and B

We suppose a conversation would flow as the set messages shown in table 3.1. The protocol is initiated when agent A creates and sends a CFP message, the corresponding token is added to place *agent_A*. When agent B receives the CFP message transition *A_to_B* is fired and the CFP token message is produced at place *agent_B*. The markings of agents A and B after firing the first message are: $A = []$, $B = [m1]$. When message m2 is created and emitted by agent B, the updated marking would be $A = [m2]$, $B = [m1]$. The marking is updated based on the two main events: first is when an agent creates an actual message and sends it, and the second is when an agent receives an actual message. Table 3.2 shows the flow of the

interaction that unfolds between agents A and B within the environment and the corresponding updates within the IPN marking.

Table 3.1: A conversation example of the CNP protocol between agent A as an initiator and agent B as a participant

A conversation example of CNP protocol between agents A and B
m1= {sender = Agent A, receiver = Agent B , ip = CNP, seq_id = 1, body = CFP } m2= {sender = Agent B, receiver = Agent A, ip = CNP, seq_id = 1, body = Propose } m3= {sender = Agent A, receiver = Agent B, ip = CNP, seq_id = 1, body = Accept } m4= {sender = Agent B, receiver = Agent A, ip = CNP, seq_id = 1, body = Inform-done }

Table 3.2: CNP protocol message flow between agents and IPN marking updates

Agent Interaction	IPN marking
A sends m1	$A = [m1]$, $B = []$
B receives m1	$A = []$, $B = [m1]$
B sends m2	$A = []$, $B = [m1, m2]$
A receives m2	$A = [m2]$, $B = [m1]$
A sends m3	$A = [m2, m3]$, $B = [m1]$
B receives m3	$A = [m2]$, $B = [m1, m3]$
B sends m4	$A = [m2, m3]$, $B = [m1, m3, m4]$
A receives m4	$A = [m2, m4]$, $B = [m1, m3]$

To model another instance of the protocol, a third agent is added to the model by repeating the procedure with input $agent\_C$. This time $agent\_C$ plays the role of another initiator. The updated IPN is shown in Figure 3.6, with another example conversation between agent b and c as shown in table 3.3. The following markings show the tokens in places A, B, and C, which represents the status of conversation at the end of the message flow in Tables 3.1 and 3.3: $A = [m2, m4]$, $B = [m1, m3, m5, m7]$, $C = [m6]$.

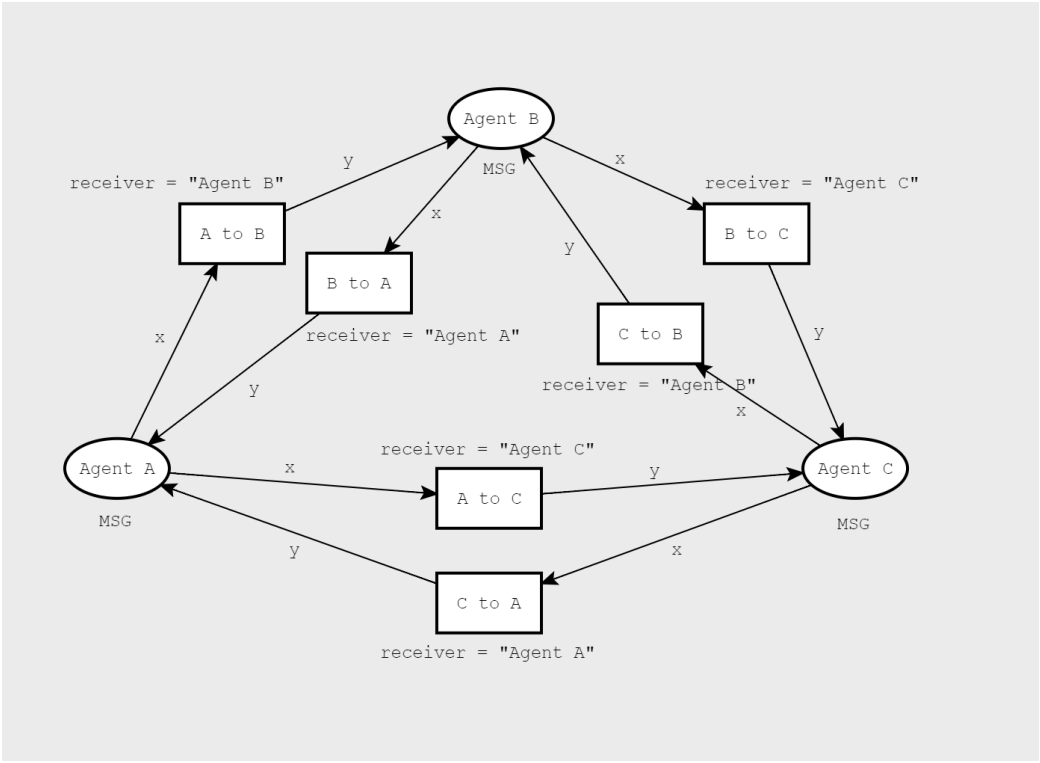

Figure 3.6: IPN of agents A,B and C.

The previous scenario illustrated two conversations of the CPN protocol where agent A accepts the proposal of agent B for some task. We now suppose that Agent B requires assistance from an additional agent, Agent D, to complete the task. This leads Agent B to initiate a FIPA Request protocol. The Request protocol, shown in Figure 3.7, is a simple protocol that includes two roles and an expected flow of three messages. In this interaction, Agent B assumes the role of the initiator and Agent D takes on the role of the participant. With a new agent, the IPN model is updated to include Agent D as shown in Figure 3.8.

An example conversation is shown in table 3.4. The message flow indicates that agent B sends a request to agent D to perform some action. Agent D accepts the request, and once the action is complete, it sends an inform-done

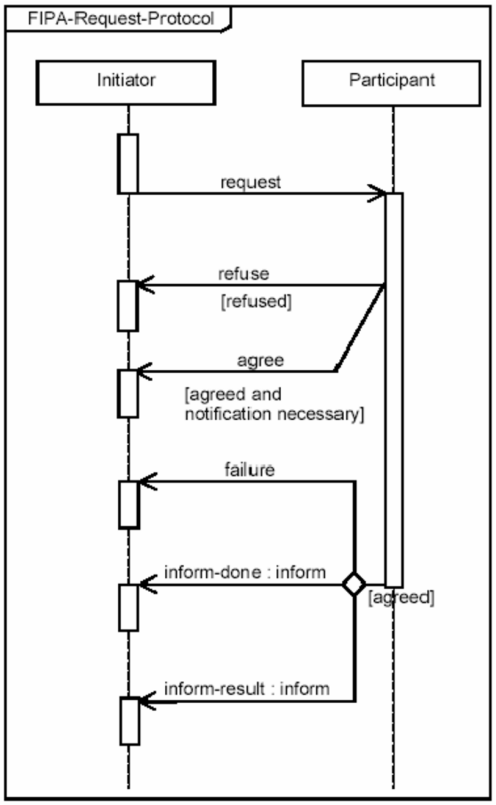

Figure 3.7: FIPA Request protocol.

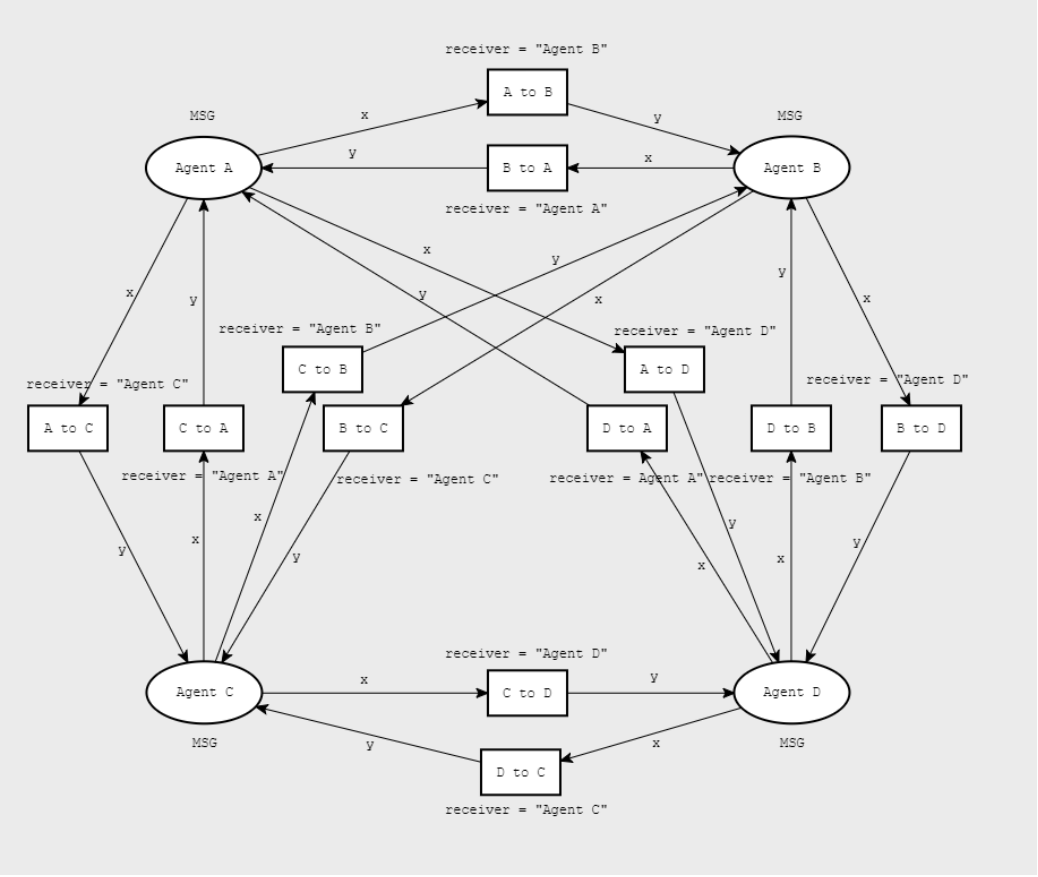

Figure 3.8: IPN of agents A,B,C and D.

Table 3.3: A conversation example of the CNP protocol between agent C as an initiator and agent B as a participant.

A conversation example of CNP protocol between agents C and B
m5= {sender = Agent C, receiver = Agent B , ip = CNP, seq_id = 1, body = CFP } m6= {sender = Agent B, receiver = Agent C, ip = CNP, seq_id = 1, body = Propose } m7= {sender = Agent C, receiver = Agent B, ip = CNP, seq_id = 1, body = Reject }

message. The updated markings after this protocol is complete are as follows: $A = [m2, m4]$, $B = [m1, m3, m5, m7, m9, 10]$, $C = [m6]$, $D = [m8]$.

Table 3.4: A conversation example of the Request protocol between agent B as an initiator and agent D as a participant.

A conversation example of Request protocol between agents B and D
m8= {sender = Agent B, receiver = Agent D , ip = Request, seq_id = 1, body = Request } m9= {sender = Agent D, receiver = Agent B, ip = Request, seq_id = 1, body = Agree } m10= {sender = Agent D, receiver = Agent B, ip = Request, seq_id = 1, body = Inform-done }

The final marking represents a total of three conversations: two are based on the CNP protocol, and one is based on the Request protocol. The distribution of tokens indicates both the history of exchanged messages and the current state of each protocol.

The proposed IPN focuses on the current interaction status of each agent and provides an overall view of the different conversations and protocols that take place. With places representing agents, IPN is able to model the agents that would communicate and the different protocols they are engaged in, not just a single specific protocol.

3.2 Governing Agent

A governing agent behaviour entails two processes: the first is to analyze the IPN marking for issues in interaction, and the second is to take action based on the issue detected. Those two processes are repeated within a cyclic behaviour.

The marking of IPN corresponds to the status of conversations and the messages that are sent and received. This is a direct approach to monitoring protocols that allows for immediate detection when applied during run-time.

The following subsection presents a description of the detection and control processes.

3.2.1 Detection and Control

The **detection** process objective is to detect unwanted scenarios that may happen in interaction. The success of a protocol is complete enactment, however there are certain scenarios that could hinder protocol enactment and consequently hinder the achievement of agent goals. The detection process is applied to detect five interaction issues that could arise in any protocol, identified as: IP delay, lost messages, busy-wait, message delay, and termination. Monitoring and detection are designed to analyze the status of conversations to determine if any of these five scenarios is present. There is a process proposed for each specific one.

The IPN acts as a monitoring mechanism to track the sending and receiving of messages. The GA takes as input the marking of an IPN to analyze the status of conversations. The detection defined for IPN enables the detection of IP delay, lost messages, busy-wait, and message delay as follows:

1. IP delay occurs when an IP is not complete within an expected time period. The delay is checked for by including a time threshold of when the end message is expected at a specific place, and when this threshold is reached, a delay is recorded.

Definition 2: *D is the set of first messages of delayed protocols,* $D = \{d \mid l \notin T_p \wedge t - d.recorded.time >= e\}$ *where l is the expected last token based on d, p is the expected place of token l,* T_p *is a set of tokens at place p, t is the current time, e is the maximum time token l is expected to arrive.*

2. Lost messages in the IPN are detected by checking message tokens that are not fired by a transition yet. Transitions are fired in the IPN when a message token is received by its recipient.
   Definition 3: *L is the set of lost messages,* $L = \{l \mid l \in T_p \wedge l.sender == p.name\}$*, where* T_p *is the set of tokens at place p.*

3. Busy-wait scenarios are checked whenever an agent receives a new request. An agent status is indicated by a place label. When an agent receives the first message token of a specific protocol, the place label value is updated to *Busy*.
   Definition 4: *B is the set of busy-wait messages,* $B = \{b \mid b \in T_p \wedge b.body == f(b.ip) \wedge p.status == Busy\}$*, where f(x) returns the first message in all defined protocols and* T_p *is a set of tokens at place p.*

4. A message transmission delay occurs when a message transmission time has exceeded an expected time range. It is applied by including a time threshold for when a message token must be present at the specified received place; when it is over that time frame, a message delay is recorded.
   Definition 5: *M is the set of delayed messages,* $M = \{m \mid m \in T_p \wedge m.received.time - m.sent.time >= e\}$*. where e is the maximum time token d is expected to arrive by, and* T_p *is a set of tokens at place p.*

5. The termination of an agent happens in IPN when a monitored agent that is part of the IPN terminates. An agent active status is indicated by a place label.
   Definition 6: *P is the set of places in IPN, $P = \{p \mid p.status == Terminated \vee p.status == Active\}$.*

Monitoring agent interaction with the IPN is achieved through mirroring the events of creating, sending, and receiving a message in real time. The definition of lost, delay, and busy scenarios in the IPN facilitates their detection as they occur.

The second component of the governing agent is the **control** entity. The governing agent aims to react to unexpected events in a way that does not influence agent interaction choices but rather help agents in following protocols correctly [61]. In contrast to the supervisory roles in [55] [39] which enforce constraints on the behaviour of agents.

The control component comprises a set of actions designed to deal with interaction issues. The governing agent is designed as a reactive agent whose decision process is specified by mapping actions to interaction issues. Based on the scenario, it picks an action from the set of actions available.

The control process design is directly influenced by the agents' goals and environmental characteristics. The objective is to control the negative effect of interaction issues on agent behaviour and performance. Software designers should specify the control action design to create an effective governing agent. The designer determines the appropriate reactions to the IPN detection output.

3.3 Case Studies

An evaluation environment for agent communication should reflect the characteristics of MAS where agents are heterogeneous and have asynchronous

communication. Furthermore, a reasonable agent interaction case-study would include interaction protocols that satisfy the criteria presented in [72] [15]. The following scenarios consist of interactions that cover the discussed criteria. First is a scenario that involves an agent engaged in more than one different protocol at the same time. A second scenario involves an agent participating in multiple instances of a specific protocol. Third is a scenario with at least one protocol that involves multiple role instances. Lastly, a protocol with one or more exception points. The selected case studies include those scenarios of interaction for the evaluation of the proposed IPN model for monitoring communication and detection.

The following sections present the selected case studies, first is a collaborative treasure hunt among three types of agents, and the second is an auction of an item with multiple participants.

3.3.1 Treasure Hunt Case Study

This research adapts the Dedale platform [37] designed to create a testbed that is open, dynamic, and asynchronous for MAS research. Dedale presents a treasure-hunting problem in which heterogeneous agents move through a connected graph to collect treasure. The graph also contains Well nodes that cause an agent to terminate when visited. There are different kinds of agents: collectors to pick up treasure, explorers to explore the graph for treasure, and tankers who are responsible for accumulating it. However, limited perception and communication range impose difficulties for agents to interact and achieve their goals.

We created a similar environment to evaluate the proposed Petri net model for run-time monitoring and control. The environment properties adapted from Dedale are partial observation, heterogeneous agents with distinct behaviour, and a common goal that drives collaboration. There are a number of elements eliminated from Dedale such as types of treasure, expertise required to pick treasure, Golem type agents that move treasure around,

and adversarial teams. This adjustment was made to focus on the collaborative interaction between the three types of agents.

This case study is selected to evaluate communication in an environment that contains four of the defined undesired scenarios: IP delay, lost messages, busy-wait and agent termination. Such scenarios may occur in a multiagent system characterized by restricted transmission range, network latency, different protocols, and partial observability. Agent interaction and collaboration is assessed in the face of such scenarios.

Figure 3.9 displays a grid world of the adapted environment. Movement of agents is horizontal and their observation is limited to the node they are in, for example an agent would not observe if there is a well in the next node. As agents move within the environment they can gain knowledge of another agents if they are in adjacent nodes, they would exchange contact information. Communication is asynchronous and limited. The range of interaction is set to three steps away, that is if the recipient can be reached with three steps it is considered within range. The behaviour of agents are as follows:

Explorer behaviour. An explorer agent will search the environment based on the Depth First Algorithm to look for treasure. The explorer is not able to pick up a treasure but it will request collectors to do so.

Collector behaviour. The collector's goal is to pick up treasure when it comes across it or when informed by an explorer agent. A collector agent will move around the grid randomly unless reacting to an explorers request. Also, a collector maintains a bag for collected treasure with a specific maximum load. Once the maximum load is reached, a collector will not be able to move and require a tanker agent's help with the treasure load.

Tanker behaviour. The goal of a tanker is to accumulate picked up treasure from collector agents. A tanker agent moves around the grid randomly unless requested by a collector.

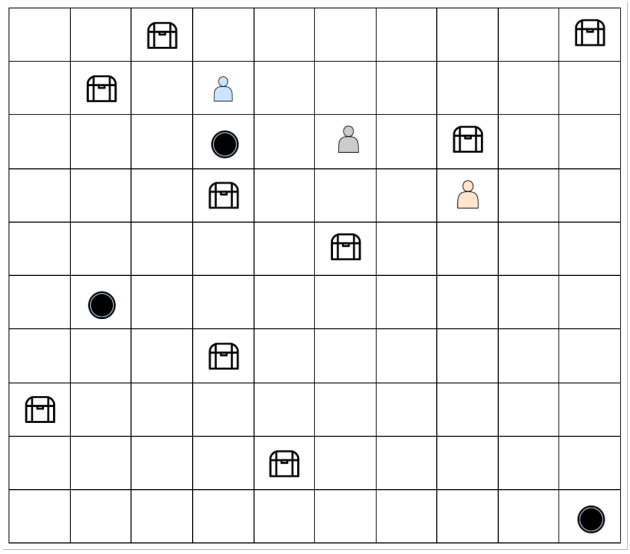

Figure 3.9: Treasure hunt environment, showing the placement of treasure depicted as treasure chests, well nodes depicted as black circles, and three kinds of agent depicted with different colours.

The case study builds a grid environment that exhibits a dynamic MAS. The Dedale platform provides agent communication through passing information but does not specify an interaction model for agents. To create a suitable case study for evaluating agent interaction, we introduce three interaction protocols. Based on each of the agents behaviour, an interaction protocol is defined to complement the agent objective.

The first protocol is a Request-Collector (RC) protocol between an explorer and a collector. The protocol is initiated whenever an explorer agent comes across a treasure, then the agent creates a request to send to all known collector agents. The explorer will stay put until it receives an accept message from a collector. A collector agent that receives a request and is free, will accept and start moving towards the treasure location for collection. This interaction is shown in Figure 3.10.

The Request-Collector protocol involves an agent participating in multiple instances of a protocol, as a collector agent may receive more than one

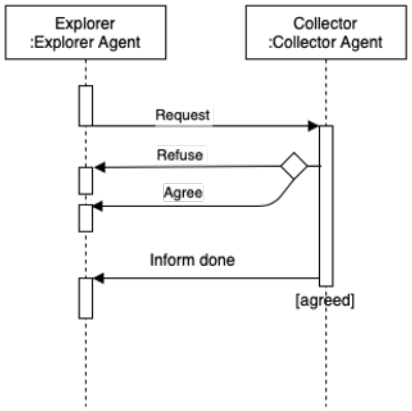

Figure 3.10: Request Collector Interaction Protocol

request. Also, it involves multiple role instances, as such a request is made to multiple collectors, and it is a a protocol with one or more exception points, as a refuse message stops the protocol instance. The RC protocol includes three scenarios that cover the criteria desired for interaction evaluation.

The second protocol is a Request-Tankern (RT) protocol between a collector and a tanker. This protocol is initiated whenever a collector agent reaches the maximum load of its bag. The collector sends a request to all known tanker agents. A tanker agent that receives a request and is free, will accept and start moving towards the collector location for handover. This interaction is shown in Figure 3.11.

The Request-Tanker protocol involves a scenario where an agent is engaged in more than one different protocol at the same time. This occurs for collector agents when they initiate a Request-Tanker protocol and receive a request to pick up treasure through a Request-Collector protocol.

Third protocol is Warning, which is a simple broadcast message. Shown in Figure 3.12, the protocol is followed by every agent type to communicate the location of a well node.

Based on the described behaviour of the three types of agents, the interaction model is defined for the expected communication between them. Figure

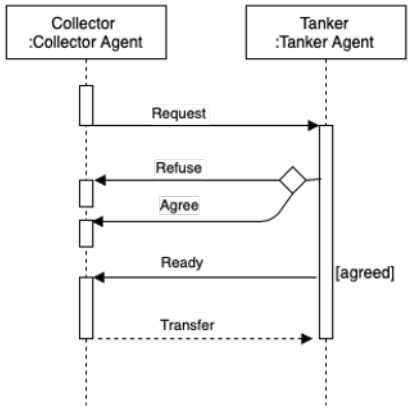

Figure 3.11: Request Tanker Interaction Protocol

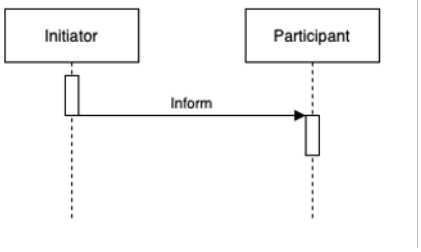

Figure 3.12: Warning Interaction Protocol

3.13 shows an IPN model with three agents of the case study. The following markings show the tokens in places Explorer, Collector and Tanker which represents the status of two complete protocols as illustrated in Tables 3.5 and 3.6: $Explorer = [m2, m3]$, $Collector = [m1, m5, m6,]$, $Tanker = [m4, m7]$.

Table 3.5: A conversation example of Request Collector protocol between Explorer and Collector.

A conversation example of Request Collector protocol between Explorer and Collector
m1= {sender = Explorer, receiver = Collector , ip = RC, seq_id = 1, body = request } m2= {sender = Collector, receiver = Explorer, ip = RC, seq_id = 1, body = agree } m3= {sender = Collector, receiver = Explorer, ip = RC, seq_id = 1, body = inform-done }

Table 3.6: A conversation example of Request Tanker protocol between Collector and Tanker.

A conversation example of Request Tanker protocol between Collector and Tanker
m4= {sender = Collector, receiver = Tanker , ip = RT, seq_id = 1, body = request } m5= {sender = Tanker, receiver = Collector, ip = RT, seq_id = 1, body = agree } m6= {sender = Tanker, receiver = Collector, ip = RT, seq_id = 1, body = inform } m7= {sender = Collector, receiver = Tanker, ip = RT, seq_id = 1, body = transfer }

The detected four scenarios are followed with a control action. In the case of lost messages, the governing agent would forward the message to its recipient. And in the case termination, the GA will replace a terminated agent with an agent of the same type. The design of control in response

to the delay and busy-wait scenarios entails informing the agents with the protocol status they are engaged in. Based on each protocol the initiator agent will react differently to the updates.

An explorer initiates an RC protocol to known collectors and may receive more that one reply. The explorer will react to an IP delay update based on the number of other current IP instances which include received collector replies. If a collector is delayed but there are other collectors that sill may come and complete the task, the explorer will cancel the delayed collector. If the delayed collector is the only agent that agreed to come pick the treasure, then the explorer will not wait any more as to further continue with exploring the environment.

An explorer may also receive a busy-wait status on a collector that it is engaged with in an RC protocol. The explorer will check the number of received collector replies as well as with delay updates. The explorer will initiate new requests to known collectors if the busy collector is the only agent that replied to its request. If there are other collectors that may still complete the pick up, then the explorer will cancel the busy collector.

A collector agent initiates the RT protocol to known tankers. The collector may receive an IP delay or busy-wait update on one of the tankers it requested. The collector will respond based on the number of current RT conversations. The collector will wait for a delayed tanker to complete the request if its the only reply it has received. But if the tanker is busy then the collector will not wait and will drop one treasure from its bag to be able to move again. However, If there are other tankers that replied and still can complete the request, the collector will cancel the delayed or busy tanker.

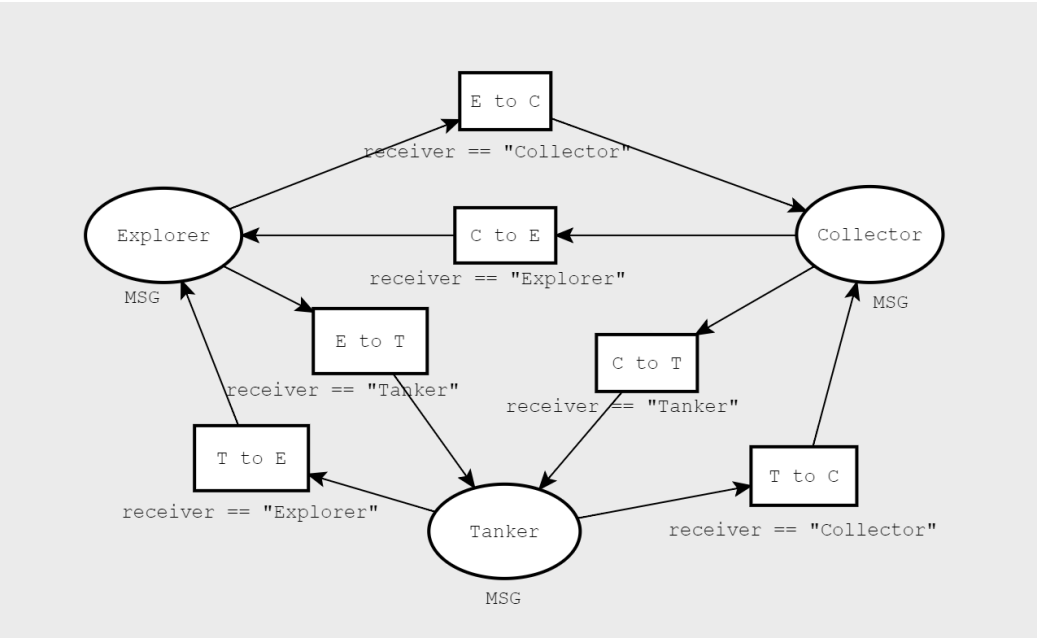

Figure 3.13: An example of an IPN in the Treasure hunt case study with three agents: Explorer, Collector and Tanker.

3.3.2 Auction Case study

The selected case study presents an environment with agents that participate in an English Auction. This type of auction is very common and involves a low starting price and ends with the highest buying price. Typically, an auctioneer initiates the auction of an item with a proposed price below the expected market value. Interested buyers show interest by proposing a price they are willing to pay. Once an auctioneer receives an incoming bid that is over the current price, it accepts and issues a new call for bids with the accepted price.

The objective of evaluation is to assess the effect of message delay on participating agents. In comparison to the treasure hunt case study, where agent interaction experienced scenarios of protocol delay, lost messages, busy-wait, and termination, the communication issue of concern here is transmission

delay. In an auction, a delay in receiving a CFP message could affect an agent's chance of winning.

The FIPA English Auction interaction protocol defined for an English auction is shown in Figure 3.14. The auction starts with a Call For Proposal (CFP) message, which contains the starting price of the item. Participants reply with a propose message including a proposed price. After the auctioneer receives a bid, it checks if the proposal is valid, that is, the bid holds an increase to the current price. If valid the bid is accepted and an Accept message is sent to the participant and a new Call For Proposal (CFP-2) message is sent to the other participants. Otherwise, the bid is rejected and the auctioneer sends a Reject message. The auction continues until there are no more interested bidders. If a specified time threshold has passed without receiving any bids, the auction terminates and the last bidder is announced a winner. An Inform message is sent to all participants announcing the winner [2].

The implemented case study includes six participant agents and an auctioneer agent. All participants share the same parameters: the wallet amount, which is the maximum amount they are allowed to reach when proposing a bid, and bidding strategy, which refers to the percentage increase they propose to bid. These parameters are fixed to ensure consistent behaviour and limit variability in auction results. This allows us to concentrate on protocol issues rather than variability caused by bidding strategies.

The goal of the control action associated with delay issues is to eliminate effects caused by delay on collected proposals of each CFP round. In the case of a CFP delay, the GA informs the auctioneer, which initiates a response behaviour. The behaviour is embedded with the auctioneer agent and it

[2] An alternative approach is described in the FIPA specification document:`http://www.fipa.org/specs/fipa00031/XC00031F.html`. An auctioneer announces a price and waits for a bidder that is willing to accept the proposed price. The auctioneer announces a new call for bids with an increased price as soon as one buyer indicates that it will accept the price. However, the implemented case study here follows the approach described above.

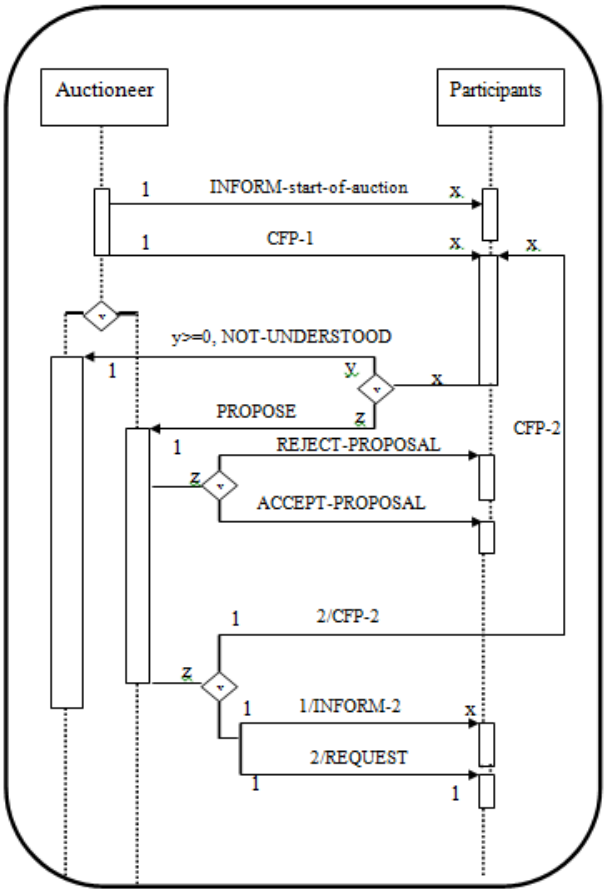

Figure 3.14: FIPA English Auction

involves a rollback to a previous price point and the start of a new CFP round. The reason behind this control action is to present bidders with an equal chance to propose and have their bids received by the auctioneer.

When a delay is detected, the price is saved. If more than one CFP is delayed, the GA takes the lowest saved price within the set of delayed CFPs. Then, an Inform message is sent to the auctioneer, which in turn starts a CFP-wait round. The difference here is that the auctioneer waits for a response from every agent before processing incoming bids. After all expected bids are received, the auctioneer processes them, which provides

agents the same chance for their bids to be accepted. However, there is the possibility that two agents may offer the same bid amount, in which case the auctioneer randomly picks a bid for acceptance. In contrast to the non-delayed case where the first valid bid received is accepted.

The Auction protocol involves multiple role instances of participants. This is one of the criteria based scenarios to be included in a case study for agent interaction.

3.4 Summary

This chapter has presented a detailed approach for run-time detection and control. A new class of Petri nets is proposed for the purpose of monitoring and analysing the state of communication. The IPN model design can be described as an agent-based model in which a Petri net place represents an agent, in contrast to previous models in which places represent a status of interaction. What distinguish the IPN is its ability to monitor multiple interaction protocols with the same Petri net.

The detection approach is based on detecting common interaction issues, where IPN model is used to define. The approach defines a communication scenario based on the message token placement within a marking. The detection process involves checking the placement of a token and the values a message token holds.

The second component consists of the GA control behaviour, designed with the aim of correcting undesired scenarios found in interaction. The control component defines a set of actions for the GA to act upon. Based on the detected issue, the GA reacts with the corresponding action. It is important to note that a control action is context dependant and its design depends on the environmental characteristics.

Also, this chapter have introduced two case-studies which will be used for evaluation of the proposed approach. These two case studies allows us to

consider the following scenarios for IPN evaluation: first is a scenario with an agent participating in different protocols simultaneously, a second scenario is with a protocol that involves multiple instances, third is a scenario that includes a protocol involving multiple role instances, and lastly a protocol with one exception point.

Chapter 4

Evaluation

The IPN model and GA are evaluated through experimental analysis using two case studies described in the previous chapter. We conducted two experiments on each case study; the difference between the first and second experiment is the application of GA control.

The objective of the first experiment is to examine IPN ability to detect the undesired scenarios and determine the efficiency of the detection process. The detection process involves token analysis within the Petri net. A concern for run-time detection is the processing time, the analysis of different protocols, and the handle of different instances of a single protocol. A different Petri net model of agent interaction is included in the experiment. This is to evaluate the proposed model in comparison to the alternative state-based approach found in the literature.

In the second experiment the GA control is activated to evaluate the complete approach of detection and control. It examines the effect of the GA in handling the defined undesired scenarios that may happen in agent interaction. The evaluation considers protocol completion with and without GA control, as well as environment status parameters that can give an indication to the overall performance of agents. This allows us to measure the effectiveness of the control process on successful protocol enactment.

The implementation environment for both case studies; Treasure Hunt and Auction, is based on Smart Python Agent Development Environment (Spade) library [33] for building a multiagent system, and the Snakes library [53] for implementing high level Petri nets.

The Treasure Hunt case study is based on the Dedale platform as mentioned previously. Its platform is based on the Jade framework and the Inter-platform Mobility Service which provides for distributed MAS. However, we chose to build the case study in Python; a reason for this is to maintain simplicity, and a number of elements were removed from the original environment, such as treasure types and the expertise required to pick treasure. Moreover, the creation of a scaled-down similar environment allows avoidance of complexity in integrating a Python-based based framework and focus on testing the Interaction Petri net model for capturing interactions between agents.

The Snakes library is a general Petri net library for creating and manipulating a variant of Python-Coloured Petri nets, where elements of a Petri net can be defined using arbitrary Python objects, expressions, and variables. It allows for the execution of Petri nets to explore all reachable markings, which makes it useful for exploring traces. In addition, a number of plug-ins are available that can handle extensions of Petri nets, such as Timed Petri nets and Hierarchical Petri nets.

The Spade library is a platform for building MAS, where the interaction of agents is based on Extensible Messaging and Presence Protocol (XMPP) and is FIPA compliant. It is chosen because it is built in Python and provides versatile agent behaviour. Agents can assume a Belief Desire Intention architecture as well as a simple reactive architecture with cyclic or periodic behaviour.

We use the ejabberd server for communication between Spade agents residing in a single computer. Ejabberd is an open, decentralized messaging service based on XMPP. It enables the switch to a decentralised interaction

through connecting with other Ejabberd servers or any XMPP compatible services.

4.1 Detection Evaluation

The IPN was implemented to register and verify initiated protocols and was tested on both case studies, the Treasure hunt and Auction.

4.1.1 Treasure Hunt

The environment is initiated with a map that indicates the placement of treasure and wells, which is fixed throughout the experiment. The start of each run includes three agents of each type: explorer, collector and tanker(total of nine) placed randomly within the grid. The grid environment limits observation for agents. Initially, each agent has no knowledge of other active agents, but as they explore they learn of other agents if they are close by. Interaction is also constrained with a transmission range of three grid node, that is any message to a receiver agent located more than three steps away from its sender will not reach.

The behaviour of GA is cyclic where each cycle involves executing the detection process. A single run is set to have a fixed number of GA cycles, and after the 150 cycle a run is terminated. The reason for this setting is that other agents vary in movement behaviour and may encounter possible termination within the environment.

At the end of each run, records of the detection process are collected. Every GA cycle executes the detection process once. Tokens within IPN are analyzed for determining any lost messages, IP delays and busy-wait scenarios. Lost messages are checked within every place within IPN, incomplete protocols are checked for delay and newly initiated protocols are checked whether the recipient agent is already engaged with an ongoing protocol. The results of each of those sub-processes are logged.

The experiment was run 100 times and each run completed 150 GA cycles. Table 4.1 illustrates a ten run sample of the detection results. The table includes the number of lost messages and delay detected in both Request Collector (RC) and Request Tanker (RT) protocols. Also, the number of busy-wait scenarios, and the total number of IPs initiated and completed from each protocol, the number of treasures picked by collector agents and terminated agents in each run.

Table 4.1: Treasure Hunt detection results

Lost Messages	IP RC	3	3	4	0	2	0	2	1	2	3
	IP RT	5	0	0	3	0	3	0	1	2	5
Delay	IP RC	2	1	0	0	1	0	0	0	0	1
	IP RT	2	0	0	0	0	1	0	0	0	0
Busy-Wait		1	0	0	0	0	0	0	0	0	0
IP Completion	IP RC	2/2	2/1	4/0	0/0	1/0	0/0	2/0	1/0	2/0	3/1
(total/completed)	IP RT	4/0	0/0	0/0	3/0	0/0	1/0	0/0	1/0	2/0	5/0
Treasure Picked		7	1	6	4	6	6	5	7	5	6
Terminated Agents		4	7	5	8	5	6	7	6	6	5

The arrival of a message depends on the transmission range and the position of the recipient. If the receiver agent is more than three steps away from the sender then transmission is out of range and the message does not reach its recipient. Another reason for failed delivery is the termination of an agent when reaching a well node. The number of lost messages varies depending on the number of IP's initiated and the movement of agents. The average of lost messages is 89.32% and the average of agent termination is 70%.

Delay is detected whenever an IP exceeds the expected time to complete. A delay in a RC protocol happens when a a collector agent is yet to reach the location to pick up the found treasure while the explorer agent stays idle waiting for a confirmation message. A delay in an RT protocol would cause a collector agent to stay put waiting for a tanker to reach its location and help with picked up treasure. Lost messages or termination contribute to the

delay in protocol completion.

A Busy-wait scenario happens when an IP is initiated with a participant that is already engaged in an ongoing protocol. This could arise in two cases: first a collector receives a request but is engaged in another instance of an RC protocol. Second is when a collector receives a request but is engaged with an RT protocol. Such scenarios could appear within instances of a single protocol as well as across different protocols.

The IPN model of agent interaction has captured the status of conversations between agents. The model has enabled the analysis of message tokens, which can belong to the RC and RT protocols, or they can be part of different instances of a protocol. On average, there have been a total of 2.35 IP instances initiated, with only 13% successfully completed.

A summary of the collected results is shown in 4.2. Overall, the detection experiment shows how lost messages, delay, busy-wait and termination have a negative effect on successful IP completion. High rates of terminated agents and lost messages are consistent with a low rate of completed IP's.

Table 4.2: Detection results summary of 100 runs

Average Lost Messages	Average Termination	Average Total IPs	Average IP Completion	Average Treasure Picked
89.32%	70.15%	2.35	13.11%	53.42%

4.1.2 Auction

The Auction environment is based on conducting an auction of a single item and includes one auctioneer agent and six participant agents. The detection process is applied for message transmission delay, which is an important factor within an auction protocol. The purpose of the experiment is to evaluate IPN in detecting message delay and investigate the effect of delay on agents participating in an auction.

All participants share the same parameters: the wallet amount, which is the maximum amount they are allowed to reach when proposing a bid, and

bidding strategy, which refers to the percentage increase they propose to bid. These parameters are fixed to ensure consistent behaviour and to allow us to consentrate on the influence of delay in a CFP message on participant bids.

The auctioneer starts the auction with a CFP call with a starting price of the item to be auctioned. When a bid proposed by a participant is received and accepted, the auctioneer initiates the next CFP round to the remaining participants with the new accepted price. CFP messages are sent out to participants in a random order.

A single run involves the completion of the auction. As with the Treasure Hunt, each GA cycle executes the detection process once. The goal is to detect CFP messages that are delayed. When a CFP message is received, a transition is fired which then consumes the message token in the sender place and produces the received message token in the receiver place supplemented with the time of arrival.

A preliminary task in the evaluation is identifying when delay occurs. A message delay can be determined using transmission time, which is the time a message takes to reach its destination. Within the experimental setting, transmission time is collected in order to obtain a meaningful delay threshold for detection. Transmission time is recorded over ten runs. The transmission time at the 80% percentile is set for delay detection, which means that messages with transmission times that fall within the highest 20% would be considered delayed.

A run is completed when the auction ends, that is when 10 seconds pass after the last received bid. Records of every CFP message are saved. A CFP delay log is kept and updated whenever a delay in a CFP message gets detected by the GA. Delay detection happens when a transition fires a CFP message token and the message arrival time exceeds the allocated transmission time. If a CFP message is detected for delay, the GA saves it for the control process.

Table 4.3 illustrates a sample of runs of the auction and details the number

of delays detected in sent out CFPs for each agent, the bids received by the auctioneer, and the acceptance percentage of those bids. The experiment is repeated for 50 auctions, at the end of each run, CFP and proposal message data are collected.

Table 4.3: Summary of results of ten runs showing the number of CFPs sent from the auctioneer to each participant, number of CFP delays detected, number of bids received by the auctioneer and acceptance rate of each participant. The winner of each run is highlighted in gray

Agent	CFP's sent from auctioneer	Delay Detected	Bids Received by auctioneer	Bid Acceptance
Participant 1	21	0	21	100.00
Participant 2	42	8	42	0.00
Participant 3	42	5	42	0.00
Participant 4	22	1	22	90.91
Participant 5	42	2	42	0.00
Participant 6	42	1	42	0.00
Participant 1	42	6	42	0.00
Participant 2	22	2	22	90.91
Participant 3	42	10	42	0.00
Participant 4	42	2	42	0.00
Participant 5	21	1	21	100.00
Participant 6	42	4	42	0.00
Participant 1	22	1	22	90.91
Participant 2	42	4	42	0.00
Participant 3	21	1	21	100.00
Participant 4	42	2	42	0.00
Participant 5	42	11	42	0.00
Participant 6	42	2	42	0.00
Participant 1	42	10	42	0.00
Participant 2	42	2	42	0.00
Participant 3	21	1	21	100.00
Participant 4	42	4	42	0.00
Participant 5	42	7	42	0.00
Participant 1	22	0	22	90.91
Participant 1	42	2	42	0.00
Participant 2	42	4	42	0.00
Participant 3	21	0	21	100.00
Participant 4	42	10	42	0.00
Participant 5	22	0	22	90.91
Participant 6	42	0	42	0.00

Agent	CFP's sent from auctioneer	Delay Detected	Bids Received by auctioneer	Bid Acceptance
Participant 1	21	0	21	100.00
Participant 2	42	4	42	0.00
Participant 3	22	2	22	90.91
Participant 4	42	4	42	0.00
Participant 5	42	6	42	0.00
Participant 6	42	13	42	0.00
Participant 1	42	1	42	0.00
Participant 2	22	0	22	90.91
Participant 3	21	1	21	100.00
Participant 4	42	9	42	0.00
Participant 5	42	0	42	0.00
Participant 6	42	6	42	0.00
Participant 1	42	2	42	0.00
Participant 2	22	1	22	90.91
Participant 3	42	6	42	0.00
Participant 4	42	10	42	0.00
Participant 5	42	1	42	0.00
Participant 6	21	0	21	100.00
Participant 1	42	23	42	0.00
Participant 2	22	2	22	90.91
Participant 3	42	11	42	0.00
Participant 4	42	7	42	0.00
Participant 5	42	18	42	0.00
Participant 6	21	1	21	100.00
Participant 1	42	11	42	0.00
Participant 2	22	1	22	90.91
Participant 3	42	3	42	0.00
Participant 4	42	6	42	0.00
Participant 5	42	2	42	0.00
Participant 6	21	0	21	100.00

To assess the effect of delay, a record of the acceptance percentage of an agent's bid proposals is collected. Results show that the acceptance rate of an agent is related to the number of delayed CFP messages it has received. At each auction, the agent with the highest acceptance rate has mostly recorded

a lower number of delayed CFP messages.

4.1.3 IPN Comparison Analysis

To further validate IPN, a comparison is made with a previous model by Gutnik and Kaminka [36], who proposed a scalable Petri net for conversation monitoring.

Their model was chosen based on a review of several PN models of interaction because it had a combination of features suitable for monitoring purposes. It allows identification of the status through PN places, content of messages, and coloured tokens, which enable the definition of multiple conversations.

The two models were assessed with respect to their design and performance. Considering a design perspective, we present how state-based model is different from agent-based in monitoring protocols. Performance of the two models is compared based on their processing time in detecting undesired scenarios within the implemented case studies.

Design

A key difference between the two is that an IPN is agent-based, where the place of a message token specifies the sender or receiver, whereas an SPN can be considered place-based, which means that the place of the message token specifies the current status of an IP. For SPN, tokens are checked in every place, where each token represents a conversation and its place represents the state of that conversation. To demonstrate, a FIPA Contract Net Interaction Protocol with one initiator and four participants is modeled using an IPN and an SPN, as illustrated in Figure 4.1.

SPN follows a structured flow of messages which means a sequential flow. This limits the monitoring process for flexible message arrival. For example, in the RC protocol shown in Figure 4.2, the arrival of an $InformDone$

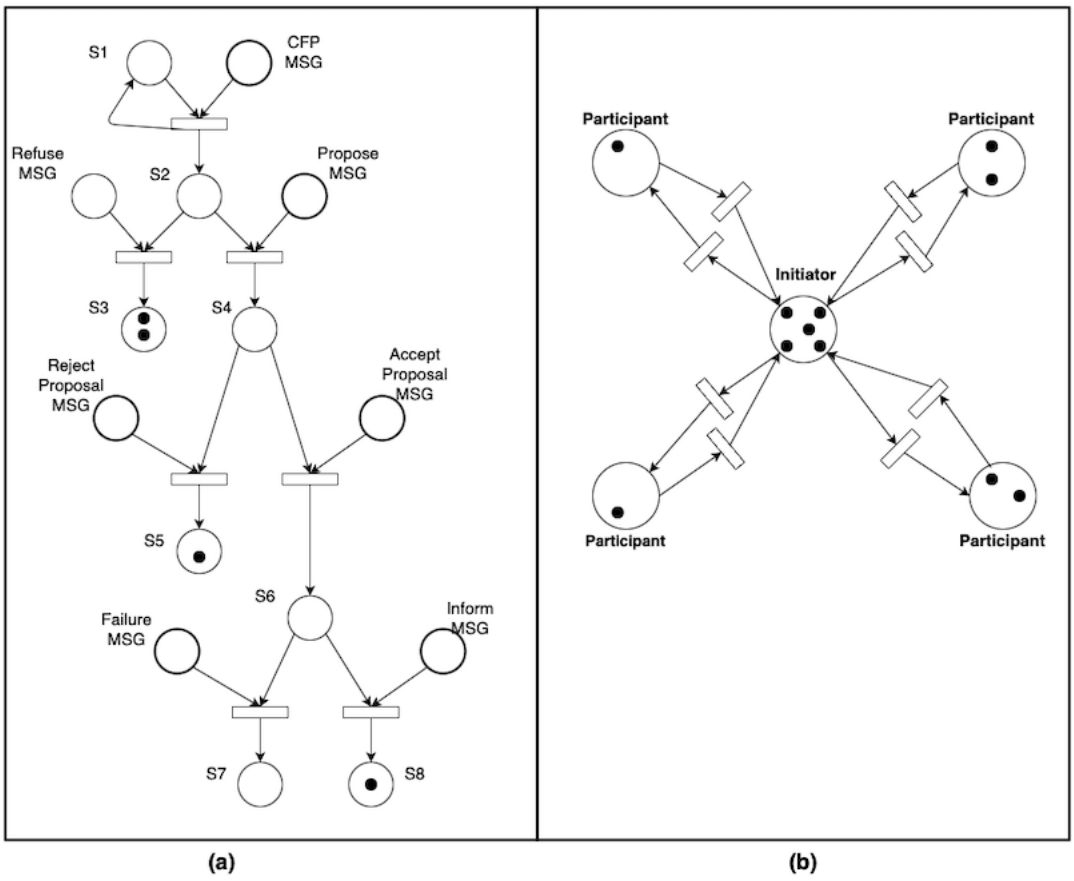

Figure 4.1: FIPA Contract Net Interaction protocol in an SPN(a), and in an IPN(b).

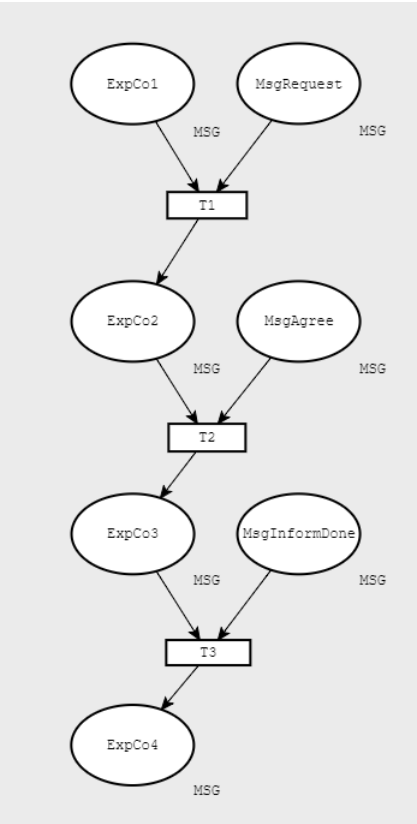

Figure 4.2: The SPN model of Request Collector Protocol

message cannot be modeled before the arrival of an *Agree* message. When a collector agent sends an $InformDone$ message, a token is added to the place $MsgInformDone$, and transition $T3$ is set to be fired when it is received by the explorer. There is the case of the transition not being enabled because the previous message token is not present at place $ExpCol3$. The reason could be a lost or delayed *Agree* message, which in turn makes SPN incapable of capturing the arrival of the final message of the protocol.

The second aspect to consider is a model's capacity to represent a large number of agents and complicated protocols. Since the SPN model is based on the IP structure and number of messages, the size of the Petri net is correlated with the size of the IP being modeled. However, the size of an

IPN is related to the number of agents present in the environment.

Moreover, an SPN models each IP with a single Petri net, so considering an environment with ten IPs for example, an IPN would be a single Petri net, but with an SPN it would be ten Petri nets. The task of adding a protocol with SPN is not as straightforward with IPN. An addition of a protocol with SPN requires the creation of Petri net to represent the roles and message flow of that protocol. IPN in contrast, does not require any update when new protocols are added.

Overall, the choice between an agent-based analysis and a place-based analysis can relate to the perspective needed most within a particular case study. The IPN presents an agent view of every conversation an agent is involved with, which provides a more holistic view of conversation instances of every protocol.

Performance

By performance of an IP monitoring model we mean the model's ability to detect errors in the least amount of time possible. For comparison and experimental evaluation, a process for error detection is proposed here to be applied to this Scalable Petri net (SPN). The detection experiment was run with both models IPN and SPN. For every run, detection functions of SPN and IPN were executed the same number of times. The comparison takes into account detection of lost messages, IP delay, busy-wait and message delay.

In terms of the type of errors that can be detected, SPN and IPN-property models enabled the detection of all four types of errors considered here within an IP. However, the detection of agent termination is not applicable in the SPN net since the detection process is based on the status of messages while in IPN a place represents an agent and provides a direct way of updating the status of the aliveness of an agent through place labels. To include agent status in SPN, the message structure should be extended with metadata on agent status.

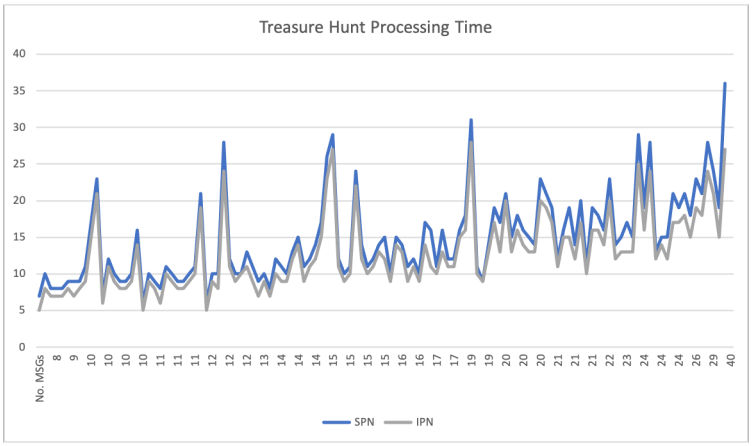

Figure 4.3: Processing time of SPN and IPN in detection of Lost message, Busy-wait, and IP delay in the Treasure hunt case study

Processing time of each process was collected from the same trace and was repeated 100 times. The time a process takes to find errors can be measured with a code profiling tool that measures processing time. Results showed that SPN and IPN are similar in the time consumed in analysing the status of conversations. Figure 4.3 and 4.4 illustrate the processing time of both models in the first and second environments, respectively.

In the first case study, IPN was slightly faster with an average of 12.83 ms while SPN average was 14.73 ms. Figure 4.3 presents the processing time in relation to the number of messages.

The process of detecting lost messages in IPN and SPN is quite similar. The process involves verifying tokens at each place of the Petri net. For the first case study, SPN contains a total of 8 places while IPN contains a total of 9 places which is the number of agents. The number of tokens that are within the places of SPN and IPN represents the number of messages created and sent. To consider the difference in time complexity of each process, we suppose that the process deals with n places and m tokens within those places which evaluates to a time complexity of $O(nm)$ for both IPN and SPN. The number of places and tokens effects the processing time of detection; a higher

number of places or messages within a place would increase processing time.

IP delay detection is also similar in both models, because the time is recorded when a new token is added. However, a noted difference is in the Busy-wait detection in the first case study. SPN requires two nets to detect cross protocol engagement of participants, while in IPN the recipient place is analyzed for ongoing protocols that could be instances of the same or a different protocol. Consequently, time complexity for this type of scenario is different, SPN would be $O(m1 + m2)$ where $m1$ and $m2$ refers to the initial places of IPs 1 and 2. IPN would have better time complexity with $O(m)$ where m is the requested agent place.

In the second case study, a single protocol is implemented and the detection process is concerned with transmission delay. The process of detecting transmission delay in both models shows similar times with an average of 1.9 ms for SPN and 1.4 ms for IPN, shown in Figure 4.4. The reason is that the detection process is applied in the same approach for both SPN and IPN. A message is checked for transmission delay when it arrives which is when a transition is fired.

Our objective of adopting a Petri net model of interaction is to facilitate capturing the status of multiple conversations and to enable error detection in real-time. Although both IPN and SPN recorded similar processing time in detecting lost messages, the IPN process of detection has better performance in handling multiple protocols simultaneously.

4.2 Control Evaluation

The evaluation of the control component of the governing agent involves a comparison of execution in two scenarios: first in normal execution, second with the application of GA control. This allows us to compare the execution of the case study system and to test whether agent interaction is improved with intervention from the governing agent.

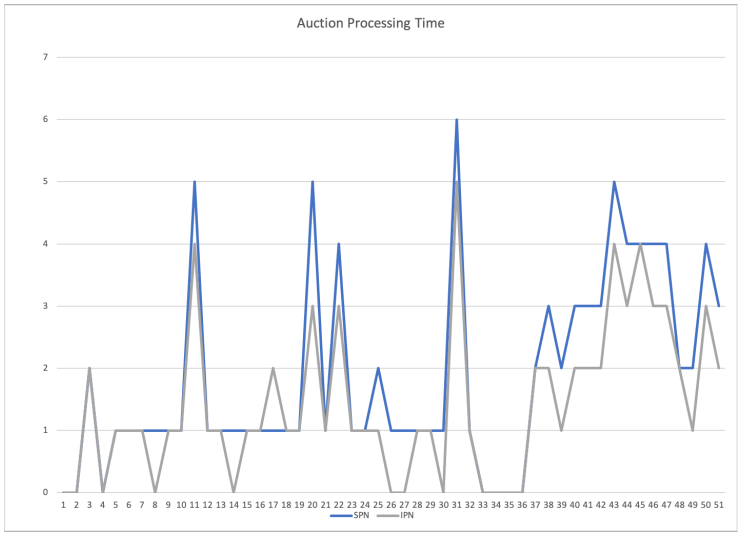

Figure 4.4: Processing time of SPN and IPN in detection of message delay in the Auction case study

4.2.1 Treasure Hunt

The main objective is to improve system performance by minimising interaction mistakes. For the Treasure Hunt case study, optimal performance means collecting all available treasure. For this reason, the amount of collected treasure is a primary performance indicator. Moreover, completed interaction protocols are essential for collaboration and achieving the agent's goal of collecting treasure. We also consider the number of terminated agents because it effects successful protocol enactment.

The control experiment was run with same environment parameters as the detection experiment. The environment is initiated with the same grid map and each run is set to have 150 GA cycles.

Application of the framework improved the performance of agents with an increased number of collected treasures. Table 4.4 show the performance indicators of the GA scenarios.

A summary of collected results is shown in Table 4.5, where an improvement in the numbers of collected treasures is observed with GA control. The

Table 4.4: Treasure Hunt control results

Lost Messages	IP RC	7	6	5	4	5	12	1	4	6	9
	IP RT	0	6	3	0	10	11	0	6	3	7
Delay	IP RC	3	3	2	1	3	9	1	2	2	3
	IP RT	0	2	1	0	4	3	1	2	1	2
Busy-Wait		0	0	1	0	1	0	0	0	0	1
IP Completion	IP RC	3/1	3/2	3/2	2/1	4/1	9/4	1/1	3/2	3/2	5/2
(total/completed)	IP RT	0/0	4/1	1/1	0/0	5/5	3/3	1/1	2/2	1/2	4/3
Treasure Picked		9	8	8	7	7	9	9	9	9	9
Terminated Agents		5/11	6/11	6/12	7/12	4/11	3 /10	6/11	5/11	5/11	6/12

average percentage of treasure collected in the non-GA control scenario is 53.42% shown in 4.2, compared to the average of 87.11% with GA control. This increase is due to two factors: First, the number of completed IPs means a higher number of complete conversations of Request Collector and Request Tanker, which enabled agents to collaborate successfully in treasure collection. Second, the average number of active agents is higher with GA control, which contributes to collecting more treasure.

Table 4.5: Control results summary of 100 runs

Average Lost Messages	Average Termination	Average Total IPs	Average IP Completion	Average Treasure Picked
77.35%	55.22%	6.69	51.66%	87.11%

Termination of agents affects the overall behaviour of agents. When an agent terminates a warning message is sent from the terminated agent to its set of known agents indicating its location for other agents to avoid. Moreover, the GA control action defined for termination first includes a warning message to all active agents, and second includes the addition of a new agent of the same type as the terminated one. The addition of new agents is limited to one per type so as not to crowd the environment.

With GA control, lost messages are forwarded by the GA, which contributes to the higher rate of completed IPs with control, in contrast to the rate of completed IPs without control, as shown in Figure 4.5

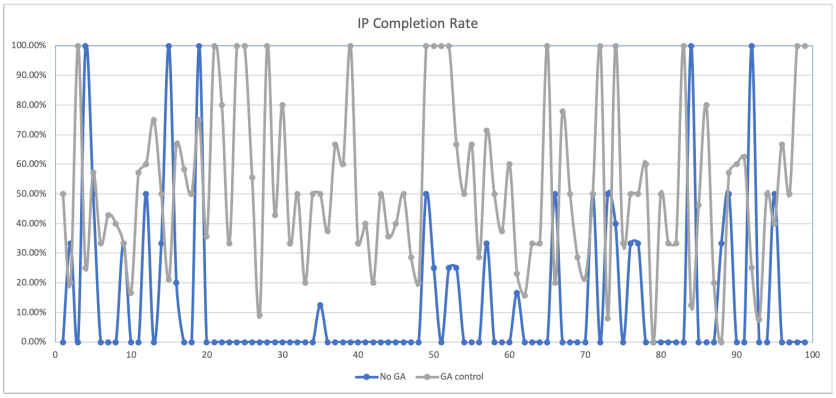

Figure 4.5: Rates of completed IPs with and without GA control.

Each scenario starts with nine agents; however, the total number of agents in the control scenario is dynamic and can increase up to 12 agents depending on the number of times a termination control action is taken. The results show an improvement in the number of terminated agents, where fewer agents are terminated, as shown in Figure 4.6. Termination control, in addition to lost messages control action, has helped agents avoid termination.

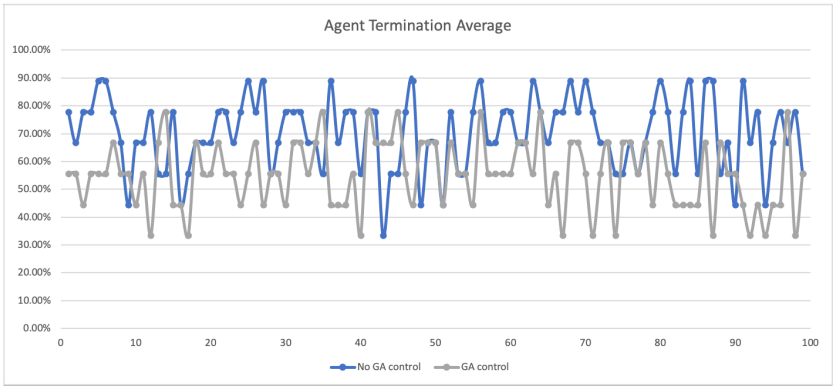

Figure 4.6: Rates of terminated agents with and without GA control.

4.2.2 Auction

In the Auction case study, the experiment is run using the proposed control process defined in Chapter 3. The same records are collected as well as a count of the governing agent control action of processed delays, as shown in Table 4.6.

The average number of CFPs has increased due to the delay control process. The delay control rolls back to a previous price point, which in turn increases the number of CFP rounds; for example, in the first run, CFPs are around 200 CFPs per participant with the control action compared to 42 CFPs without the control action. A CFP-wait round is introduced when a delay occurs where there is time window to collect incoming bids before one is accepted.

Table 4.6: Summary of results of ten runs with GA control showing the number of CFPs sent from the auctioneer, number of CFP delays detected, number of processed delays, number of bids received by the auctioneer and acceptance rate of each participant. The winner of each run is highlighted in gray

Agent	CFP's sent from auctioneer	Delay Detected	Delay Processed	Bids Received by auctioneer	Bid Acceptance
Participant 1	254	168	7	216	5.09
Participant 2	258	153	8	220	3.18
Participant 3	169	63	5	147	65.31
Participant 4	183	84	3	167	49.10
Participant 5	251	168	8	213	6.57
Participant 6	251	171	9	213	6.57
Participant 1	163	73	3	147	48.98
Participant 2	221	135	10	190	7.37
Participant 3	226	149	3	195	4.62
Participant 4	224	138	9	193	5.70
Participant 5	225	151	6	194	5.15
Participant 6	151	46	3	136	61.76
Participant 1	236	150	8	206	3.40
Participant 2	163	70	2	150	53.33
Participant 3	236	141	6	206	3.40
Participant 4	154	57	2	137	64.96
Participant 5	230	158	11	200	6.50
Participant 6	230	153	4	200	6.50
Participant 1	234	159	12	200	3.50
Participant 2	228	140	6	194	6.70
Participant 3	225	139	7	191	8.38
Participant 4	170	87	0	155	45.81
Participant 5	160	55	3	141	57.45
Participant 6	226	143	9	192	7.81
Participant 1	175	61	3	155	58.71
Participant 2	181	84	2	165	51.52
Participant 3	248	148	10	212	8.49
Participant 4	255	151	10	219	5.02
Participant 5	255	176	6	219	5.02
Participant 6	256	173	8	220	4.55

Agent	CFP's sent from auctioneer	Delay Detected	Delay Processed	Bids Received by auctioneer	Bid Acceptance
Participant 1	221	139	6	192	4.69
Participant 2	217	142	6	188	6.91
Participant 3	145	58	4	129	65.89
Participant 4	223	143	7	194	3.61
Participant 5	157	68	3	144	50.69
Participant 6	220	148	6	191	5.24
Participant 1	226	153	4	194	7.73
Participant 2	231	155	5	199	5.03
Participant 3	157	52	3	138	60.87
Participant 4	231	143	9	199	5.03
Participant 5	228	145	8	196	6.63
Participant 6	168	78	6	155	47.10
Participant 1	168	76	3	150	51.33
Participant 2	230	159	7	198	7.58
Participant 3	236	143	10	204	4.41
Participant 4	235	160	5	203	4.93
Participant 5	235	145	6	203	4.93
Participant 6	156	60	3	142	62.68
Participant 1	148	53	2	135	67.41
Participant 2	235	142	6	205	1.95
Participant 3	229	148	6	199	5.03
Participant 4	166	74	3	149	48.99
Participant 5	230	149	6	200	4.50
Participant 6	221	154	10	191	9.42
Participant 1	227	154	9	197	3.55
Participant 2	230	155	5	199	2.01
Participant 3	158	76	4	142	53.52
Participant 4	151	54	3	134	61.94
Participant 5	217	123	6	187	9.09
Participant 6	220	132	5	189	7.41

The negative correlation between bid acceptance and the number of delays is still observed with GA control. However, an increase in a participant's chance of winning is found despite a high number of CFP delays. This is because the CFP-wait round has allowed bids from every participant to be considered.

Furthermore, an agent's chance of winning an auction has no visible correlation with its acceptance rate, in contrast to without GA control. In the detection experiment, the fastest bid to reach the auctioneer is likely to be accepted due to the fixed parameters of the internal bidding behaviour of participant agents. For the control experiment, bids from all participants are

considered in the CFP-wait round. Because the fixed bidding strategy means that all bidders will bid the same amount, the winner is randomly chosen. Of course if there are different bidding strategies in use, which would be the case in real-life, the highest bidder would win the round.

To examine the relationship between a CFP delay and bid acceptance, an analysis is made of each bid round. After applying the control action of CFP-wait rounds, total bids are collected along with information on CFP delay and outcome.

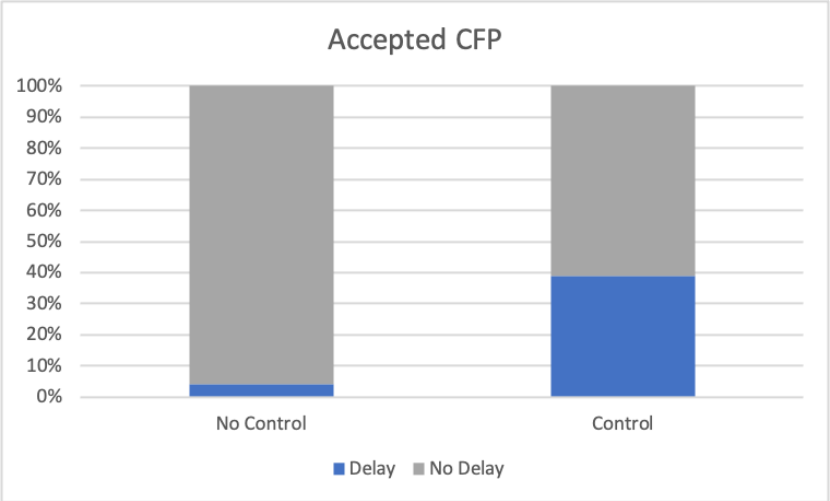

Figure 4.7: Delay rates in **Accepted** CFPs with no GA control

For accepted CFPs, a CFP delay has a negative affect on the outcome of a bid. For each run with no GA control, accepted CFP rounds show a lower delay percentage, as shown in Figure 4.7, compared with the rejected CFPs shown in Figure 4.8. With GA control there is a clear increase in accepted bids with delayed CFPs.

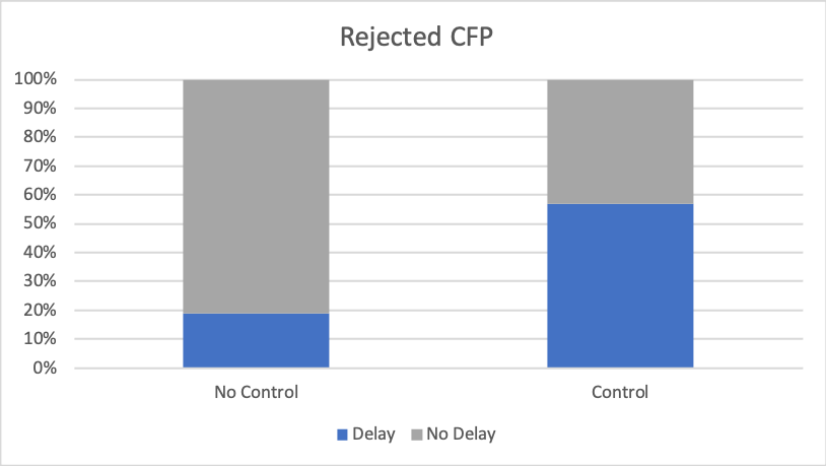

Figure 4.8: Delay rates in **Rejected** CFPs with no GA control

4.3 Summary

This chapter has examined the effectiveness of the framework design in detecting and correcting communication issues through two MAS case studies involving heterogeneous agents interacting in an open and asynchronous environment. The first case study is adopted from the Dedale test-bed with the addition of three interaction protocols for agents to collaborate on the objective of treasure collection. The second case study is an auction with multiple participants, which is a commonly applied MAS protocol.

The IPN approach was found to be successful in detecting defined undesired scenarios using the proposed detection processes. Message tokens are analysed within each agent place of the Petri net to detect scenarios. The detection experiment results show how the defined scenarios of lost message, IP delay, busy-wait, and agent termination have impact on successful completion of IP conversations in the first case study. In the second case study, message delays have reduced participants chance in having their bids accepted.

To further evaluate the IPN, a comparison was made with the state-based

SPN model. Both models successfully detect scenarios of IP delay, busy-wait, lost messages, and transmission delay. The IPN also detects termination of agents through place labels. The time efficiency of IPN and SPN were found to be similar. However, when it is the case of an agent role participating in two protocols, IPN was found to have better time complexity. Another limitation of SPN is modeling flexible message arrival. A scenario where messages of a protocol arrive in a different order cannot be modeled.

The control evaluation applied the GA behaviour presented in Chapter 3 for each of the case studies. For every scenario a control action has been selected based on the case study environment. The GA control could be a simple information update to the agents or a modification in the environment such as introducing an additional agent. Results of GA control in the Treasure hunt environment have shown improved performance and an increase in successful protocol enactment. In the Auction environment, GA control has minimized the negative effect of transmission delay on participating agents.

Limitations of the proposed approach are twofold: First, for the GA control to be effective it needs to be designed by the MAS designer. While common undesired scenarios are defined for detection within the IPN model, controlling the effect of those scenarios is context dependant. For example, in the Auction case study, the control mechanism was designed to handle the fixed agent bidding strategy. In a realistic setting, bidding strategies are different therefore an appropriate control action is needed to better handle this parameter. Different control mechanisms for varied bidding strategies would be interesting to explore in future work.

The second limitation is that the approach assumes successful delivery of GA messages. Any GA information updates are guaranteed to reach the recipient agents. The GA is considered as an environment entity, one way to avoid this limitation would be through only controlling environment parameters.

Chapter 5

Distributed IPN

The proposed interaction assurance approach was presented and evaluated in Chapters 3 and 4, respectively. The approach, which constitutes two parts (the Petri net model and a governing agent) has been shown to detect undesired scenarios of communication and minimise their negative effects. There are two key features to the approach: first, it defines general issues of communication to be detected within every interaction of agents, and second, it defines a structure for governing interaction where control actions can be tailored to the specification of the target environment.

Despite this, the experimental evaluation of the approach has been limited to a local context where agents involved were operating within the same machine. As a multi-agent system is inherently distributed, it is important to employ and evaluate the approach using distributed agents.

The application of distributed monitoring and control involves the challenge of tracking communication across different locations. It is essential for a governing agent to obtain full oversight of the status of communication across distributed nodes. To address this, the task of monitoring would be distributed over a number of governing agents. Each GA would monitor local interaction and receive monitoring updates from other governing agents of their local information.

The IPN model is key to monitoring and detection. With distributed governing agents, the IPN would be replicated within each node/place of the environment. Each GA will then maintain an IPN model and would capture the status of communication locally. This will distribute the task of monitoring interaction between governing agents.

This chapter presents a distributed design of the proposed approach to be applied.

5.1 Introduction

The goal of this chapter is to examine distributed detection and control with an IPN and governing agents. The approach would extend to support distributed monitoring, which requires the tracking of sent and received messages across nodes.

A multiagent system is composed of a group of agents that communicate to achieve a specific goal. Typically, MAS is decentralised, and there is no single entity for making decisions about the system, and management decisions are distributed among agents. To create a distributed MAS, a peer-to-peer infrastructure is adopted for its environment. Each node or agent can directly communicate with other agents.

A governing agent would be located based on the distribution of agents. The purpose of placing a GA is to set a monitoring entity in such a way that message events can be recorded directly, so that messages received and sent by close agents are tracked by the GA and updated within the IPN model. As the model consists of places that represent agents, a distinction would be made between local and non-local agent places.

On the other hand, tracking of message events of non-local agents is handled indirectly. To capture message status of non-local places, marking transfer is carried between agents. A marking is the placement of tokens in Petri net places, which reflect the status of messages for each agent. Dis-

tributed GAs can share the local status of conversation by sending over a marking of the local places in the IPN.

5.2 Distributed Approach

The distributed design approach involves multiple instances of a governing agent. The placement of a GA is taken from the concept of a super-peer in a super-peer network. A super-peer is a special node created to act as a server to a group of client nodes. In a similar manner, a single GA is linked with agents that are located within the same node or a close location, as depicted in Figure 5.1. The GA can be seen as a supervisor over that group of agents with the task of detection and control within that node.

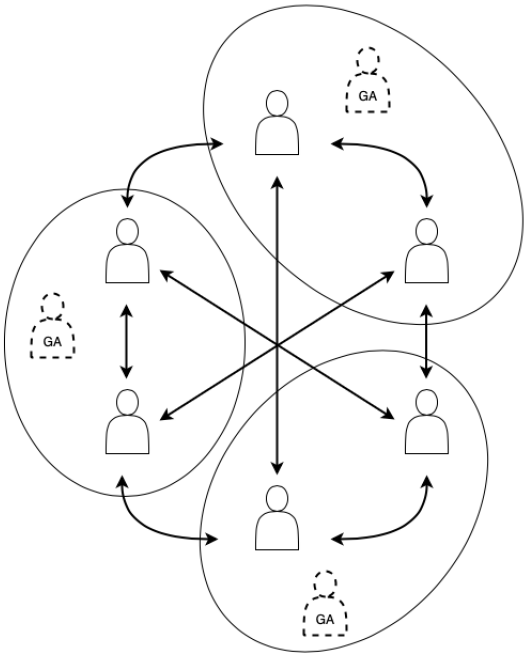

Figure 5.1: Peer-to-peer architecture with GA

It is required that a governing agent's identity be known by other gov-

erning agents, since the task of monitoring is distributed over GAs. When a governing agent is created, it is initialised to connect with other agents.

Each GA will hold an IPN for monitoring and communication analysis. The monitoring process tracks two message events: sending a message and receiving a message. Within a distributed environment, a GA tracking of message events would be limited to those happening within its node; thus, GA observation is partial. An IPN would represent incomplete status of interaction and hold a marking that represents local interaction only. As a result, IPN instances in different nodes would have different markings.

To ensure consistency between IPN markings, a marking update process is introduced. Figure 5.2 provides an overview of the process, where a GA sends its current marking of local places to other GA agents. Once a GA receives a marking, it updates its non-local places accordingly. The update includes two steps: first, it simply replaces the current marking of non-local places with the received marking, and second, it eliminates redundant tokens.

The first step involves updating the marking of non-local places. Based on the received marking, the GA selects the places given and removes all tokens, then simply adds the received marking. Adding a marking to a Petri net will add each set of tokens to its corresponding place.

The second step involves checking for redundant tokens. The local places hold message tokens that are sent to non-local places; however, the event of receiving those message and firing the transition is not reflected locally. As a result, a message token could be present in both a sender and receiver place. To resolve this issue, the GA checks the local marking for message tokens that have been received and removes them from the IPN marking.

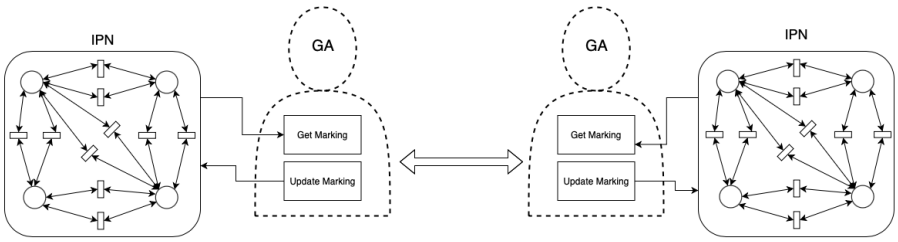

Figure 5.2: Marking update

Once the marking update is completed, the GA detection and control process is initiated. The remit of each GA's control actions extends only to agents that are local to it.

To demonstrate and evaluate the distributed approach, the Treasure Hunt and Auction case studies are used. In the section that follows, a description of the implemented case studies and experimental results are presented.

5.3 Evaluation

The objective is to evaluate the detection and control approach in a distributed environment. The experiments presented in Chapter 4 are now implemented to operate in two different machines.

5.3.1 Treasure Hunt

The treasure hunt environment is distributed between two different locations. The grid is divided in two parts, where one part is on one machine and the other is on a different machine. Agents can move around the grid in a similar manner and can transfer from one part of the environment to the other.

The case study properties of message transmission, agent placement, treasure and well locations remain the same. Agent transmission range, behaviour and interaction protocols are also not changed. The only addition is a migration entity, implemented to handle agent movement between the two

parts of environment. The migration agent handles the process of transfer of an agent where the agent address is updated to reflect its location.

Communication between agents is constrained by the transmission range, which means that reachable agents are in the same part of the environment.

The first part of experiment is run without GA control. Detection is concerned with the four undesired scenarios: IP delay, lost messages, busy-wait, and agent termination. The experiment is conducted with 100 runs, and Table 5.1 show the results of IP detection from a sample of the runs. The detection process takes as input the IPN marking. In the distributed setting an IPN marking represents the status of conversations of agents in both nodes of the environment.

Table 5.1: Treasure Hunt detection results

Lost Messages	IP RC	2	0	1	2	1	3	1	2	1	2
	IP RT	0	2	0	0	3	0	0	2	0	3
Delay	IP RC	1	0	0	0	0	1	1	2	0	1
	IP RT	0	0	0	0	0	0	0	0	0	0
Busy-Wait		0	0	0	0	0	0	0	0	0	0
IP Completetion	IP RC	1/1	0/0	1/1	2/0	1/0	2/1	2/1	4/2	2/1	1/1
(total/completed)	IP RT	0/0	2/0	0/0	0/0	3/0	0/0	0/0	2/0	0/0	2/0
Treasure Picked		7	6	5	4	6	3	2	6	5	6
Terminated Agents		7/9	7/9	7/9	6/9	8/9	5 /9	7/9	6/9	6/9	6/9

A summary of the collected results is shown in 5.2. Similar to results found in the local experiment, the average termination and lost messages are high with low rates of IP completion. On average, there have been a total of 1.5 IP instances initiated, with only 24% successfully completed.

Table 5.2: Detection results summary of 100 runs

Average Lost Messages	Average Termination	Average Total IPs	Average IP Completion	Average Treasure Picked
84%	69%	1.5	24%	53%

The second experiment involves applying the complete approach of detection and control. The GA control action, described in Section 3.3.1, is the

same as the control applied in the local experiment presented in the previous chapter. A ten run sample is presented in Table 5.3, with records of the number of lost messages, IP delay and busy-wait scenarios detected.

Table 5.3: Treasure Hunt control results

Lost Messages	IP RC	2	5	9	2	6	10	4	2	9	5
	IP RT	0	0	3	0	0	0	0	6	0	3
Delay	IP RC	1	2	3	1	2	4	2	1	3	1
	IP RT	0	0	1	0	0	0	0	1	0	1
Busy-Wait		0	0	0	0	0	5	0	0	2	1
IP Completetion	IP RC	1/1	5/3	8/3	2/1	4/3	14/6	5/4	1/1	6/2	6/1
(total/completed)	IP RT	0/0	0/0	1/1	0/0	1/0	0/0	0/0	3/1	0/0	1/1
Treasure Picked		9	7	8	8	6	8	6	8	7	8
Terminated Agents		5/11	5/12	6/11	7/11	6/12	4/12	5/11	7 /10	4/11	3/11

A summary of the total runs is found in Table 5.4. The average number of initiated protocols has increased from one IP per run to four protocols. One reason for this increase is the updates received by agents from the GA on busy agents. An explorer will initiate new RC protocols if it receives the information that the only agent that replied to its first request is busy.

The rate of completed protocols recorded is 32%, a slight increase from 24% completion rate without GA control. This increase can be because of the control on termination and lost messages. Termination control involves creating a new agent in replacement of a terminated one and informing other agents of terminated ones. And detected lost messages are forwarded to their recipient.

Table 5.4: Control results summary of 100 runs

Average Lost Messages	Average Termination	Average Total IPs	Average IP Completion	Average Treasure Picked
53%	44%	4.3	32%	78%

5.3.2 Auction

The Auction case study is evaluated here in a distributed setting where agents operate from different locations. As in the auction experiment in Chapter 3, the auction involves a single item, one auctioneer agent, and six participant agents. The participant agents are placed within the same location while the auctioneer is run from a different one. This placement is made to examine the effect of message transmission delay on participant bidding outcomes in the protocol.

With each CFP call, agents determine their bids and send out proposals to the auctioneer. Each proposal is checked against the current price and has two possible outcomes. A higher bid is accepted, the current price is updated, and the bidding agent is informed of acceptance while a new CFP is sent out to the remaining agents. A lower bid is rejected, and the bidding agent is informed of rejection.

The running of a single auction constitutes a run in the experiment, and there were a total of 50 runs. Before the start of the experiment, the transmission time is calculated within the distributed settings because establishing the average time it takes to send and receive a message is a preliminary step before setting the delay threshold.

An objective of the auction case study is to evaluate IPN in monitoring a protocol with multiple role instances by the participants. Another goal is to assess the scenario of message transmission delay and its effect on interacting agents.

The detection of message delay involves checking the time it takes a message to arrive and relies on temporal information recorded in tokens. Message tokens contain times of message emission and arrival. Message arrival time is recorded in the IPN when an agent receives a message; IPN fires a transition based on the received message information and records the time.

Table 5.5 illustrates the numbers collected from ten runs. At the end of each auction, we collect the number of CFPs sent out by the auctioneer,

number of delays detected within those CFP's, the number of bids received by the auctioneer, and lastly the acceptance percentage of those bids.

The acceptance rate of an agent is affected by the number of delayed CFP messages it has received. At each auction, the agent with the lowest number of delays received mostly a higher acceptance rate. Similar to the local experiment, there is an observed high variance in acceptance rates of 44. The is caused by the delay in receiving a CFP as well as the shared bidding strategy.

There does not seem to be a relationship between an agent's chance of winning an auction and the number of delays detected, but it is often the case, though not always, that the winning agent has a lower number of delays.

Table 5.5: Summary of results of ten runs showing the number of CFPs sent from the auctioneer, number of CFP delays detected, number of bids received by the auctioneer and acceptance rate of each participant. The winner of each run is highlighted in gray

Agent	CFP's sent from auctioneer	Delay Detected	Bids Received by auctioneer	Bid Acceptance
Participant 1	37	5	37	13.5
Participant 2	35	6	35	20.0
Participant 3	37	6	37	13.5
Participant 4	30	2	30	40.0
Participant 5	37	7	37	13.5
Participant 6	35	7	35	20.0
Participant 1	32	1	32	31.3
Participant 2	35	0	35	20.0
Participant 3	34	0	34	23.5
Participant 4	35	1	35	20.0
Participant 5	39	2	39	7.7
Participant 6	36	2	36	16.7
Participant 1	34	6	34	23.5
Participant 2	33	5	33	27.3
Participant 3	36	10	36	16.7
Participant 4	34	6	34	23.5
Participant 5	34	8	34	23.5
Participant 6	40	8	40	5
Participant 1	35	8	35	20.0
Participant 2	38	6	38	10.5
Participant 3	34	3	34	23.5
Participant 4	37	5	37	13.5
Participant 5	32	2	32	31.3
Participant 1	35	2	35	20.0
Participant 1	31	31	0	35.5
Participant 2	37	37	3	13.5
Participant 3	36	36	2	16.7
Participant 4	33	33	3	27.3
Participant 5	37	37	2	13.5
Participant 6	37	37	2	13.5

Agent	CFP's sent from auctioneer	Delay Detected	Bids Received by auctioneer	Bid Acceptance
Participant 1	33	0	33	27.3
Participant 2	33	3	33	27.3
Participant 3	35	1	35	20.0
Participant 4	37	1	37	13.5
Participant 5	37	4	37	13.5
Participant 6	36	4	36	16.7
Participant 1	33	13	33	27.27
Participant 2	38	19	38	10.53
Participant 3	36	13	36	16.67
Participant 4	33	8	33	27.27
Participant 5	37	12	37	13.51
Participant 6	34	12	34	23.53
Participant 1	34	1	34	23.53
Participant 2	39	8	39	7.69
Participant 3	36	6	36	16.67
Participant 4	33	3	33	27.27
Participant 5	35	4	35	20.00
Participant 6	34	4	34	23.53
Participant 1	32	3	32	31.25
Participant 2	35	3	35	20.00
Participant 3	35	6	35	20.00
Participant 4	34	7	34	23.53
Participant 5	38	4	38	10.53
Participant 6	37	4	37	13.51
Participant 1	37	17	37	13.51
Participant 2	33	6	33	27.27
Participant 3	38	10	38	10.53
Participant 4	34	9	34	23.53
Participant 5	34	9	34	23.53
Participant 6	35	11	35	20.00

The experiment is run after implementing the proposed control process defined in Chapter 3. Results were collected from 50 runs, Table 5.6 illustrates results from a sample of those runs. The same data is collected as the detection experiment with addition to the number of delays processed. These refer to the number of times an auctioneer received a notification of delay and initiated a CFP-wait round.

As was the case in the local setting, the average number of CFPs has increased due to the delay control process. The number of CFPs sent out

to each participant has reached over 100 CFPs in comparison to around 30. The variance in the number of CFPs to each participant is a result of the protocol flow, when a participant bid is accepted it receives an accept and is excluded from the next CFP round.

Table 5.6: Summary of results of ten runs with GA control showing the number of CFPs sent from the auctioneer, number of CFP delays detected, number of processed delays, number of bids received by the auctioneer and acceptance rate of each participant. The winner of each run is highlighted in gray

Agent	CFP's sent from auctioneer	Delay Detected	Delay Processed	Bids Received by auctioneer	Bid Acceptance
Participant 1	162	16	7	149	17.45
Participant 2	161	10	3	145	18.62
Participant 3	153	8	2	138	25.36
Participant 4	163	13	2	149	16.78
Participant 5	158	6	1	145	20.69
Participant 6	159	6	0	145	20.00
Participant 1	144	32	3	132	24.24
Participant 2	140	39	3	123	29.27
Participant 3	151	36	1	137	18.25
Participant 4	150	46	3	137	18.98
Participant 5	155	40	3	141	14.89
Participant 6	157	36	3	143	13.29
Participant 1	123	9	2	113	19.47
Participant 2	116	16	1	106	27.36
Participant 3	124	10	3	115	18.26
Participant 4	127	10	1	113	15.93
Participant 5	130	9	3	118	12.71
Participant 6	118	8	2	109	24.77
Participant 1	109	4	2	100	26.00
Participant 2	116	7	2	107	17.76
Participant 3	113	6	1	104	21.15
Participant 4	114	5	2	102	20.59
Participant 5	119	6	2	108	14.81
Participant 6	116	5	2	106	17.92
Participant 1	102	6	4	93	16.13
Participant 2	100	5	1	89	19.10
Participant 3	103	3	0	94	14.89
Participant 4	98	6	1	88	21.59
Participant 5	94	6	2	87	26.44
Participant 6	100	9	3	88	19.32

Agent	CFP's sent from auctioneer	Delay Detected	Delay Processed	Bids Received by auctioneer	Bid Acceptance
Participant 1	84	1	0	74	16.22
Participant 2	84	4	1	75	16.00
Participant 3	79	2	0	72	23.61
Participant 4	80	6	3	70	22.86
Participant 5	78	7	4	70	25.71
Participant 6	86	5	2	77	12.99
Participant 1	125	10	2	114	15.79
Participant 2	126	20	3	109	15.60
Participant 3	123	6	0	112	17.86
Participant 4	115	13	4	101	27.72
Participant 5	116	14	2	103	26.21
Participant 6	125	13	3	110	16.36
Participant 1	141	25	3	122	18.03
Participant 2	135	21	1	120	23.33
Participant 3	138	21	3	125	20.00
Participant 4	143	30	3	128	15.63
Participant 5	133	21	2	118	25.42
Participant 6	141	22	3	126	17.46
Participant 1	114	8	3	102	13.73
Participant 2	109	6	2	96	19.79
Participant 3	115	5	3	104	12.50
Participant 4	110	7	4	98	18.37
Participant 5	100	2	0	90	31.11
Participant 6	106	4	1	97	22.68
Participant 1	143	24	1	123	15.45
Participant 2	135	23	1	121	22.31
Participant 3	138	21	2	123	19.51
Participant 4	135	19	6	120	22.50
Participant 5	136	24	3	116	22.41
Participant 6	140	33	3	123	17.89

The relationship between bid acceptance and the number of delays has changed. The participant with a lower bid acceptance is not necessarily the one with the highest delay count. Furthermore, an agent's chance of winning an auction has no visible correlation with its acceptance rate, as was the case with no GA control. A possible reason for this is the fixed parameters of the internal bidding behaviour of participant agents.

A main outcome of the control action is the reduction in variance between acceptance rates among bidders. Since bidders share the same strategy, the effect of delay without GA control is to give high variance in acceptance rates, which indicates that delayed CFPs have caused lower acceptance rates. The variance with GA control is reduced to 18. This improvement is because the control action allowed participants to have a better chance of acceptance even with a CFP delay.

An important aspect of the application of a distributed IPN is maintaining consistency of markings among the different instances of IPN. The marking update process enables up to date tracking of message events across nodes. At the end of each run a comparison between IPN markings of both nodes is done. Comparison results have shown that markings are equal at every run.

5.4 Summary

This chapter has presented a distributed design and evaluation of the proposed framework of detection and control. To enable distributed monitoring of agents, a GA is placed at each node in a peer-to-peer architecture where a local IPN is maintained. To handle differences of IPN markings, the detection process is extended with a preliminary process of marking update. Such a process provides a GA with a complete up-to-date status of agent interaction.

Two case studies are evaluated for the detection and control of the five undesired scenarios. Results from the Treasure hunt case study showed an improvement in completed IPs with GA control. Furthermore, experimental results from the Auction show that delays in transmission time have a negative effect on an agent's bid outcome. Participant agents that record a higher number of delays show a lower acceptance rate. GA control has minimised the variance in acceptance rates among agents, thereby improving participants' chances of having their bids accepted.

The effect of delays on agents with more varied bidding strategies has not been explored in the present evaluation. The bidding strategy of agents has been controlled, so that all agents have the same percentage increase for bid proposal. A more realistic scenario would be where agents adopt different strategies and display complex behaviour in announcing their bids. However, an initial step is to examine the effect of delay on bidding agents.

Chapter 6

Conclusion

This thesis has addressed the verification of agent interaction through run-time monitoring and control. Through the review of verification approaches to agent interactions, it is found that there is a need for run-time approaches [8]. The aim of this project was to minimise faults in communicating agents, thus ensuring correct agent interaction.

The idea of a governing entity may contribute to the control of emergent risk from undesired scenarios in communication. Many challenges to applying this verification approach exist, such as the lack of uniform design and development methodologies for MAS communication as well as the diverse landscape of MAS environments [25].

In this work, the application of Petri nets as a model for monitoring and error detection at run-time was investigated. An IPN model was proposed to represent multiple interactions simultaneously. The experimental evaluation demonstrated the ability of the IPN to detect errors through marking analysis.

The second component of this work was exploring the feasibility of applying a governing agent. The agent's objective is to detect undesired scenarios in interaction and react with predefined actions. A governing agent takes the role of a supervisor over agent communication and ensures correct interac-

tion.

The following sections outline the contributions of this research, the limitations of the proposed framework, and areas of future work that might be considered.

6.1 Contributions

The main contribution of this thesis is a new class of Petri nets called Interaction Petri nets. The proposed model addresses the issue of monitoring multiple interaction protocols simultaneously. Many Petri net models of interaction have been proposed [12, 36, 20], but these have mainly focused on representing a single interaction protocol within a Petri net. Agents are expected to engage in multiple conversations which may also be instances of different interaction protocols.

The IPN was proposed for the objective of monitoring the state of interaction. To capture multiple conversations between agents, the IPN defines Petri net places as agents and tokens as messages. In contrast to previous models in which a Petri net place represents a state, the IPN model can be considered an agent-based model. This approach enables the tracking of messages that are part of different interaction protocols.

Furthermore, common undesired scenarios of communication such as IP delay, lost messages, busy-wait, transmission delay, and agent termination are defined for detection through an IPN marking. A key advantage to the specification of general issues is the loose coupling with interaction protocols. The protocols that regulate interactions vary between different application areas. As a result, the definition of common scenarios provides a usable approach for detection and does not rely on the specifics of a particular interaction protocol.

The IPN facilitates the identification of five scenarios that can affect protocol enactment. The IPN marking, which defines the placement of tokens,

enables analysis of the current state of conversations. With message metadata defined using a colour set, issues of interaction can be detected such as delay in arrival, IP completion delay and lost messages. Complete protocol enactment may be affected by other issues such as incorrect message formats or unexpected messages; however such scenarios are not covered for detection by this approach but may be explored in future work.

A governing agent is introduced for detection and control. GA behaviour deals with controlling the effect of detected undesired scenarios. First, a detection process for each of the five scenarios of communications is implemented. Second, a number of actions are designed in response to the these scenarios for assuring correct interaction. The internal structure of the GA is purely reactive, where an action is selected based on the detected issue. A corrective action is specified for each scenario, which includes status updates passed to the participating agents.

The GA and IPN combined comprise our framework for runtime detection and control. We demonstrated the effectiveness of the framework through experimental evaluation. Two MAS case studies that involves heterogeneous collaborative and competitive agents in an open and asynchronous environment was implemented. The first case study was based on the Dedale test-bed [37], but was supplemented with three interaction protocols to examine the effect of undesired scenarios of interaction. The second case study is an auction among multiple participants. The English Auction protocol is a common protocol that is widely used for commerce. The experimental results show the successful detection of scenarios of interaction. Moreover, the control actions are evaluated based on agent performance indicators. The application of GA control has corrected interaction protocols, which consequently improved agent performance.

Furthermore, a comparison analysis was conducted of IPN and a previous Petri net model of interaction called Scalable Petri nets [36], referred to here as SPN. We extended the SPN model with a detection process for the defined

undesired scenarios in Chapter 3 to carry out the experiment. Two SPN nets were implemented to model the protocols of the treasure hunt case study. IPN has the advantage when it comes to monitoring different protocols within the same environment, as it shows better time complexity when it comes to monitoring mora than one protocol. When considering scalability, it may depend on the target MAS environment. IPN space is correlated with the number of agents while SPN is on the size of the protocols and the states a protocols holds.

Finally, an extension of IPN to be applied in a distributed environment is presented. The idea is to distribute the task of monitoring among a number of governing agents. Each GA maintains an IPN to capture the status of messages within its local node. A marking update process is introduced to maintain consistency of the replicated markings managed by governing agents. The distributed monitoring and control is evaluated in both case studies. The results in Chapter 5 are consistent with the local experiments of Chapter 3.

6.2 Limitations

The contributions of this thesis needs to be considered in light of the following limitations:

The application of the proposed framework requires knowledge of the target environment and the behaviour of agents, specifically the control component. Control is achieved through GA behaviour. The aim of a control action is to minimise the effect of a detected scenario, and successfully achieving this goal relies on how well it is designed.

The governing agent presented in Chapter 3 is designed as purely reactive agent. Such a GA selects its action based on the detected scenarios. The evaluation is carried out within an environment that also consists of simple reactive agents. The design of control actions would be more challenging

in cases where agents exhibit complex behaviour and have a high level of autonomy.

Moreover, the design of the governing agent is made on the assumption that agents are willing to share interaction information with a governing agent. This is suitable for collaborative environments where agents share an objective, but it might not be the case for adversarial agents.

Additionally, an issue that relates to the distributed application of IPN in Chapter 5 is the consistency of replicated markings across nodes. In the applied model, consistency of marking relies on GA communication. Each GA would forward its local marking to other agents for updates through message exchange. The reliability of this approach is affected by risks of communication. Any network issues such as delays would in turn impact detection.

6.3 Future Work

The following areas are suggested for future work:

Commitment protocols. The presented IPN model is concerned with interactions that follow operational interaction protocols. An important area for future work is to extend the IPN to model commitment-based protocols. A commitment protocol is different from an operational protocol in that messages are defined in terms of social commitments. Agents interact through the declaration and manipulation of social commitments to one another, which provides for more flexible message exchange.

In order to consider this area, we consider the framework presented in [29]. Fornara and Colombetti transformed the semantics of an ACL into a commitment-based definition. They present an approach to produce operational specification of a commitment-based ACL. Key components of the approach include commitment objects, temporal proposition objects, and

communicative acts. A commitment object is represented as:

$$C_{id}(state, debtor, creditor, content \,|conditions, timeout)$$

Each object consists of an identifier, a state of the commitment, a reference for the debtor, a reference for the creditor, content, conditions for the commitment, and an optional timeout parameter. Temporal proposition is used to specify the content, to which a debtor is committed, and the conditions state that needs to be satisfied for the commitment to be active. A commitment has a life-cycle indicated by the state. A change in a commitment state is affected by events that change the values of its content and conditions.

Furthermore, a group of communicative acts is redefined with social meaning. For example, the performative *inform*, used by agent a to inform agent b of content P, can be defined as the commitment object: $C_{id}(active, a, b, P \,|T)$.

Following work of [29], tokens in the IPN model will represent commitments instead of message structures. A token can be defined as a commitment object that binds interacting agents. A commitment declaration is represented through the Petri net transition, which produces a commitment token at the creditor place.

The IPN marking will hold the current status of commitments between agents. Monitoring agent commitments is carried out through the distribution of tokens within a marking combined with the state of each commitment.

Detection and control can be achieved through a specification of undesired patterns within a commitment protocol. Unlike operational protocols, where an expected sequence is defined, interaction with commitment-based protocols is more flexible. A wider set of patterns can be defined based on commitment states with respect to a specific sequence of a commitment protocol. Patterns of interaction that hold either expected or unwanted states can be specified for detection.

Consistency of replicated markings. Another direction for future work is related to Chapter 5 and distributed IPNs. To enable consistency in replicated IPN markings for governing agents, a marking update process has been implemented. However, the update process can be considered unreliable, as discussed in the limitations section. An approach based on data-centric consistency models enables access to a shared IPN marking.

We consider the IPN marking as a shared data store where read and write processes are performed by governing agents. An IPN marking reflects the creation of messages through adding tokens into a sender place, and sending messages by firing a transition that produces a token at the receiver place. The addition of tokens and transition firing can be viewed as write processes that modify a marking. On the other hand, a GA will perform a read process of the IPN marking at every detection cycle.

The Sequential consistency model can be applied to regulate write and read processes [42]. Sequential consistency ensures the order in which concurrent operations are executed appears to be sequential. Concurrent read and write processes are interleaved, but every GA will see the same order of operations. This gives every GA access to the same marking version.

Large scale evaluation. An evaluation of the monitoring and control framework in environments with a large number of agents would be beneficial for assessing scalability of the IPN model. The IPN model can modify its structure and adapt to open environments where the number of agents changes dynamically. The addition of a new agent is done by simply adding a place and connecting it to other places with transitions. It is worthwhile to examine further how well IPN adapts to an increased number of new added agents. It would also be valuable to assess the effectiveness of the detection process within a large IPN.

Detection of alternative flows. Defining general undesired scenarios in communication for detection increases the usability of the proposed frame-

work. However, in certain contexts, it may also be beneficial to detect specific patterns of interaction. An interaction protocol defines a main flow of message exchange, and in some protocols alternative flows are also defined to handle exceptional scenarios.

The detection process can be extended to detect an alternative flow within a protocol. Based on Petri net markings, a specific token message that indicates an exception within the flow of a protocol can be flagged for detection.

6.4 Concluding Thoughts

This research aimed to investigate run-time verification of agent interaction. Run-time approaches are not as widely implemented as design time verification. However, run-time verification is not a straightforward task given the large landscape of multiagent system classes. Also, there is a varied range of autonomy among agents.

This research takes on the perspective of common undesired communication scenarios in order to assure interaction. We have shown that the detection of common issues can be applied in different classes of multiagent systems.

Moreover, a verification approach applied in run-time has a window to manage mistakes in communication. The application of a corrective action can be beneficial to dynamic, asynchronous environments, which are typically the characteristics of a multiagent system. We have improved agent performance by assuring agent interaction in both case studies.

Bibliography

[1] Marco Alberti, Marco Gavanelli, E Lamm, Federico Chesani, Paola Mello, and Paolo Torroni, 'A logic based approach to interaction design in open multi-agent systems', in *13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 387–392. IEEE, (2004).

[2] Hyggo Almeida, Leandro Silva, Angelo Perkusich, and Evandro Costa, 'A formal approach for the modelling and verification of multiagent plans based on model checking and petri nets', volume 3390, pp. 162–179, (05 2004).

[3] Davide Ancona, Sophia Drossopoulou, and Viviana Mascardi, 'Automatic generation of self-monitoring mass from multiparty global session types in jason', in *Declarative Agent Languages and Technologies X: 10th International Workshop, DALT 2012, Valencia, Spain, June 4, 2012, Revised Selected Papers 10*, pp. 76–95. Springer, (2013).

[4] Davide Ancona, Angelo Ferrando, and Viviana Mascardi, 'Improving flexibility and dependability of remote patient monitoring with agent-oriented approaches', *International Journal of Agent-Oriented Software Engineering*, **6**(3-4), 402–442, (2018).

[5] František Apkovi, 'Cooperation and negotiation of agents by means of petri net-based models', in *2012 17th International Conference on Meth-*

ods & Models in Automation & Robotics (MMAR), pp. 256–261. IEEE, (2012).

[6] John Langshaw Austin, *How to do things with words*, volume 88, Oxford university press, 1975.

[7] Marina Bagić, Aleksandar Babac, and Marijan Kunštić, 'Verification of communication protocols in a multi-agent system', in *Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology*, pp. 286–291, (2008).

[8] Najwa Abu Bakar and Ali Selamat, 'Agent systems verification: systematic literature review and mapping', *Applied Intelligence*, **48**(5), 1251–1274, (2018).

[9] Matteo Camilli, Angelo Gargantini, Patrizia Scandurra, and Carlo Bellettini, 'Event-based runtime verification of temporal properties using time basic petri nets', in *NASA Formal Methods Symposium*, pp. 115–130. Springer, (2017).

[10] BG Campbell, 'Searle: Speech acts: An essay in the philosophy of language (book review)', *General Linguistics*, **14**(4), 220, (1974).

[11] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani, 'On global types and multi-party session', *Logical Methods in Computer Science*, **8**, (2012).

[12] Jose R Celaya, Alan A Desrochers, and Robert J Graves, 'Modeling and analysis of multi-agent systems using petri nets', in *2007 IEEE International Conference on Systems, Man and Cybernetics*, pp. 1439–1444. IEEE, (2007).

[13] Nader Chmait, David L Dowe, David G Green, and Yuan-Fang Li, 'Agent coordination and potential risks: Meaningful environments for

evaluating multiagent systems', in *Evaluating General-Purpose AI, IJCAI Workshop*, (2017).

[14] Amit K Chopra, Alexander Artikis, Jamal Bentahar, Marco Colombetti, Frank Dignum, Nicoletta Fornara, Andrew JI Jones, Munindar P Singh, and Pinar Yolum, 'Research directions in agent communication', *ACM Transactions on Intelligent Systems and Technology (TIST)*, **4**(2), 1–23, (2013).

[15] Amit K Chopra, Munindar P Singh, et al., 'An evaluation of communication protocol languages for engineering multiagent systems', *Journal of Artificial Intelligence Research*, **69**, 1351–1393, (2020).

[16] Amit Khushwant Chopra, Munindar P Singh, et al., 'Bungie: Improving fault tolerance via extensible application-level protocols', *Computer*, **54**(5), 44–53, (2021).

[17] Samuel H Christie, Amit K Chopra, and Munindar P Singh, 'Mandrake: multiagent systems as a basis for programming fault-tolerant decentralized applications', *Autonomous Agents and Multi-Agent Systems*, **36**(1), 16, (2022).

[18] Samuel H Christie, Munindar P Singh, and Amit K Chopra, 'Kiko: programming agents to enact interaction protocols', in *Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems (AAMAS*, volume 22, (2023).

[19] Roberta Coelho, Elder Cirilo, Uira Kulesza, Arndt von Staa, Awais Rashid, and Carlos Lucena, 'Jat: A test automation framework for multi-agent systems', in *2007 IEEE International Conference on Software Maintenance*, pp. 425–434. IEEE, (2007).

[20] R Scott Cost, Ye Chen, Tim Finin, Yannis K Labrou, Yun Peng, et al., 'Modeling agent conversations with colored petri nets', in *Working notes*

of the Autonomous Agents' 99 Workshop on Specifying and Implementing Conversation Policies, (1999).

[21] Stephen Cranefield, Martin Purvis, Mariusz Nowostawski, and Peter Hwang, 'Ontologies for interaction protocols", *Ontologies for Agents: Theory and Experiences*, 1–18, (2002).

[22] Romain Demangeon, Kohei Honda, Raymond Hu, Rumyana Neykova, and Nobuko Yoshida, 'Practical interruptible conversations: distributed dynamic verification with multiparty session types and python', *Formal Methods in System Design*, **46**, 197–225, (2015).

[23] Nirmit Desai and Munindar P Singh, 'On the enactability of business protocols.', in *AAAI*, pp. 1126–1131, (2008).

[24] Virginia Dignum and Julian Padget, 'Multiagent organizations', *Multiagent systems*, **2**, 51–98, (2013).

[25] Mariana Falco and Gabriela Robiolo, 'A systematic literature review in multi-agent systems: Patterns and trends', in *2019 XLV Latin American Computing Conference (CLEI)*, pp. 1–10, (2019).

[26] Angelo Ferrando, Michael Winikoff, Stephen Cranefield, Frank Dignum, and Viviana Mascardi. On the enactability of agent interaction protocols: Toward a unified approach, 2019.

[27] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire, 'Kqml as an agent communication language', in *Proceedings of the Third International Conference on Information and Knowledge Management*, CIKM '94, p. 456–463, New York, NY, USA, (1994). Association for Computing Machinery.

[28] ACL Fipa, 'Fipa acl message structure specification', *Foundation for Intelligent Physical Agents, http://www. fipa. org/specs/fipa00061/SC00061G. html (30.6. 2004)*, (2002).

[29] Nicoletta Fornara and Marco Colombetti, 'Defining interaction protocols using a commitment-based agent communication language', in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 520–527, (2003).

[30] Ernesto German and Leonid Sheremetov, 'An agent framework for processing fipa-acl messages based on interaction models', in *International Workshop on Agent-Oriented Software Engineering*, pp. 88–102. Springer, (2007).

[31] Castagna Giuseppe, Mariangiola Dezani, Luca Padovani, et al., 'On global types and multi-party sessions', *Logical Methods in Computer Science*, **8**, 1–45, (2012).

[32] Jorge J Gomez-Sanz, Juan Botía, Emilio Serrano, and Juan Pavón, 'Testing and debugging of mas interactions with ingenias', in *Agent-Oriented Software Engineering IX: 9th International Workshop, AOSE 2008 Estoril, Portugal, May 12-13, 2008 Revised Selected Papers 9*, pp. 199–212. Springer, (2009).

[33] Miguel Escrivá Gregori, Javier Palanca Cámara, and Gustavo Aranda Bada, 'A jabber-based multi-agent system platform', in *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, p. 1282–1284, New York, NY, USA, (2006). Association for Computing Machinery.

[34] Celia Gutiérrez, Iván García-Magariño, Emilio Serrano, and Juan A Botía, 'Robust design of multi-agent system interactions: A testing approach based on pattern matching', *Engineering Applications of Artificial Intelligence*, **26**(9), 2093–2104, (2013).

[35] Gery Gutnik and Gal Kaminka, 'A scalable petri net representation of interaction protocols for overhearing', in *International Workshop on Agent Communication*, pp. 50–64. Springer, (2004).

[36] Gery Gutnik and Gal A Kaminka, 'Representing conversations for scalable overhearing', *Journal of Artificial Intelligence Research*, **25**, 349–387, (2006).

[37] Cédric Herpson, 'Dedale: A dedicated testbed for multi-agents problems', (2019).

[38] Kurt Jensen and Lars M Kristensen, *Coloured Petri nets: modelling and validation of concurrent systems*, Springer Science & Business Media, 2009.

[39] Chung-Hsien Kuo and Ting-Shuo Chen, 'Modeling and control of autonomous soccer robots using high-level petri nets', in *Proceedings of SICE Annual Conference 2010*, pp. 2226–2231. IEEE, (2010).

[40] Timothy Lacey and Scott A DeLoach, 'Automatic verification of multiagent conversations', in *Proceedings of the Eleventh Annual Midwest Artificial Intelligence and Cognitive Science Conference*, pp. 93–100. AAAI Press Fayetteville, Arkansas, (2000).

[41] Dung N Lam and K Suzanne Barber, 'Debugging agent behavior in an implemented agent system', in *International Workshop on Programming Multi-Agent Systems*, pp. 104–125. Springer, (2004).

[42] Lamport, 'How to make a multiprocessor computer that correctly executes multiprocess programs', *IEEE transactions on computers*, **100**(9), 690–691, (1979).

[43] Martin Leucker and Christian Schallhart, 'A brief account of runtime verification', *The Journal of Logic and Algebraic Programming*, **78**(5), 293–303, (2009).

[44] Yoo Jin Lim, Gwangui Hong, Donghwan Shin, Eunkyoung Jee, and Doo-Hwan Bae, 'A runtime verification framework for dynamically adaptive

multi-agent systems', in *2016 International Conference on Big Data and Smart Computing (BigComp)*, pp. 509–512. IEEE, (2016).

[45] Pedro Lima, Hugo Gracio, Vasco Veiga, and Anders Karlsson, 'Petri nets for modeling and coordination of robotic tasks', in *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)*, volume 1, pp. 190–195. IEEE, (1998).

[46] Alessio Lomuscio and Franco Raimondi, 'Mcmas: A model checker for multi-agent systems', in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 450–454. Springer, (2006).

[47] Xiangyu Luo, Mengmeng Zou, and Lingjie Luo, 'A modeling and verification method to multi-agent systems based on kqml', in *2012 IEEE Symposium on Electrical & Electronics Engineering (EEESYM)*, pp. 690–693. IEEE, (2012).

[48] Laetitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat, 'Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems.', (2012).

[49] Tadao Murata, 'Petri nets: Properties, analysis and applications', *Proceedings of the IEEE*, **77**(4), 541–580, (1989).

[50] James J Odell, Harry Van Dyke Parunak, and Bernhard Bauer, 'Representing agent interaction protocols in uml', in *International Workshop on Agent-Oriented Software Engineering*, pp. 121–140. Springer, (2000).

[51] Nardine Osman, David Robertson, and Christopher Walton, 'Run-time model checking of interaction and deontic models for multi-agent systems', in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 238–240, (2006).

[52] Shamimabi Paurobally, Jim Cunningham, and Nicholas R Jennings, 'Verifying the contract net protocol: a case study in interaction protocol and agent communication semantics', (2004).

[53] Franck Pommereau, 'SNAKES: a flexible high-level Petri nets library', in *Proceedings of PETRI NETS'15*, volume 9115 of *LNCS*, pp. 254–265. Springer, (06 2015).

[54] David Poutakidis, Lin Padgham, and Michael Winikoff, 'Debugging multi-agent systems using design artifacts: The case of interaction protocols', in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, pp. 960–967, (2002).

[55] Ali Akbar Pouyan and Fateme Jafarinejad, 'Collision avoidance in dynamic multi-agent systems using petri net-based supervisor', in *International Conference on Artificial Intelligence, Energy and Manufacturing Engineering*, pp. 38–42, (2015).

[56] MK Purvis, Peter Hwang, MA Purvis, SJ Cranefield, and Martin Schievink, 'Interaction protocols for a network of environmental problem solvers', (2002).

[57] Awais Qasim, Sidra Kanwal, Adnan Khalid, Syed Asad Raza Kazmi, and Jawad Hassan, 'Timed-arc petri-nets based agent communication for real-time multi-agent systems', *International Journal of Advanced Computer Science and Applications*, **10**(9), (2019).

[58] Anne Vinter Ratzer, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen, and Kurt Jensen, 'Cpn tools for editing, simulating, and analysing coloured petri nets', in *Applications and Theory of Petri Nets 2003*, eds., Wil M. P. van der Aalst and Eike Best, pp. 450–462, Berlin, Heidelberg, (2003). Springer Berlin Heidelberg.

[59] Jeffrey S Rosenschein and Gilad Zlotkin, *Rules of encounter: designing conventions for automated negotiation among computers*, MIT press, 1994.

[60] Francesca Saglietti, David Föhrweiser, Stefan Winzinger, and Raimar Lill, 'Model-based design and testing of decisional autonomy and cooperation in cyber-physical systems', in *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, pp. 479–483. IEEE, (2015).

[61] Michael Schumacher and Sascha Ossowski, 'The governing environment', in *International Workshop on Environments for Multi-Agent Systems*, pp. 88–104. Springer, (2005).

[62] Sanjit A Seshia, Dorsa Sadigh, and S Shankar Sastry, 'Towards verified artificial intelligence', *arXiv preprint arXiv:1606.08514*, (2016).

[63] Weihua Sheng and Qingyan Yang, 'Peer-to-peer multi-robot coordination algorithms: petri net based analysis and design', in *Proceedings, 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics.*, pp. 1407–1412. IEEE, (2005).

[64] Carles Sierra, 'Agent-mediated electronic commerce', *Autonomous agents and multi-agent systems*, **9**, 285–301, (2004).

[65] Munindar P Singh, 'Information-driven interaction-oriented programming: Bspl, the blindingly simple protocol language', in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 491–498, (2011).

[66] Munindar P Singh, 'Bliss: Specifying declarative service protocols', in *2014 IEEE International Conference on Services Computing*, pp. 235–242. IEEE, (2014).

[67] Jiacun Wang, 'Petri nets for dynamic event-driven system modeling.', *Handbook of Dynamic System Modeling*, **1**, (2007).

[68] Gerhard Weiss, *Multiagent Systems*, The MIT Press, 2013.

[69] Danny Weyns, Alexander Helleboogh, and Tom Holvoet, 'The packet-world: A test bed for investigating situated multi-agent systems', in *Software Agent-Based Applications, Platforms and Development Kits*, pp. 383–408. Springer, (2005).

[70] Danny Weyns and Fabien Michel, 'Agent environments for multi-agent systems–a research roadmap', in *Agent Environments for Multi-Agent Systems IV: 4th International Workshop, E4MAS 2014-10 Years Later, Paris, France, May 6, 2014, Revised Selected and Invited Papers*, pp. 3–21. Springer, (2015).

[71] Danny Weyns, H Van Dyke Parunak, Fabien Michel, Tom Holvoet, and Jacques Ferber, 'Environments for multiagent systems state-of-the-art and research challenges', in *Environments for Multi-Agent Systems: First International Workshop, E4MAS 2004, New York, NY, July 19, 2004, Revised Selected Papers 1*, pp. 1–47. Springer, (2005).

[72] Michael Winikoff, Nitin Yadav, and Lin Padgham, 'A new hierarchical agent protocol notation', *Autonomous Agents and Multi-Agent Systems*, **32**, 59–133, (2018).

[73] Michael Wooldridge, Michael Fisher, Marc-Philippe Huget, and Simon Parsons, 'Model checking multi-agent systems with mable', in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, pp. 952–959, (2002).

[74] Michael Wooldridge and Nicholas R Jennings, 'Intelligent agents: Theory and practice', *The knowledge engineering review*, **10**(2), 115–152, (1995).

[75] Dianxiang Xu, Richard Volz, Thomas Ioerger, and John Yen, 'Modeling and verifying multi-agent behaviors using predicate/transition nets', in *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pp. 193–200, (2002).

[76] Nobuko Yoshida, Raymond Hu, Rumyana Neykova, and Nicholas Ng, 'The scribble protocol language', in *Trustworthy Global Computing: 8th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers 8*, pp. 22–41. Springer, (2014).

[77] Lin Zishen, Li Wei, and Li Maoqing, 'Modeling decision and cooperation of multi-agent system using petri net', in *2009 4th International Conference on Computer Science & Education*, pp. 643–646. IEEE, (2009).