A Concurrent Bilateral Negotiation Model for Open E-Markets

by
Bedour Alrayes

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

Department of Computer Science,
Royal Holloway, University of London

December 2015

Declaration of Authorship

I, Bedour Alrayes, hereby declare that this thesis and the work presented in it is entirely my own. Where I have consulted the work of others, this is always clearly stated.

Signed:

Date:

Supervisor and Examiners

Supervisor: Professor Kostas Stathis

Internal Examiner: Dr. Antonis Bikakis

External Examiner: Professor Jeremy Pitt

Abstract

Negotiation has been widely applied in computational systems, particularly in e-marketplaces (e-markets), to facilitate transactions between buyers and sellers. In existing e-markets, participants must repeatedly be online to follow the progress of a transaction, especially where negotiation is required. Also, some market mechanisms (e.g. auctions) can be of long duration, and pursuing the best deals can require participants to engage in multiple negotiations simultaneously. In this context, the issue is how to build models that support agents to engage in open and dynamic environments concurrently.

We develop a model that allows software agents to negotiate concurrently with other agents. The main goal is to deploy agents that achieve the best outcome in agreements that allocate resources such as goods and services. We build upon previous work to develop a complete concurrent negotiation architecture, describing all the necessary components to allow an agent to take decisions in a concurrent negotiation. We also revise the known alternating protocol to support concurrent negotiations in open e-markets. This allows participants to request-to-reserve, cancel or exit a negotiation.

We also develop a novel strategy, CONAN, which relaxes assumptions previously made regarding deadlines and knowledge about the market and the opponents. Consequently, our strategy is more realistic for open e-markets in deciding what action to take and, if the action is to offer, what offer to make next. We represent CONAN using a logic-based knowledge representation. We then build a negotiation simulator RECON to simulate and evaluate our strategy. RECON supports the development of software agents negotiating concurrently with other agents;

previous work only supported single bilateral negotiation. We then create a set of realistic experimental negotiation scenarios using opponents from the existing literature. We show empirically that CONAN outperforms the state-of-the-art and other agents in terms of average utility gained from negotiations.

Acknowledgements

Undertaking this PhD has been a truly life-changing experience for me and it would not have been possible without the support and guidance that I have received from many people.

I would like to express my special appreciation and thanks to my supervisor Professor Kostas Stathis, for his positive and encouraging approach. His knowledge, wisdom, patience, support, valuable guidance and advice on both the research and my career have been invaluable. I have been very privileged to have him as my supervisor.

A very special thank you to Dr.Özgur Kafalı for his invaluable advice and feedback on my research and for always being very supportive of my work.

I am extremely grateful to my parents. Firstly, my mum, for her unconditional love and support. She is the most important person in my world and I dedicate my PhD to her; without her I would not be here today. Also, a big thank you to my dad for his support and for always believing in me.

Profound gratitude goes to my beloved husband Saad, for his continuous support and for all of the sacrifices that he has made on my behalf. I would like also to thank my daughters Alia and Yara, who have shared with me all the good and the hard times during my PhD.

My deep appreciation goes to my siblings Nora, Hadeel, Nasser, Haifa, Reema and Faisal, for their moral support and precious love throughout my entire life.

To my dearest friend and confidante, Latifa Alabdlkrim, for sharing every moment in my life and being the best friend who inspires me; I always look forward

to spending time with her.

My sincere thanks also go to my family-in-law. Words cannot express how grateful I am to my mother-in-law and father-in-law, whose prayers for me have sustained me thus far.

I also thank the DICE Lab for all the lovely times and for sharing so many experiences. In particular, I am grateful to Paulo Ricca for helping me to fix the code bugs and Ulli Schaechtle for useful discussions about statistical tests.

Finally, I would like to thank King Saud University for funding my PhD.

Contents

List of Figures

List of Tables

Nomenclature

α time to complete one negotiation round

$\delta^t_{ag_i \to ag_j}$ is a negotiation action from ag_i communicated to ag_j at time t,

δ global negotiation situation for buyer b

δ_i sum of the scores $Criteria_1$ and $Criteria_2$

λ is a real constants

μ is a real constants

ϕ window of consecutive opponent offers

A set of protocol actions

AU_j average utility per parameter combination

AUS_j average utility over successful runs

B set of buyers

b buyer agent

$c1$ competitor agent

C^t_b set of competitor agents for b at time t

C_t number of active competitor agents factor

CR_{t,s_i} concession rate at time t for seller s_i

$Criteria_1$ first criteria to evaluate negotiation situation for all threads

$Criteria_2$ second criteria to evaluate negotiation situation for all threads

D percentage of the canceled reserved offer

DM impact of the negotiation price range

$DO_{Seller_b^t}$ number of canceled offers that occur from sellers $Seller_b^t$

E_t effect of the environment factors

EG_b eagerness of buyer b

IP_b and RP_b represent the initial and reservation prices of buyer b

IP_{s_i} and RP_{s_i} represent the initial and reservation prices of seller s_i

MAN maximum number of active threads

MC number of seconds at which the e-market has to change

MCO maximum number of reserved offers

MD number of market agents in the e-market at any given time point

MDL deadline for all market agents

MR ratio of buyer agents to seller agents in the e-market

$NoSimulations$ number of simulation runs

NS negotiation situation for all threads factor

$Penalty_b$ total cancellation penalty for b

$Penalty_{b,s_i}$ cancellation penalty to/from seller s_i

r a resource

R_{ds} demand/supply ratio factor

RC number of repeated runs in each parameter combination

S set of sellers

s seller agent

s_i sellers agents

S_t effect of self factors

Se_t number of sellers that are actively negotiating with b factor

$Seller_b^t$ set of seller agents negotiating with b at time t

SR_j number of successful runs

t time

T_b maximum duration of time that b can negotiate for

T_e deadline of negotiation for b

T_s negotiation start time

TE is the effect of the passage of time on the concession rate for buyer b factor

$U_b(x)$ utility function of agent b

w_{Env_t} environment factor weights

w_{Self_t} self factor weight

y is the number of sellers

Chapter 1

Introduction

Negotiation is back and forth communication to reach an agreement when two or more parties have some shared and conflicting interests [35]. Negotiation occurs in many real-life situations: between husband and wife, between siblings, between friends, between employees and employers, between firms and between countries [88]. Negotiation between humans may consume a lot of time and the negotiators need to have relevant negotiation skills. Consequently, to make negotiation effective, the field has been studied in such diverse areas as Economics and Computer Science. In Computer Science, automated negotiation is studied in the field of Artificial Intelligence to develop improved technology and to overcome the limitations of human negotiations. Negotiators in this setting are normally represented in automated negotiation as software agents. An *agent* is a software component that encapsulates its own state and can interact autonomously with other agents in multi-agent systems to achieve their goals [49, 117]. A multi-agent system is a system that contains a number of agents, which interact with each other by exchanging messages [116].

Automated negotiation is the process of agents communicating with one another in order to reach agreements or solve conflicts on matters of common interest [48, 63, 116]. Automated negotiation between multiple agents has recently gained substantial attention [54] because agents isolate emotions and feelings when

they negotiate [35], and the use of agents saves time, especially in open and dynamic environments that are hard for humans to handle [97, 99]. An environment is open and dynamic if the number and the behaviour of the participant agents is unknown and changing over time. There are many applications of automated negotiation, including e-commerce [30, 82, 107], service selection and composition [20], virtual organisations [71], supply chains [44], service-oriented industries [72] and cloud computing [3].

In this thesis the focus is on electronic marketplaces (e-markets) as dynamic environments for automated negotiation. E-markets are normally understood as inter-organizational information systems that allow participating buyers and sellers (i.e. opponents) to exchange information about prices and items (i.e. products or services). E-markets of this type provide an electronic, or online, method to facilitate transactions between buyers and sellers, and typically support all of the steps in the entire order fulfilment process [13]. E-Bay and Amazon are examples of e-markets that have become popular due to their supporting mechanisms such as advertising, buying/selling, and paying for items online, thus providing an efficient and convenient way to carry out online commercial activities.

In the remainder of this chapter we will first outline the reasons for undertaking an investigation in this field in Section 1.1. Then we will demonstrate our negotiation problem via scenarios in Sections 1.1.1 and 1.1.2. Our negotiation challenges and the limitations that need to be addressed are then presented in Section 1.2. The aim and objectives of our work are discussed in Section 1.3. Our contributions to the state-of-the-art are listed in Section 1.4. The organization of the thesis is summarized in Section 1.5. Finally, the publications and awards resulting from some of the thesis work are outlined in Section 1.6.

1.1 Motivation

One problem with existing e-markets is that a market participant must repeatedly be online in order to follow the progress of an activity such as buying an item,

especially if the activity involves bidding. In addition, the duration of some market mechanisms (e.g. auctions) can be long, so e-market activities can often become tedious for a participant. What is worse, buyers and sellers come and go, and they have different goals and preferences. Thus pursuing the best deals in an e-market requires participants to engage in multiple, possibly conflicting activities at the same time, sometimes in different e-markets.

To deal with the open and dynamic environment that e-markets give rise to, we follow a multi-agent systems approach [112] whereby agents negotiate with other agents on behalf of their users to allocate a resource (item). A challenging task in this context is the ability of agents to negotiate with one another because e-markets are dynamic in nature. New buyers and sellers with different behaviours may enter or leave an e-market, prices of items may go up or down and the needs of the user may change over time. In addition, when a buyer agent enters an e-market to purchase an item, the buyer might need to negotiate with several sellers to get the best deal. As a result, the buyer has to keep track of each individual negotiation while deciding which ones to pursue further and perhaps compete with other buyers wishing to acquire the same item.

For all the above reasons, we are interested in many-to-many negotiations, that occur in pairs (bilaterally) and at the same time (concurrently) as illustrated in Figure 1.1. The reason behind our choice is that when a buyer agent enters any practical e-market to acquire a resource, there will be many seller agents who sell the same required resource [100] and competitors who want to buy the same resource. Thus, we will imitate a real-life buying process conducted via the Internet. Next we will explain the reason for negotiating concurrently instead of sequentially.

One of the most powerful features in Computing is the ability to support multiple tasks simultaneously (often referred to as the multi-threaded execution of a task). Thus, negotiating agents prefer to negotiate concurrently to observe the environment and to overcome one of the shortcomings of human negotiations, which is slowness. Furthermore, concurrent negotiation provides the opportunity to make

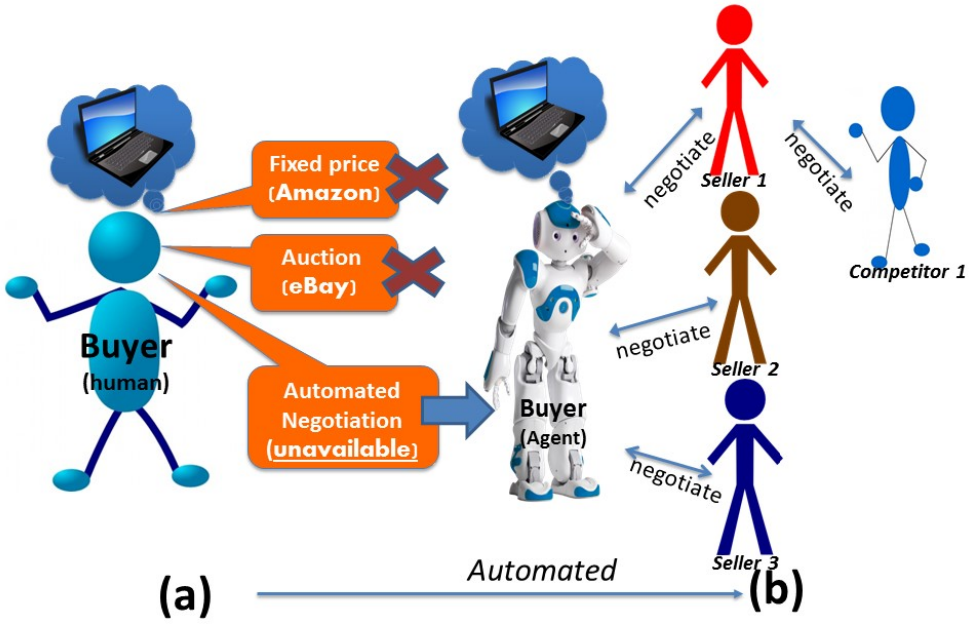

Figure 1.1: Thesis motivations.

a decision at the same time for all offers, so the buyer explores the best offer from multiple sellers and this gives more chances to reach an agreement, which maximize the agent's *outcome* [5, 79, 115]. The agent's outcome from the negotiation is either an *agreement deal* or *no agreement deal* and is determined by the agent's *utility*, specifying how much the agent gains from the negotiation. Utility can be seen as a function that is designed by the agent user to measure the performance of the agent during negotiation [55]. Moreover, concurrent negotiation provides the agent with negotiating power, which means the agent can negotiate with many opponents at the same time, to acquire a stronger negotiation stance relative to another opponent [87, 108]. In contrast, sequential negotiation consumes time in exploring and negotiating with all the relevant opponents in the e-market.

1.1.1 Scenario 1

The focus of this thesis is to develop a novel model that can handle practical, real-life time settings in negotiations where a buyer agent wants to negotiate with multiple seller agents at the same time to acquire a resource on behalf of its user. To explain the motivation for our research, this section presents a scenario from a real-life situation where our contribution will help to solve some of the main problems.

> *Bob* wishes to purchase a second-hand laptop, so he joins an e-market to negotiate a deal. *Bob* has a price range of [700-900] and a 3-day deadline to conclude the deal. When he enters the e-market, he finds two laptop sellers: *Alice* and Tom. Figure 1.2 illustrates the example. On *Monday:morning* and *Monday:afternoon*, *Bob* starts a negotiation with both *Alice* and Tom by making them an offer. On *Monday:evening*, *Alice* sends a counter-offer to *Bob* and on *Tuesday:morning*, Tom also sends a counter-offer. On *Tuesday:afternoon Bob* considers buying from Tom but on *Tuesday:evening* a new seller, $John$, appears in the e-market. *Bob* has a hard decision to make. Should he (a) buy from Tom? (b) negotiate with $John$ risking losing $Tom's$ offer? or (c) negotiate with *Alice*, Tom and $John$, in which case what offer should he make to them? *Bob* has no information about the strategy of the sellers (i.e. *Alice*, Tom and $John$) or the other buyers (i.e. competitors) who are negotiating with the sellers. In addition, the e-market environment is dynamic, in that existing sellers and competitors could leave the e-market and/or new sellers and competitors could join the e-market. Competitors are buyers trying to obtain the same resource, which implies that the agent should take the negotiation situation into account. Also, each buyer and seller has different deadlines for negotiation compared to other buyers and sellers.
>
> The main problems that occur in this scenario are: (1) *Bob* is taking

a long time to negotiate and (2) *Bob* cannot conduct a concurrent negotiation where he can communicate with multiple buyers at the same time.

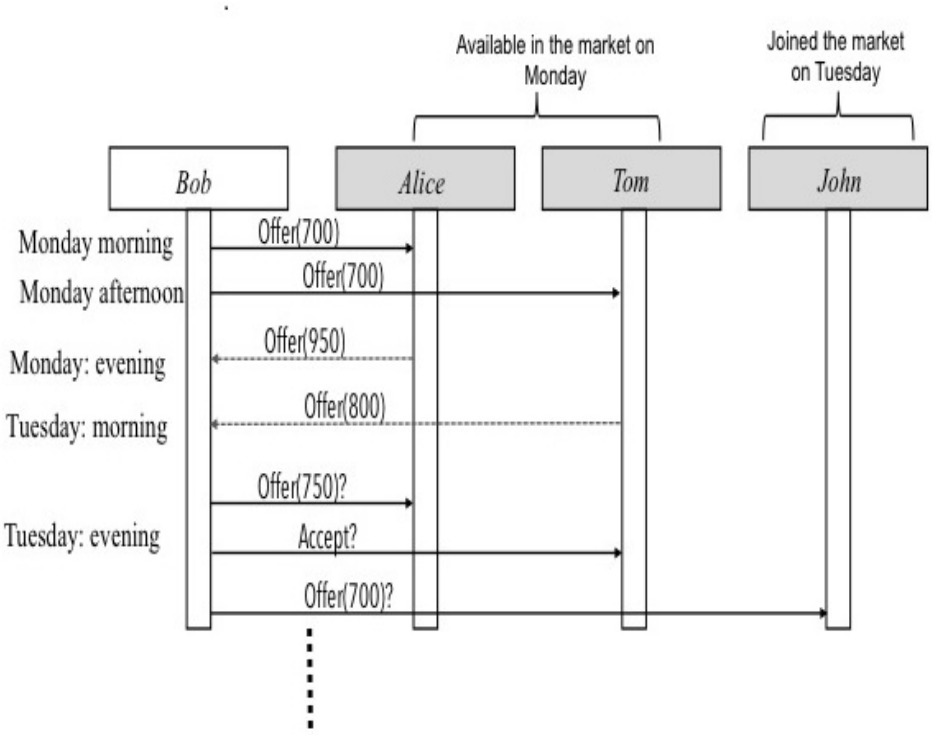

Figure 1.2: Buyer *Bob* negotiates concurrently and bilaterally with sellers *Alice*, *Tom* and *John*.

In order to overcome human limitations, in the next section, we will adapt the scenario to allow negotiation between agents in an open and dynamic environment.

1.1.2 Scenario 2

In negotiation between agents, *Bob* in Section 1.1.1 is represented by agent b, *Alice* by $s1$, *Tom* by $s2$ and *John* by $s3$. In this scenario, Figure 1.3, the buyer agent b wants to purchase a laptop on $Bob's$ behalf with a price range [700-900] and a 5-minute deadline. Buyer agent b starts to negotiate with two laptop seller agents $s1$ and $s2$ simultaneously in a trading e-market environment. This scenario

is presented from the buyer's perspective.

Since automated negotiation involves many tasks and functions to complete the whole negotiation, we will distinguish between three phases of negotiation: initial, middle and final. The reason for this distinction is to eliminate the phases that are not the focus of this thesis.

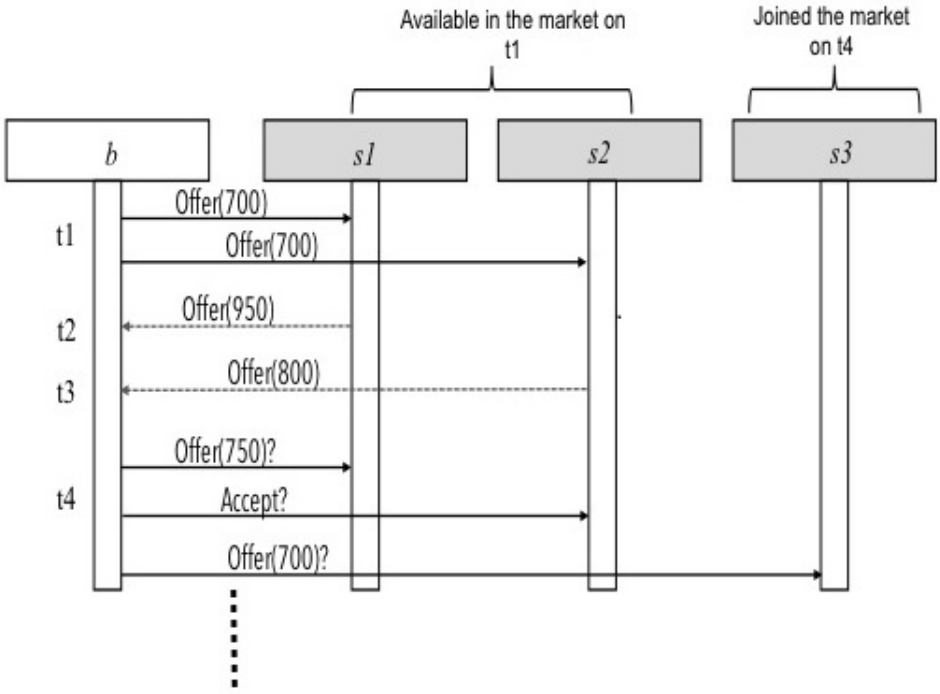

Figure 1.3: Buyer *b* negotiates concurrently with sellers *s1*, *s2* and *s3*.

Initial Phase

In the initial phase, planning and preparing for negotiation is taking place. This phase includes:

- defining the resource issues under negotiation: The item (laptop) under negotiation in our scenario involves one issue, which is the price of the laptop in GBP. This is not the focus of this thesis.

- determining the protocol of the concurrent negotiation. The protocol needs

to be flexible in order to support situations where the participants are self-interested and sometimes irrational in their actions. This is one of the important steps in the negotiation. We will focus on the negotiation protocol in this thesis.

- establishing the basic negotiation parameters predefined by the user, which we will focus on this thesis. This process includes:
  - defining the minimum value (i.e. initial price) and maximum value (i.e. reservation price) for the item. In our scenario this is defined by the user to be [700-900].
  - setting the negotiation deadline, which is the duration of the negotiation. In our scenario this is defined by the user to be 5 minutes.

Middle phase

In the middle phase, actual negotiation takes place, which includes deciding the agent's negotiation actions, exchanging offers, predicting the environment and/or opponents' behaviour changes, and measuring the status of the negotiation. Specifically, our negotiation problems (which will be listed later in Section 1.2) occur in this phase.

Furthermore, environmental or self situation changes may affect the negotiation decision-making process by increasing or decreasing the negotiation utility. The questions that arise here are: What changes should the buyer take into consideration in order to maximise the negotiation utility? How will the buyer behave in the light of these changes? What is the effect of these changes on proposing offers? The answers to these questions are determined by the negotiation strategy.

The scenario we have been exploring so far raises the following sub-scenarios for each environmental or self situation change that the buyer faces during negotiation when selecting an action for negotiating with different sellers. In the sub-scenarios, t represents a real period of time (i.e. in seconds), not a round.

(1) At time $t1$, before b starts negotiating concurrently with $s1$ and $s2$, as in Figure 1.3, b needs the answers to certain questions: (a) what kind of components does the agent need to model the negotiation and how do these components interact with each other? (b) what are the actions that can be used by the agent in the negotiation, and what are the rules governing these actions? (c) how will the agent represent its knowledge and decisions during negotiation?

(2) At time $t1$, b is negotiating concurrently with $s1$ and $s2$, as in Figure 1.3. At time $t2$, $s1$ sends a counter-offer to b while at time $t3$, $s2$ sends back an offer to b. At $t4$, a new seller $s3$ enters the e-market and b needs to make a decision on how to proceed in the negotiation, either by: (a) buying from $s2$; (b) negotiating with $s3$ risking losing $s2$'s offer; or (c) negotiating with $s1$, $s2$ and $s3$, in which case what offer should be made to them? In all these situations above, b has no information about:

   (i) the strategy of the sellers (i.e. $s1$, $s2$ and $s3$);

   (ii) how many buyers and sellers will enter and leave the e-market; and

   (iii) the deadline of the sellers.

(3) Adding and/or removing a seller to/from the negotiation: while agent b is negotiating with agent $s1$ and agent $s2$, as in Figure 1.4, a new seller agent $s3$ enters the e-market and starts a new negotiation thread with b. This creates a new option for b. As a result, b may concede by a smaller amount to see how the negotiation proceeds with agent $s3$. On the other hand, when $s1$ exits the negotiation, it means b has lost an option, thus, it has to concede by a larger amount.

(4) Adding and/or removing negotiation competitors: consider agent b negotiating with agents $s1$ and $s2$ as before, but now b perceives that a competitor $c1$ is negotiating with agent $s1$, as in Figure 1.4. Hence, b may concede by a larger amount to attract $s1$ to sell the resource. In contrast, if $c1$ leaves the e-market,

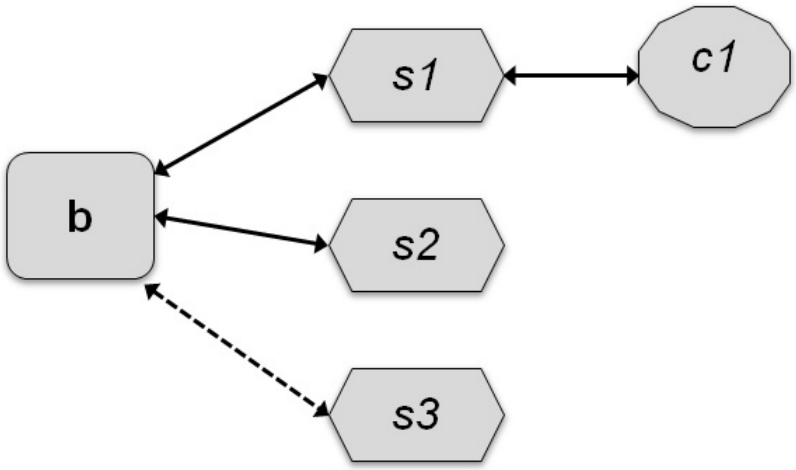

Figure 1.4: Buyer b negotiates concurrently with sellers *s1*, *s2* and *s3* with an existing competitor $c1$ in the e-market.

b may concede by a smaller amount since it has no other competitors while negotiating with $s1$.

(5) At time $t1$, b is negotiating concurrently with $s1$ and $s2$, as in Figure 1.5. At time $t2$, $s1$ sends a counter-offer to b while at time $t3$, $s2$ sends back an offer to b. At $t4$, b sends back a counter-offer to $s1$ and *Accept* to $s2$. Also at $t4$, b receives an *accept* from $s3$. Since b needs to buy one item, the problem now arises at $t5$, of how b will deal with the two accepts and whether b will continue the negotiation or not.

(6) Change in the negotiation situation: if b is in a bad negotiation situation, where the sellers: (a) take a long time to reply; and/or (b) concede in small amounts or do not concede (a notion to be defined by a function which will be explained in Chapter 3), then b has to concede more in order to secure the resource before the deadline. In contrast, when b is in a good negotiation situation, b has a higher chance of winning the negotiation, which in turn leads b to reduce its concession rate.

(7) if b's strategy is designed and implemented, the question becomes how to

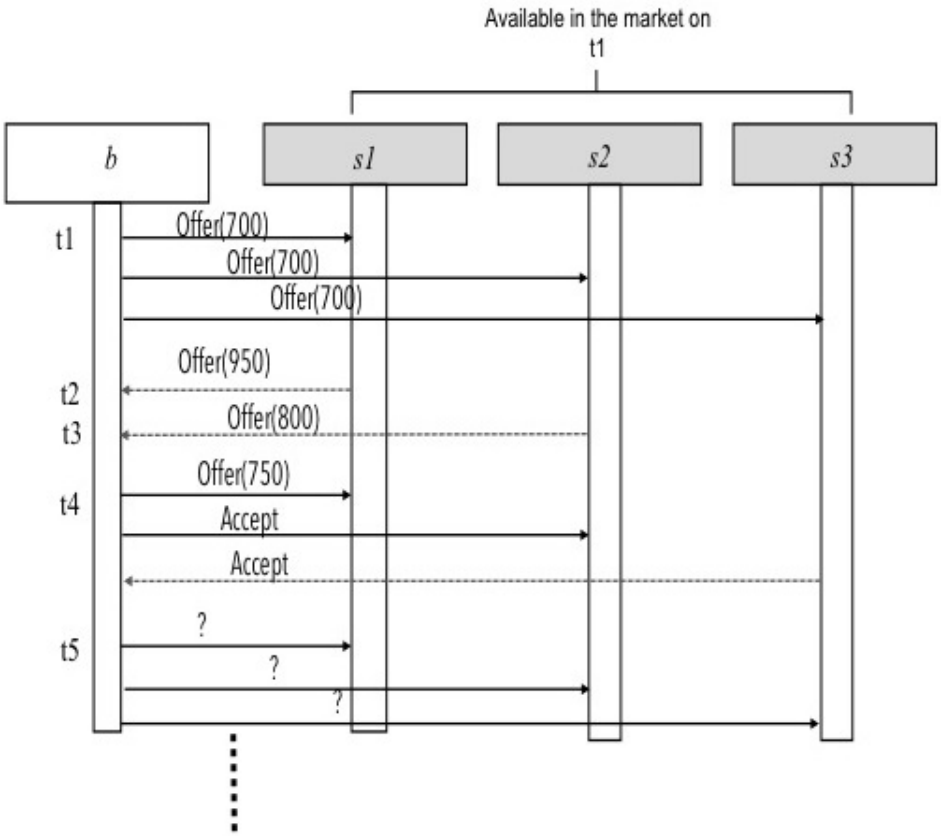

Figure 1.5: Buyer b negotiates concurrently with sellers *s1*, *s2* and *s3*.

measure the negotiation performance of b's strategy. Is there a method to compare b's performance with other agents?

Final Phase

In this phase, the buyer implements the agreement deal and payment is transferred to the seller to purchase the resource. This is not the focus of this thesis.

So, based on our scenario, the open question about negotiation in multi-agent systems is: what is the negotiation model that maximizes the agent's utility? After developing the negotiation model, we have to evaluate the new model to make sure it is maximizing the agent's utility. In this thesis, we will study our negotiation

model in open and dynamic negotiation environments. The complexity of such negotiations is discussed in the next section.

1.2 Research Issues and Challenges in Agent Negotiation

This thesis investigates the following four challenges in the concurrent negotiation field for resource allocation.

1.2.1 Complete Concurrent Negotiation Model and Knowledge Representation

From sub-scenario (1) in Section 1.1.2, we can conclude that the negotiation model includes architecture, protocol and strategy. The notion of strategy will be discussed in detail in the next section (Sections 1.2.2 - 1.2.3). There is always a challenge if the agent develops one part of the negotiation model and neglects others. This incompleteness makes the negotiation harder to model.

As we will see in Section 2.3, most of the agent architectures are either too general or are concerned only with the negotiation environment, without explicitly representing the main components of the concurrent negotiation and how these components interact with each other. It is thus a challenge to develop an architecture that contains all the necessary components for concurrent negotiation. Protocols define the rules of the negotiation. Since we are negotiating concurrently, the existing protocols are not sufficient to handle the complexity of concurrent negotiations due to their limited actions.

To the best of our knowledge, most of the agent negotiation strategies in the literature use imperative strategies as a knowledge representation method, which are usually implemented in Java. However, this type of implementation hides many details of the agent decision-making process during negotiation and it becomes hard to re-implement the original strategy without having access to the code. This

led us to look for a transparent representation for decision-making for the agent's user. Knowledge representation of the negotiation strategy should be unambiguous and well understood, and it has to provide a level of abstraction close to the key concepts of the software agent to be developed [34].

1.2.2 Open and Dynamic Environment

In real-life situations, e-markets (which represent our class of applications in negotiation) are open, dynamic environments. Openness implies that each agent enters the e-market with its own goals, preferences, interests and policies, pursuing different resources. In other words, these goals and preferences are personal and private for each agent, and so represent unknown information for other agents in the e-market [39]. In particular, there will be agents entering and exiting the environment, with each one using different strategies and demanding or offering different goods or services.

As agents are likely to be designed by different people, it is likely they will have competing agendas and goals. The issue then becomes how agents can reach agreement(s), resolve conflicts and achieve compromise without disadvantaging themselves at the end of the process. In addition, agents may behave in an irrational way. For example, an opponent may offer a lower price in one negotiation round and then offer a higher price in the next round. Also each agent has a private deadline, which may not be known by other agents in the e-market. All these challenges affect the agent's decision-making during negotiation, which is motivated in sub-scenario (2.i-2.ii) in Section 1.1.2.

The dynamism of the negotiation environment presents the agent with the following issues:

- Rapid changes in the number of opponents in the e-market, which make it difficult for the agent to construct a model of opponent behaviour or make assumptions about the number of agents that will be in the e-market. For instance, in Scenario 2, the e-market can change every two to five seconds.

This issue is motivated by sub-scenario (3) in Section 1.1.2.

- Changes in the number of competitors, which are other agents that are trying to buy the same item as our own agent. The more competitors the agent has, the more difficult it is to secure an agreement for a resource. The change in the number of competitors is elaborated upon in sub-scenario (4) in Section 1.1.2.

- Changes in the demand/supply ratio, which is the ratio of the number of buyers to the number of sellers in the e-market. The higher the ratio, the higher the price for the resource and the more difficult it is to reach an agreement.

1.2.3 Concurrent Negotiation

When a negotiating agent enters the e-market, it will usually be surrounded by multiple opponents offering its preferred resource, which reflects what happens in real-life situations. When the agent starts to negotiate with multiple opponents at the same time, the issue becomes to how to maximize the agent's outcome by selecting the negotiation with the opponent that will offer the best agreement within a certain time limit. In addition, there is the effect of large numbers of multiple opponents and other agents operating in the e-market at the same time (as in sub-scenario (1) in Section 1.1.2). A problem also arises when an agent has a large number of opponents but limited communication capabilities (e.g. time and computational resources), since concurrent negotiation requires more communication capabilities as the number of agents increases.

Furthermore, in concurrent negotiation, there is a need for highly coordinated processes to manage the ongoing negotiation threads (as in sub-scenario (5) in Section 1.1.2). Also, one must consider how progress in negotiation with one opponent will affect the progress of negotiations with other opponents (as in sub-scenario (6) in Section 1.1.2).

Another challenge is that agents can quit any negotiation process that is taking

place, which is an issue for the agent who may lose a good seller at any time. An opponent may leave the e-market either because it reaches an agreement with another agent or it reaches its deadline (as in sub-scenario (2.iii) in Section 1.1.2). Moreover, a concurrent negotiation protocol has many actions. It is up to an agent to decide when to take each action, and if an offer is received, to decide on the response.

1.2.4 Concurrent Negotiation Simulation

As raised in sub-scenario (7) in Section 1.1.2, one major issue in the design of a negotiation agent that participates in e-markets is how to evaluate the performance of its strategy. While some researchers test their agents theoretically [33], most agent developers use simulation platforms [29, 46, 59, 80, 115], especially for evaluating heuristic-based strategies. This gives rise to the need for a standardised simulation environment to provide fair and objective comparisons between negotiating agents.

A successful negotiation platform should be: (i) able to provide an open and dynamic environment for its concurrent negotiating participants; (ii) robust to the changes that occur in the e-market, (iii) reliable in its communication with other agents; and (iv) scalable in terms of the number of agents it can support. Also, the simulation should support the deployment of state-of-the-art agents that use concurrent negotiation strategies and state-of-the-art opponents.

1.3 Aims and Objectives

The main aim of the thesis *is to develop a concurrent negotiation model that supports software agents to make decisions on how to negotiate with other agents on behalf of their users for resource allocation (buying and selling goods or services) in open and dynamic electronic marketplaces (e-markets) to reach an agreement with the maximum utility for the agent.*

Achieving this aim is important because it will allow agent technology to have

an impact in widely popular e-markets such as E-Bay and Gumtree. However, existing negotiation strategies, although they offer a number of sophisticated features, such as modelling an opponent and negotiating with many opponents at the same time, abstract away from the dynamicity of the e-market and the progress of ongoing negotiations, thus ignoring information that increases an agent's utility during negotiation. In addition, they often make major assumptions about the domain (e.g. that deadlines are public), thus concentrating their applicability to a narrow range of practical negotiation settings.

Objectives

According to the concurrent negotiation challenges in Section 1.2 the objectives of this thesis are to:

1. Develop a concurrent negotiation *architecture*. If we are to build a practical negotiation agent, we need to have an architecture that can support the concurrent negotiation processes. The architecture has to combine both the agent's components and its behaviour. The agent's components should provide the basis that is needed to formulate the agent's strategy, including which component is involved in each phase of the negotiation, as represented in Section 1.1.2. The architecture will be part of the agent's complete negotiation model.

2. Develop a concurrent negotiation *protocol*. Negotiation between parties should be governed by a flexible interaction protocol that: (a) is suitable for self-interested agents seeking to maximize their utility without any kind of cooperation with other agents; (b) allows actions to handle different situations in concurrent negotiations, where the agent has to deal with different actions at the same time; (c) ensures fairness of actions for all the agents involved in the negotiation; and (d) is appropriate for an open and dynamic environment.

   The protocol will be part of the complete negotiation model.

3. Develop an agent that is able to:

   - Negotiate in an *open and dynamic environment*. The agent has to deal with open and dynamic environments. This is important because most real-life e-markets are open and dynamic in nature.
   - Negotiate with *unknown opponents*. The agent has to deal with opponents where it has no information about their behaviour in negotiations. Unknown opponents will ensure imitation of real-life situations. *Unknown opponents* means that there is undisclosed information about their: (a) negotiation strategy; (b) rational or irrational nature; (c) negotiation deadline; (d) maximum and minimum negotiation price; and (e) previous negotiation history.

     Also, opponents should be developed by different researchers to confirm comparative negotiation and to make sure there will be unpredictable and complex negotiation behaviour. As a result, our opponents should be drawn from the state-of-the-art [29, 37].
   - Support *continuous real-time* negotiation. The agent will negotiate in real time, reflecting real-life situations such as existing e-markets, where the decision on what action to take is based on how much time is left in the negotiation and not on the number of rounds that have been determined for the negotiation.
   - Handle *continuous negotiation outcomes*. The agent will negotiate the price of a resource, which is the only attribute of the resource. The price has a continuous value, for instance, price $\in [300 - 500]$. Consequently, the agent needs to deal with a large number of possible outcomes.
   - *Negotiate concurrently*. In this case the agent takes advantage of negotiating with multiple opponents at the same time. This should include adaptive strategy that incorporates a method to take into consideration the agent having to receive and generate many negotiation actions si-

multaneously. In addition it needs to take advantage of the concurrent negotiation protocol (Objective 2).

- Has *explicit decisions* for negotiation actions. The agent has a decision strategy that determines clearly when to take each of the appropriate negotiation actions; for instance, when to accept or when to exit.
- *Maximize the agent utility.* Since there are other agents described in the literature that can negotiate concurrently with multiple opponents, the agent has to maximize the utility gained during negotiation compared to existing state-of-the-art agents.
- Decide in a *computationally tractable* manner. The agent has to negotiate in an open and dynamic environment, so the agent's decisions in negotiations should be completed in a reasonable finite amount of time, which is measured in seconds. It is therefore important that the agent uses a computationally tractable strategy.
- Support *asynchronous negotiation.* In asynchronous negotiation, the agent does not have to wait to receive all the sellers' counter-offers before replying, which saves the agent negotiation time and ensures that the agent replies on time.

4. Develop a *declarative agent.* The agent should have a logical reasoning process for taking actions. This is important for the agent's user, who will be able to better understand the reason behind the agent's actions in an abstract way. This will help the agent in the future to justify each action taken in negotiation.

5. Build a concurrent negotiation *simulator.* We need to develop an environment that supports the simulation of concurrent negotiations. The simulator needs to: (a) support concurrent negotiation; (b) provide an open and dynamic environment; (c) allow agents to enter and leave the environment dynamically at run-time; (d) integrate different agent strategy implementa-

tions and technologies; (e) be robust to simulation parameter changes; (f) allow reliable communication among agents; and (g) be scalable.

6. Conduct extensive *empirical evaluations*. Our agent has to perform as well as or better than the state-of-the-art agents in e-market settings. We will therefore need to conduct empirical evaluations via simulations to provide an experimental approach for assessing the performance of agents. In addition, with empirical evaluation we can imitate real-life dynamic e-markets, where the setting of the e-market can be changed to test different conditions.

1.4 Contributions

The work described in this thesis makes a number of important contributions, as follows:

- We propose a *concurrent negotiation architecture* designed as a specialized extension of previous work with the KGP model [36, 107] to satisfy the requirements of concurrent bilateral negotiation. KGP is a model of agency consisting of (a) the knowledge K that the agent uses to reason and act in the environment in which it is situated; (b) the goals G that represent what states of the environment the agent wishes to achieve; and (c) the plans P that represent how the goals of the agent can be achieved as a series of actions or further sub-goals. The architecture (Objective 1) describes all the necessary components for taking decisions in the concurrent negotiation environment.

- We introduce a *concurrent negotiation protocol*. The protocol (Objective 2) is a revised version of the well-known alternating protocol that can support concurrent negotiations for open e-markets. In this revised protocol, buyers and sellers can decide to offer, accept, request-to-reserve, cancel or exit a negotiation, thus providing the necessary flexibility that was not always possible in previous work.

- We develop a *novel strategy,* CONAN, as part of an adaptive agent model for concurrent bilateral negotiations by considering a weighted combination of modelling the e-market environment and the progress of concurrent negotiations in which the agent is involved. We show how to increase the agent's utility during negotiations. We also relax strong assumptions previously made at the multi-agent level. CONAN satisfies Objective 3 in Section 1.3.

- We represent the *logic-based knowledge representation of* CONAN, which includes how negotiation action and its effects are represented for a declarative agent using Event Calculus. To the best of our knowledge, we believe there is a lack of using Event Calculus to formalize negotiation strategies. This formalization has, amongst other things, the concomitant advantage that it can help the agent in the future to justify each action taken in the negotiation. Knowledge representation in CONAN satisfies Objective 4 in Section 1.3.

- We present the *design, implementation, and experimental evaluation of* RECON*: a Robust multi-agent Environment for simulating COncurrent Negotiations* (Objective 5). RECON supports the development of software agents (both buyers and sellers) negotiating concurrently with other agents, while all the existing work only supports bilateral negotiation. In addition, most negotiation agent development platforms, such as GENIUS, only support imperative (e.g. Java) agents, while RECON supports both imperative (e.g. Java) and declarative (e.g. Prolog) concurrent strategies. Declarative strategies allow developers to specify strategies that can be transparent to a human user, in that the agent can explain why it has taken certain actions during a negotiation.

- We show empirically that CONAN outperforms the state-of-the-art [115], Random and Faratin's [29] strategies in terms of average utility gained from negotiations (Objective 6). Via experiments, we show that our results are statistically significant. We designed experiments by creating various real-

istic negotiation scenarios using opponents from the existing literature. We did so because available agent negotiation simulators did not allow open e-markets or concurrent negotiations. We then implemented a wide range of opponents from the agent strategies used in the Automated Negotiating Agents Competition (ANAC).

1.5 Thesis Organization

The rest of this thesis is structured as follows: Chapter 2 provides a structure for classifying the wide domain of negotiation and then offers a review of architectures, protocols and strategies; in particular, those that are most relevant to our concurrent negotiation setting. Chapter 3 presents our negotiation strategy CONAN by formalising our concurrent negotiation environment in terms of the negotiation architecture and protocol, and describing the technical details of our CONAN strategy, with emphasis on the factors that the strategy considers, and conditions under which the agent takes action. In Chapter 4, we introduce the knowledge representation of CONAN using Event Calculus and Prolog. In Chapter 5, we discuss the development and evaluation of our concurrent negotiation simulation RECON. Chapter 6 explains our experimental setting and evaluates the experimental results for CONAN. We benchmark the current state-of-the-art concurrent negotiation strategy and other strategies; also we implement a wide range of opponents' strategies. Finally, in Chapter 7, we conclude this thesis and outline areas of future work.

1.6 Publications and Awards

The work in this thesis has generated the following publications:

(i) **Alrayes, B.**, Stathis, K.: An agent architecture for concurrent bilateral negotiations. In: F. Dargam, J.E. Hernndez, P. Zarat, S. Liu, R. Ribeiro, B. Delibai, J. Papathanasiou (eds.) Decision Support Systems III - Impact of

Decision Support Systems for Global Environments, Lecture Notes in Business Information Processing, vol. 184, pp. 79–89. Springer International Publishing (2014).

This publication presents our concurrent negotiation architecture and protocol. Section 3.1, Section 3.3 and Chapter 4 are based on this publication.

(ii) **Alrayes, B.**, Kafalı, Ö., Stathis, K.: CONAN: A heuristic strategy for concurrent negotiating agents. In Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems, AAMAS14, pp. 1585–1586. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2014).

These papers present the CONAN strategy. Chapter 3 and Chapter 6 are based on this publication.

(iii) **Alrayes, B**., Kafalı, Ö., Stathis, K.: RECON: a robust multi-agent environment for simulating concurrent negotiations. In Seventh International Workshop on Agent-based Complex Automated Negotiations(ACAN), AAMAS (2014).

This publication presents the RECON simulator. Chapter 5 is based on this publication.

In addition, **awards** have been granted as follows:

- (iii) won best student presentation award in ACAN2014.
- The overall system won second place in the Innovation and Entrepreneurship Prize for Saudi Students in the UK in 2015.

Chapter 2

Literature Review

In this chapter we present the background work that provides the foundations of the existing work on the automated negotiation. We will first briefly introduce the concepts relating to negotiation preliminaries (Section 2.1). The characteristics of the resource (item) under negotiation will be illustrated in Section 2.2. We will then discuss the literature that is relevant to our negotiation architecture (Section 2.3) and protocol (Section 2.4). Next, we will review negotiation strategies proposed in the current literature (Section 2.5). A literature review of heuristic negotiation strategies will then be presented in Section 2.6. Then we will assess the simulation platforms for negotiation that are currently available (Section 2.7). Section 2.8 presents the limitations of the existing work. Finally, we summarize our findings in Section 2.9.

2.1 Negotiation Preliminaries

The ability to negotiate (bargain) is valuable and it can significantly affect our lives in positive ways. In this section, we will introduce: *software agent* and *negotiation model*. The negotiating agent is a software agent that uses a *negotiation model* to allocate a resource on behalf of the user.

2.1.1 Software Agent

Agents are the key players in negotiation. There are three key concepts for agents: flexibility, autonomy, and situatedness. Jennings *et al.* [49] state that flexibility means that the agent has three properties. The first property is *reactivity*: agents are capable of responding to changes in their environment in a timely manner to achieve their goals. The second property is *proactiveness*: agents are able to satisfy their objective by behaving in a goal-directed manner. The last property is *sociality*: agents can interact with other agents or humans to satisfy their objectives [116]. Autonomy means that the agent is self-controlled and able to act without being directed by a human or other agents [49]. On the other hand, situatedness means that the agent can perceive its environment via its sensory inputs, and according to these it generates actions which change the environment.

In this thesis, we will deal with buyer and seller (we will use the terms opponents and sellers interchangeably) software agents.

2.1.2 Negotiation Model

Negotiation is a complex process which can be seen from many perspectives. For this reason, there are multiple ways to classify negotiation models. Negotiation models have three main elements:

- *Architectures* define the general structure and components of the negotiating agents.
- *Protocols* define the legal actions that negotiating agents can make (i.e. the rules of negotiation). The protocols are public, so every agent knows the rules of negotiation in advance.
- *Strategies* determine an agent's behaviour in negotiation by specifying when and how to act. The strategies are private, which implies that each agent does not know the other agents' strategies.

An agent makes proposals defined by its protocol, using an agent strategy. The negotiation procedure contains a series of rounds, and each round can involve one or more proposals. Negotiation terminates when the strategy of one of the negotiating agents determines it.

Lopes *et al.* [66] presented a generic model for automated negotiation. The model consists of four phases: preliminaries; pre-negotiation; actual negotiation; and renegotiation. The negotiation starts with the preliminaries, which describe the process of detecting social conflict and determining the negotiating parties. In the pre-negotiation step, the agents prepare and plan for negotiation by structuring personal information, analysing their opponents' agents, and selecting their protocols and initial strategies. Next is the actual negotiation, where the following take place: exchange of offers and feedback; argumentation; learning from negotiation to improve future performance; choosing a strategy dynamically; and finding resolution for impasses. The final step is renegotiation, which involves the process of analysing and improving the final agreement. The strength of the presented framework is that it outlines all the phases of negotiation, avoiding considering the middle phase only, which can be a limitation. In addition, the authors focus on the lack of structure in the negotiation field.

A well-defined classification scheme for negotiation models was proposed by Lomuscio *et al.* [63]. This classification model is based on six groups of parameters. The first parameter, the cardinality of negotiation, specifies the number of parties and the number of issues. The second, the characteristics of agents, involves role, rationality, knowledge, commitment, social behaviour and bidding strategy. The third parameter, environment description, specifies whether it is a static or dynamic environment, and whether the items are public or private. The fourth consists of the event parameters, which discuss bid validity and visibility, clearing and quote schedules. Information parameters constitute the fifth group, and they involve the quote's price, transaction history and arguments. The final set of parameters is the allocation parameters, which determine the winner. Also, Kraus [54] reviewed the automated negotiation process and described the

literature in the following areas: negotiation models, protocol and strategies.

2.2 Negotiation Resources

Negotiation involves dealing with the issues (attributes) of the resource under negotiation. These attributes represent the features of the resource. The agents may negotiate on one issue only (e.g. price). In Section 2.2.1 we describe *single-issue negotiation*, as involved in our scenarios in Chapter 1, while in Section 2.2.2 we deal with more than one issue, which is known as *multi-issue negotiation* (e.g. price, colour, and warranty). In addition, a negotiation issue may hold a *continuous* or a *discrete* value. A continuous issue value belongs to a range of values, for instance, price $\in [300 - 400]$. A discrete issue has a value that belongs to a finite set; for instance, price $\in \{300, 310, 350\}$. The negotiation issues set the *solution space* [48]. The solution space represents all proposals that may be offered by the agents in the negotiation and is determined by the number of issues.

2.2.1 Single-Issue Negotiation

Many real-world commercial negotiations deal with one issue only. For instance, a landlord and a tenant negotiate about the rent price for a property; and an employer and an employee negotiate the salary of the employee [31]. In our model, we will use continuous single-issue negotiation over price, for multiple reasons: (a) automated negotiation fields, and specially concurrent negotiation, need many improvements and concentrating on single-issue negotiation will accelerate the development of the automated negotiation area; (b) strategy that has been developed for a single issue can be easily adapted to multiple issues [23]; (c) many real-life applications need to negotiate a single-issue only, such as price, when buying/selling an item; and (d) single-issue negotiation will ensure our computationally tractable objective (Section 1.3), since a single-issue requires less computation than multiple issues.

2.2.2 Multi-Issue Negotiation

Many resources have multi-issue characteristics. For instance, when a buyer wants to purchase a car, he/she negotiates with the seller about many issues related to the car (e.g. price, colour, and warranty). In multi-issue negotiation, agents may have different preferences on the issues, hence, agents prioritise issues that are most important for them by making concessions on less important issues.

However, multi-issue negotiation requires more complex protocols and strategies than single-issue negotiation. This is because modelling the preferences of an agent on multiple issues is a complex process. The complexity increases when the number of issues increase and when the issues' values depend on other issues' values. Thus, the solution space in multi-issue negotiation is n-dimensional ($n > 1$) rather than a one-dimensional line, as in a single-issue negotiation. Therefore, the negotiation strategy in multi-issue negotiations has to be more sophisticated [42, 57] to handle the complexity of the multiple issues. In addition, many problems arise when the agent wants to concede. It is challenging to specify which issues the agent should concede on during negotiation, what represents the best combination of issues, and whether it is possible to concede on issues that are more strongly preferred by the opponent in order to make the offer more acceptable. Also, the amount of concession on each issue must be determined, along with whether the value of a single-issue is dependent on other issues [93]. As a result, reasoning in a multi-issue negotiation strategy is more complicated and consumes more time and computational resources than a single-issue negotiation.

We refer to *negotiation procedures* [33] as the way to conduct multi-issue negotiations over resources, for example:

- *separate*: agents negotiate each issue separately (independently and simultaneously)
- *package deal* (simultaneous): agents negotiate a complete package on all issues simultaneously.

- *sequential*: agents negotiate issue by issue sequentially, which requires deciding the order of the issues to be negotiated (agenda). The issues may be independent, or partially dependent on each other.

2.3 Negotiation Architecture

An agent's architecture shows how an agent is structured and how it functions [113]. The review below discusses previous research on agent architectures for negotiation and especially how these architectures fail to support the objectives of this thesis, in particular:

1. support for concurrent bilateral negotiation;
2. explicit representation of the agent's state in relation to the components described in the architecture and how these components interact with each other;
3. independence from the negotiation strategy or protocol;
4. interaction between negotiating software agents and not humans.

We start with the work of Fasli [30], who introduced the agent's structure in an architecture which includes a self-model representing the agent's beliefs, intentions, preferences and mental attitudes during negotiation. However, the model does not include the environment. Also, the model is very abstract and not designed for concurrent negotiation. Ashri *et al.* [10] proposed a bilateral agent architecture that includes the environment, the opponent and the self-model, but which includes only the goal of the negotiation, not the current state of the negotiation. Resinas *et al.* [92] defined the negotiation architecture as consisting of four phases: (1) a logical phase that identifies the key components and their interactions; (2) a process phase that identifies how the architectural components can be grouped together into processes; (3) a development phase that includes a reference implementation that developers can use to build their own negotiation agents; and

(4) a scenarios phase which is used to test the negotiation. However, the proposed architecture is protocol-dependent and strategy-dependent. Fabregues *et al.* [27] proposed an agent architecture for concurrent negotiation. However, it is tailored to build agents capable of interacting with humans in competitive environments, which does not satisfy our objectives.

It is important to address the limitations identified with the architectures above to: (a) assist the agent's developers in implementing agent strategies based on the capabilities defined in the architecture [10]; (b) provide adaptable designs to be used by the agent developer to implement any negotiator concurrent strategies, thus saving the cost of developing the architecture from scratch [10, 56, 67, 92]; and (c) provide a basic terminology that facilitates communication between agent developers [92].

In the next chapter, we will develop an agent architecture that overcomes the architecture limitations mentioned above.

2.4 Negotiation Protocol

In order to allow agents to communicate with each other they need an *interaction protocol*. A protocol defines the rules of the interaction and guides the agents in how to move as they perform actions from the initial state of the interaction to the final state. The interaction actions in a protocol have precise and unambiguous meanings [7]. Negotiation is a particular kind of interaction, so, for agents to interact they need a *negotiation protocol*. A negotiation protocol defines the rules of negotiation and therefore in automated negotiation we need to decide on the protocol that governs the negotiation process. Protocols are dependent on the number of agents participating in the negotiation; either one-to-one or many-to-many, as follows:

- **one-to-one negotiation (bilateral)**: the simplest case where two agents negotiate with each other, e.g. a car salesman and a buyer;

- **many-to-many negotiation (multilateral)**: the general case where many agents negotiate with different groups of agents concurrently, so multiple negotiations from the same agent occur at the same time. Examples include a continuous double auction and concurrent negotiations. A special case from many-to-many is one-to-many negotiation where one agent negotiates with several other agents, e.g. an auction.

As mentioned in Chapter 1, we are interested in many-to-many negotiation models, sometimes referred to as concurrent bilateral negotiation models. The reason behind our choice is that typically the negotiation model for many practical settings, like e-markets, is many-to-many, but the negotiation between parties is bilateral, to respect the privacy of individual negotiations. So, when a buyer enters any practical e-market to allocate a resource, there will be many opponents and other agents who sell/buy the same required resource [100].

In this section, we will review the available protocols for many-to-many negotiation. Then we will justify which protocol is most suitable for our concurrent bilateral negotiation model. In Sections 2.4.1– 2.4.3, we will present the *Contract Net Protocol, Auction Protocol* and *Alternating Offers Protocol*. We chose these three protocols because they are the only many-to-many protocols needed to illustrate the existing limitations with respect to the concurrent bilateral negotiation model. Then we will discuss previous attempts to extend the Alternating Offers Protocol in Section 2.4.4.

2.4.1 Contract Net Protocol

The Contract net protocol (*CNP*) is a multilateral protocol used in task allocation. It was developed by Smith [106] to specify communication and control of nodes in a distributed problem. The nodes are the agents and the task allocation is an example of a distributed problem. There are a *manager* and a set of *contractors*. A manager is responsible for monitoring the execution of a task and the contractor is responsible for executing the task.

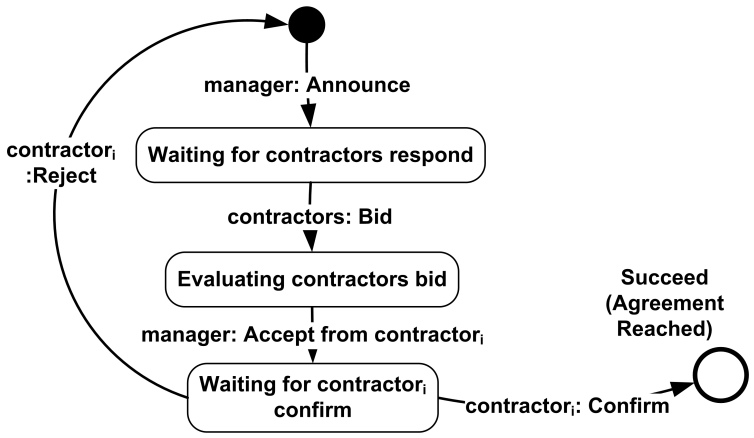

Figure 2.1: Contract Net Protocol State Diagram.

Figure 2.1 illustrates the *CNP* state diagram, where the manager starts the negotiation by announcing a task to all agents, then those contractors who are able to accomplish the task will send a bid message to the manager. The manager will award the task to the contractor with the most favourable bid. When the contractor finishes the task, it will send back the results to the manager.

However, the *CNP* is not suitable for our protocol design objectives, which are mentioned in Section 1.3, since we have self-interested agents and the market is open and dynamic.

2.4.2 Auction Protocol

Auctions are one of the oldest forms of markets [30] and are widely used in real life for resource allocation for different contexts, including buying and selling in person or via the Internet and between governments or traders [99]. In fact, the popularity of the current online automated auction system (e.g. eBay, the Google Adwords framework, trading agent competition [111]) reflects the success of auctions. An auction is a protocol for many types of one-to-many negotiations [116]. The most popular type is the English auction (e.g., eBay), which includes one resource, one seller and multiple buyers, as in Figure 2.2. The buyers offer bids to the seller and the seller replies with yes or no. The auction ends when an agreement is reached

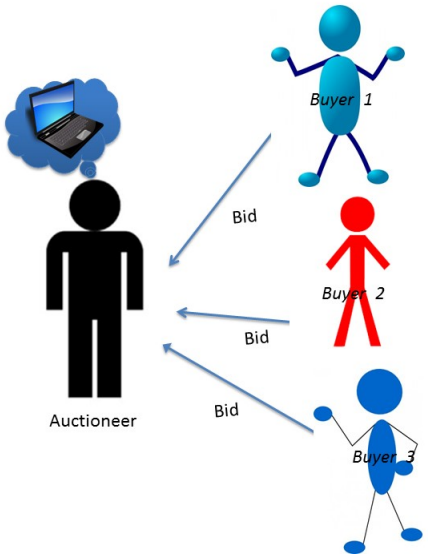

Figure 2.2: English Auction.

or the deadline for the auction is reached.

An auction is not suitable for our concurrent protocol for many reasons. Auctions consume time to allocate a resource. For example, a customer participating in an English auction can spend several days purchasing an item. The buyers must continually bid for the resource until the auction closes after several days. Accordingly, auctions will not be useful for impatient or time-constrained buyers [41].

Communication between agents in an auction is unidirectional, while it is bidirectional in a bargaining model; thus, the buyer in the bargaining model can send proposals and counter-proposals, and can use different types of strategies for different types of sellers [78, 84].

Due to their limitations in e-markets as mentioned above, which are contrary to our objectives of developing a concurrent negotiation protocol (Objective 2, Section 1.3), auction protocols are not suitable for adoption as our negotiation protocol.

2.4.3 Alternating Offers Protocol

An alternating offers protocol [95] is a simple and widely used protocol [2]. As shown in Figure 2.3, in this protocol a buyer b and a seller s take turns to negotiate, with possible actions: *offer, accept* and *reject*. *offer* represents the value of the resource under negotiation; for instance, in a single-issue negotiation, the offer will be on the price of the resource. *Accept* indicates that agreement has been reached between b and s and the negotiation terminates. *Reject* means that agent b did not agree with the offer from opponent s and it will propose a counter-offer. The negotiation terminates either because an agreement is reached (buyer/seller accept) or when the time limit is reached (buyer/seller reach their deadlines).

The alternating offers protocol is suitable for self-interested agents and ensures fairness between agents. However, it is not sufficient to handle the complexity of concurrent negotiation due to its limited actions.

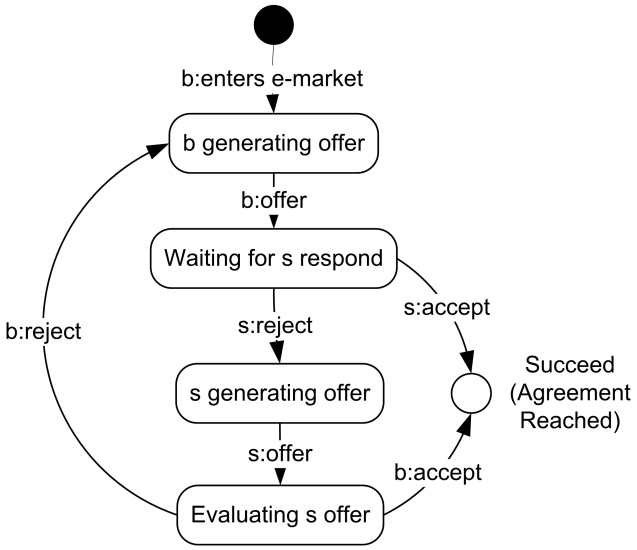

Figure 2.3: Alternating Offers State Diagram involving a buyer **b** and a seller **s**.

2.4.4 Concurrent Alternating Offers Protocol

The original alternating offers protocol [95] is simple and commonly used [2] in many real life situations, for instance, buying and selling in the context of a sec-

ond hand market. However, in realistic applications, and especially in concurrent negotiation settings where a buyer agent engages in multiple bilateral negotiations in order to acquire a resource, agents need more complex negotiation actions to handle the following situations: (a) agents have different negotiation deadlines, so the agent needs to terminate the negotiation even before agreement is reached; (b) the agent could reach an agreement with one opponent, so it needs to quit the negotiation with other opponents; (c) agents have limited computational resources, so an agent may need to exit from some negotiations to accommodate new negotiations with new opponents; (d) an agent could receive multiple accept actions from multiple opponents at the same time, so the agent needs to have an action that allows it to choose the most preferred offer; (e) an agent could choose one offer and want to secure it until it explores all the other options available in the e-market; and (f) agents and opponents need to withdraw from previous agreements if they find new preferred agreements.

For these reasons, the original alternating offers protocol is an appropriate base protocol for dealing with concurrent negotiations. Therefore we need to extend the original alternating offers protocol to handle the complex concurrent negotiation situations mentioned above.

There have been two previous attempts to extend the alternating offers protocol for use in concurrent negotiations. In [1], the available protocol actions are $A = \{offer[x], accept, exit, confirm\}$; a confirm action is presented, which comes after an accept, and the buyer or seller cannot cancel confirmed offers. The limitations in this method are: (a) it does not ensure fairness in negotiations where the seller has to wait a long time before receiving confirmation; and (b) there is no cancel action, so the buyer will lose the opportunity to find another best offer from another seller.

In [115], Figure 2.4, a concurrent protocol is proposed and the possible actions are:

$A = \{offer[x], accept, exit, confirm, de\text{-}commit\}$

The limitations in this method are that: (a) when the buyer makes a deal

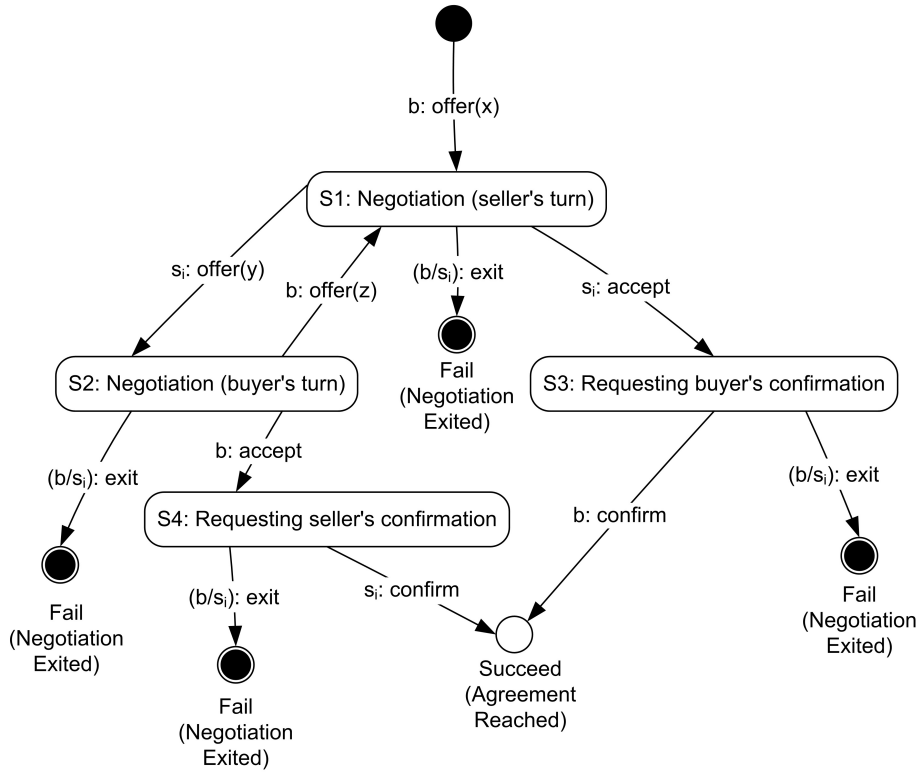

Figure 2.4: Williams' Concurrent Alternating Offers State Diagram.

via a confirm action, the deal will be completed and the buyer will obtain the item under negotiation and the seller will receive the money for that item, so if the buyer cancels the deal (via a de-commit action), it results in a more complex situation where the buyer has to return the item back to the seller and the seller has to return the money back to the buyer, which can be inefficient in real-time applications; and (b) only the buyer can cancel the deal (via a de-commit action), which is not fair for the seller if the seller receives a better offer from another buyer. Moreover, the de-commit action is not explicitly shown in the state diagram of [114]. Furthermore, it implies that the protocol goes with a de-commit action from a final state (an agreement) to another final state (a conflict deal), which is logically problematic.

Because of the shortcomings in [1, 115], which do not allow all parties to cancel a reserved offer, we chose to extend the alternating offers protocol to create the

concurrent alternating offers protocol by (a) adding a cancel action for both buyer and seller to ensure fairness; and (b) allowing the buyer to take more opportunities to find the best offer by introducing a new action, 'request-to-reserve' that can provide a hold situation for a deal.

2.5 Negotiation Strategies

In this section, we will firstly classify agent strategies based on the type of artificial intelligence (AI) techniques used to design the strategy and our choice of the negotiation strategy type (Section 2.5.1). Next, we will present a classification of heuristic strategies with a discussion of related existing work (Section 2.5.2).

2.5.1 Negotiation Strategy Types

A negotiation strategy determines an agent's behaviour in negotiation and it is an important element in the negotiation model. Each agent uses a strategy to describe how it will behave during the negotiation. For example, the strategy defines the content of the first proposal and also when and how much agents can concede [116]. The negotiation process is analysed based on three types of strategies [48, 85, 86]: *Game Theoretic Strategies*, *Argumentation-Based Strategies* and *Heuristics Strategies*.

Game Theoretic Strategies

These strategies are highly abstract and use mathematics to express their strategies formally [81]. A game theory strategy concentrates on the negotiation outcome rather than the negotiation process itself. A game theoretic strategy calculates the best negotiation action from all possible actions by taking into account the decisions that other agents may make [48]. For example, the buyer will formulate a mathematically abstracted strategy that assumes that the number of sellers in the market and their strategy is known in advance.

However, game theoretic strategies contain many unrealistic assumptions including: (a) the negotiation environment is closed and static; (b) agents have unbounded computational resources; (c) possible negotiation outcomes are small and completely known; (d) agents are fully rational; (e) negotiation time is discrete; and (f) the opponent's strategy is known.

As a result, game theory is often not adopted in multi-agent systems in general and in practical negotiation models in particular [48, 87]. Since it is unrealistic to assume common known information about the negotiation environment and opponents, it is also difficult to compute optimal negotiation strategies [5, 87]. Therefore we do not consider game theory strategies in this thesis because they do not satisfy our objectives (Objective 3) outlined in Section 1.3.

Argumentation-Based Strategies

The agents in the argumentation model attach reasons to their proposals, as well as reasons for accepting or rejecting them, to try to change the opponent's negotiation stance [86]. Negotiation proceeds by exchanging proposals and counter-proposals. Moreover, the exchange includes promises, threats, or preferences [70]. Preference is about the features of the item under negotiation. For instance, if agent b wants to buy a laptop from s_i, the following dialogue represents the argumentation model using preferences:

s_i: I will sell the laptop with price 900.

b: I will buy the laptop with price 700 if the laptop color is red.

Since our focus is on single-issue negotiation (Section 2.2) and one of our objectives is to develop an agent that behaves in a computationally tractable manner (Section 1.3), presenting an argument will increase the complexity of the model (i.e. the argument needs a different evaluation mechanism than the price). For this reason, argumentation-based models are beyond the scope of this thesis.

Heuristic Strategies

Heuristic strategies are functions that determine how the agent behaves both at the beginning and during the negotiation to reach the best course of action that leads to an agreement. Heuristic strategies contain rules that produce good but not optimal outcomes [87]. Hence, these strategies are based on empirical testing and evaluation. Heuristics offer decisions that approximate the ones made in game-theoretic studies, and are therefore more suitable for practical applications.

Heuristic strategies have many features. They: (a) are suitable for open and dynamic environments, where it is a very complex process to optimize the negotiation strategy [3]; (b) deal with uncertain and unknown knowledge about the negotiation environment and opponents [43]; (c) do not assume that agents exhibit perfect rationality, since in the practical application agents may be irrational, malicious or badly coded [87]; (d) support continuous real-time negotiation; (e) can handle very large amounts of solution space (Section 2.2); (f) are computationally tractable; (g) concentrate on both the negotiation process and the outcome; and (h) involve direct modelling of agent behaviour. Accordingly, heuristic strategies have been adopted successfully in a wide range of automated negotiation literature [3, 5, 6, 23, 28, 79, 83, 91, 115].

There are, however, some disadvantages: (a) they produce sub-optimal negotiation outcomes because they approximate the decision process of negotiation; and (b) it is difficult to accurately predict the models' behaviour, thus, this type of strategy needs intense empirical evaluation and simulation. Since heuristic strategy features satisfy our objectives (Objective 3) outlined in Section 1.3, we chose to develop a concurrent heuristic negotiation strategy to be used by an agent.

On reviewing the existing work on heuristic strategies, we found that there was no complete list of all negotiation strategies in the multi-agent literature [85, 87]. Next, we will classify the basic heuristic strategies proposed so far in the multi-agent negotiation literature.

2.5.2 A Classification of Heuristic Strategies

Lopes *et al.* [66] list three fundamental groups of strategies:

- *Contending*: agents maintain their stance and try to persuade opponent agents to concede.
- *Concession making*: agents reduce their offer until it is acceptable to their opponent. Concession strategies [64, 65] are functions that define how the agent behaves at the start of the negotiation and the concession behaviour of the agent. The following are three sub-classes of concession strategies:
  - Starting high and conceding slowly: agents adopt an optimistic opening position and make only low concessions during negotiation.
  - Starting high and conceding moderately: agents adopt an optimistic opening position and make moderate concessions throughout negotiation.
  - Starting reasonable and conceding slowly: agents adopt a realistic opening position and make low concessions during negotiation.
- *Problem-solving*: agents maintain their offers and try to adjust the offer to match the opponents. It only applies when the agent negotiates a recourse with multiple issues. Problem-solving strategies [64, 65], like concession strategies, are functions that determine the behaviour of the agent at the start and during the negotiation. The following are three sub-classes of problem-solving strategies:
  - *expanding the pie*: agents increase the available resources under negotiation (i.e. the pie) so that each agent achieves its goals.
  - *logrolling (trade-offs)*: each agent concedes on issues that are of low preference to itself and of high preference to the other agents.

- *nonspecific compensation*: one agent achieves his goals by paying off an opponent for accommodating his interests.

Our strategy focuses on concession strategies because they are the simplest and most widely used. In addition, our negotiation objective concentrates on single-issue negotiation only (Section 2.2.1). The essential work that defines concession-making strategies is that of Faratin *et al.* [29], who developed a model that defines a range of strategies for proposing an offer and a counter-offer and making a decision about a proposal. Faratin *et al.* developed the following families of strategies:

- **Time-dependent** [29]: an agent that adopts this strategy will concede more rapidly as the deadline approaches. Therefore, the value of the counter-offer and the acceptance value for the issue depend on the remaining negotiation time. There are three variations within this family of strategies:
  - *Boulware*: the agent concedes less at the beginning of the negotiation, but when the available negotiation time nears its end, it will concede more; thus the concessions will take place at the end.
  - *Linear*: the agent concedes constantly during negotiation.
  - *Conceder*: the agent concedes more at the beginning of the negotiation, but it will concede less at the end of the negotiation.

  Figure 2.5 illustrates the Boulware, Linear and Conceder strategies. The y axis represents the concession amount where 0 is the lowest value and 1 is the highest value. The x axis represents the negotiation time where 0 indicates the start of the negotiation and 1 indicates the end of negotiation.

- **Resource-dependent** [29]: this family of strategies is similar to the time-dependent family of strategies except that the agent will concede more rapidly as the quantity of resources becomes limited. Resources could be the money transferred between agents, the number of agents participating in a particular negotiation, or negotiation time. Thus, time-dependent strategies can

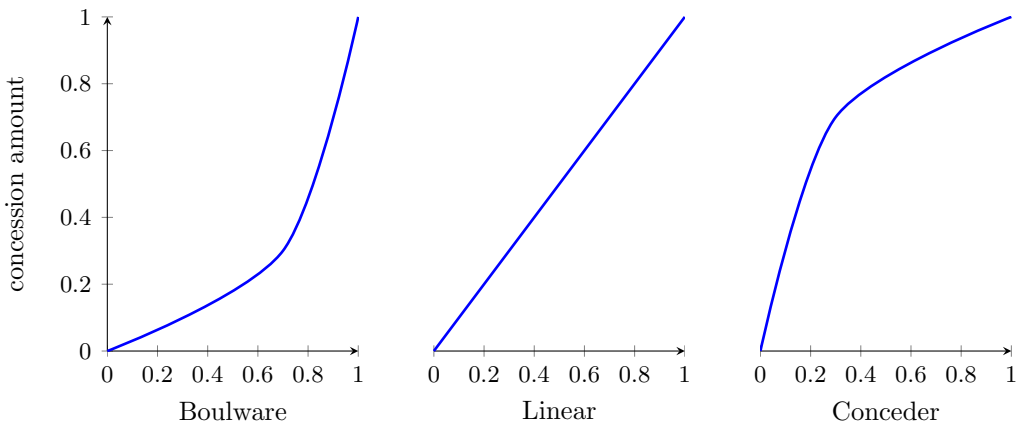

Figure 2.5: Faratin's *et al.* [29] concession-making strategies.

be considered a type of resource-dependent strategy. There are two types of strategies in this family:

- *Dynamic-deadline strategy*: this strategy is used to determine the deadline value. The quantity of negotiation time is proportional to the number of participating agents and inversely proportional to the average length of the negotiation thread.
- *Resource-estimation strategy*: a counter-offer is calculated depending on the way a particular resource is consumed. In general, if there are many resources, then the agent will use a Boulware strategy. For example, if the number of negotiating agents is the resource, then the more agents, the smaller the concession.

- **Imitative** [29]: The strategies in this family are used when the agent is not under pressure to reach an agreement, which is different from the previous families, in which there is a reason for the pressure (approaching deadline and limited resources). The agent will imitate the behaviour of the opponent, and will compute the next offer based on the previous attitude of the opponent. These strategies vary in type depending on the behaviour of the opponent they imitate and the degree to which they imitate the opponent's behaviour.

There are three strategies in this family:

- *Relative Tit-For-Tat*: the agent reproduces the behaviour of the opponent in a percentage manner.
- *Random Absolute Tit-For-Tat*: the agent adopts the opponent's behaviour in absolute terms. This strategy is randomised by the use of a function to generate a random integer that modifies the value of the offer by an increasing or decreasing amount. For instance, if the opponent increased the offer by 3, then the agent's counter offer should be decreased by 3 plus or minus the random number.
- *Averaged Tit-For-Tat*: the agent uses a period of the opponent's history to compute the average percentage of change in the opponent's behaviour in order to produce an offer.

Since Faratin's *et al.* [29] strategies are simple and widely used in the negotiation field [5, 31, 116], we will use Faratin's *et al.* strategies to develop buyers (as benchmark agents) and sellers (as opponent agents) in our experimental evaluation in Chapter 6.

2.6 Heuristic Strategies

We will classify examples in the literature depending on the number of agents negotiating at the same time: bilateral negotiations and multiple concurrent bilateral negotiations. Then, we will illustrate asynchronous negotiation. At the end of this section, we will review the cancellation management models used to cancel an negotiation agreement.

In bilateral negotiation strategies [29, 58, 69, 75] a buyer negotiates one-to-one with a seller to obtain a resource. We will not compare our strategy with the bilateral negotiation strategies since our strategy lies in the category of concurrent bilateral negotiations.

Our review will focus on addressing the following questions: (a) how can an agent behave in a dynamic environment where there are both environmental changes (i.e. changes that are out of the agent's control, namely changes in the environment) and/or self changes (i.e. changes in the agent's model that the agent holds for itself in terms of ongoing negotiations); and (b) what information is available for an agent during the negotiation to support the decision-making process. In addition, we will focus on the functionalities of the concurrent negotiation strategies listed in Table 2.1.

2.6.1 Concurrent Bilateral Negotiation Strategies

In this section, we will discuss in detail the existing strategies that are related to our model: multiple concurrent bilateral models.

- Nguyen and Jennings [78] developed and evaluated a heuristic concurrent bilateral negotiation strategy for e-commerce purchasing. In this strategy, an agent wanting to purchase a service (the buyer) engages in multiple concurrent bilateral negotiations with a group of service-providing agents (the sellers), where the agents have no prior information about their opponents. The service here involves multiple issues (including price, quality, and other aspects).

  The strategy consists of a *coordinator* component and a number of *negotiation threads*. The coordinator is responsible for coordinating all the threads, deciding and changing the negotiation strategy for each thread, and changing the reservation value (i.e. the lowest utility value of an offer that is considered acceptable by the agent). Each thread represents the interaction with a particular seller using the alternating offers protocol. Also, it is responsible for deciding what offer to accept and what counter-offers to send to the seller, and it reports their status back to the coordinator. In addition, threads obtain their preferences from the buyer agent, including the range of acceptable values for negotiation, the reservation value, and the deadline of

the negotiation. The strategy classifies seller agents into two types: conceder (an agent that is willing to concede during negotiation), and non-conceder (an agent that adopts a tough stance).

In [79], the authors extended the original strategy [78] by improving both the initial strategy selection using the buyer's belief about the sellers. In the extended strategy, the coordinator determines the negotiation strategy using three elements: (a) the *probability distribution of the seller*, determining whether the seller is a conceder or a non-conceder; the (b) *percentage of success matrix*, describing the chance of reaching an agreement when the buyer agent applies a specific strategy to negotiate with a particular type of seller; and (c) the *payoff matrix*, measuring the average utility value of an agreement reached in similar situations. The success and payoff matrices are initially set to a common value (i.e. domain information) and updated after the negotiation finishes. For example, in a *percentage of success matrix*, when the agent negotiates with a non-conceder seller using a tough strategy, the average chance of reaching an agreement is 15%. An example of a *payoff matrix* is an average utility of agreement of 0.7 when the buyer agent uses a tough strategy to negotiate with the conceder seller.

The negotiation process begins with the coordinator, who calculates the probability of the first seller (randomly picked) being of a specific type. In the next step, the coordinator calculates the expected utility of applying different strategies (conceder, linear and tough) to negotiate with a particular seller (driven by the probability distribution function); it selects the strategy for the first thread that maximises the expected utility. Next, the coordinator uses a Bayesian update function [118] to update the probability distribution of the seller type and uses it for the second seller. The process terminates when the coordinator allocates a strategy for each thread. The threads then begin to exchange proposals with the sellers.

After each negotiation round (exchanging one proposal and one counter-

proposal), the threads report their negotiation status to the coordinator. The coordinator changes the strategies based on its beliefs regarding the opponent type. During the analysis time (the time after the first proposal and before the end of the deadline), the coordinator reclassifies the sellers' types (conceder or non-conceder) based on the previous utility value of their proposals. Thus, if the previous proposal offered a value over a threshold value (where the threshold value is set to concessionary behaviour), then the seller is considered a conceder. In the next step, the coordinator will update the strategies of the threads that changed their seller's type. The coordinator selects a new strategy that provides the maximum expected utility using the percentage of success matrix, the pay-off matrix, and the seller's new probability of being a conceder or non-conceder. In the empirical evaluation, the authors found that changing the threads' strategies during run-time led to 30-40% successful negotiations, a better result than having the threads retain their initial strategies.

While this strategy is very relevant to our study since it models concurrent negotiations, it is not directly applicable to our negotiation setting. This is due to its strict assumptions about the knowledge of the opponent types, negotiation success and payoff matrices before the negotiation and negotiating in discrete time. In addition, the strategy is based on a Bayesian learning approach requiring prior knowledge about the environment as well as the opponents. Moreover, it takes into consideration only two types of sellers (i.e. conceder or non-conceder) and three types of buyer strategies. There are no indications of how frequently the coordinator should reclassify the sellers.

In the strategy described above, the coordinator deals with the threads' status separately. In other words, the coordinator should deal with the threads' status cumulatively to decide whether to change the threads' strategy or not. Our strategy is more open than that of Nguyen and Jennings as it does not

deal only with two types of sellers, agents can come and go during negotiation and it does not require prior information about the opponents and the environment in the form of success and payoff matrices. In our experiments (Chapter 6), we will not compare our strategy with the Nguyen and Jennings since we will compare our work with Williams *et al.* [115], which outperforms the Nguyen and Jennings strategy.

- An *et al.* [6] proposed a strategy for deciding what proposal to make and when to make it, in multiple concurrent negotiations. They propose four decision functions supporting agents' proposal generations. These decision functions are based on the available negotiation time, trading opponents' strategies, the negotiation situations of the different negotiation threads, and the other competitor buyers.

  The limitation of this strategy is that compared with our strategy, the negotiation situations of different negotiation threads are calculated according to the minimum offer received from all sellers, while in our case, we analyse each negotiation thread based on how the negotiation with the opponent progresses. Also, in the An *et al.* strategy, there are no rules to decide when to accept an offer, but in our strategy there are. Moreover, their strategy does not employ a concurrent protocol, as we do. Furthermore, An *et al.* did not benchmark their strategy against other state-of-the-art strategies, while in our strategy we do.

- Li *et al.* [59] presented a strategy for bilateral negotiations about one issue (i.e. price). Their strategy is composed of sub-strategies about single-threaded negotiations where one buyer and one seller are assumed; synchronised multi-threaded negotiations where there are multiple threads in the buyer account for negotiating with different sellers; and dynamic multi-threaded negotiations where the uncertainty of new sellers joining in the future is factored in to the buyer's strategy. The sub-strategy that is most related to our setting is that of dynamic multi-threaded negotiations in the

presence of other opponents.

When an agent of the type described by Li *et al.* [59] negotiates in the dynamic multi-threaded negotiations strategy, it makes adjustments to its reservation price when a new opponent arrives. Moreover, in dynamic multi-threaded negotiations, the agent has the ability to set the reservation utility and thus change the negotiation strategy proactively by predicting the arrival and impact of future outside opponents. The strategy is based on the notion of expected utility. Expected utility is calculated based on the arrival probability and the seller resource value distribution. The arrival probability is the probability that the buyer finds an alternative and launches a new negotiation thread.

One limitation of the strategy is that it adopts only the basic time-dependent negotiation strategy, viz., the concession rate of the offer only depends on the negotiation time. Furthermore, the work does not provide implementation details on how the authors set up their empirical evaluation. In addition, the strategy assumes the availability of prior information about opponents (e.g. the estimated reservation price of an opponent), which is again not appropriate for practical negotiation settings like ours.

- Ponka [83] proposed a concurrent strategy that consists of three layers. The first is the negotiator layer, where each negotiator engages in a negotiation with a single provider agent over a single service, and is independent from other negotiators of the same agent. The second is the controller layer, which manages a group of negotiators that negotiate about the same service by determining the number of negotiations running at the same time and selecting the opponents for each negotiator. It decides when to accept an offer, when to start a new negotiation and with whom to negotiate. In addition, each negotiator reports its status to the controller after each round. The third is the coordinator layer, which manages all the agent controllers and obtains a report of their progress in every turn.

The limitation of this strategy is that the buyer knows the reservation price of the seller. Ponka's model focuses on how cancellation policies affect the behaviour of the agent. In addition, the coordinator waits until it receives the round's information from the entire group of negotiators and then advises an overall strategy, taking into account the developments in all the negotiations. In addition it benchmarks the strategy against a random strategy.

- Kolomvatsos *et al.* [52] adopted Particle Swarm Optimization (PSO) and Kernel Density Estimation (KDE) for each thread in the concurrent negotiation to find the proposed prices in every round of the negotiation. Similarly to our strategy, they focused on the buyer side and they studied concurrent negotiations between a buyer and a set of sellers. In this setting, the buyer utilizes a number of threads, with each thread following a specific strategy that adopts swarm intelligence techniques for achieving the optimal agreement. PSO algorithm is adopted by each thread. What is interesting in their approach is that their strategy requires no central coordination. Their strategy does make some assumptions about the opponents. However, their experimental results focused on the time interval where an agreement is possible. They abstracted away from comparing their results with the current state-of-the-art, as we do.

- Ren *et al.* [91] presented a model for designing a strategy for agents to make different levels of concession during negotiation depending on changes in the environment, in any negotiation strategy (bilateral or multilateral) with any number of issues. The contribution of the model is to extend the market-driven agents (MDAs) [102, 103] to consider both current and expected changes in open and dynamic negotiation environments.

  The limitation of this strategy is that the solution to the environmental change is to change the negotiation strategy. However, the strategy takes into account only adding and/or removing opponents and competitors.

In addition, in the Ren *et al.* strategy, agents negotiate in rounds with a maximum number of agents involved (five). Furthermore, there is no indication of who are the opponents. In the empirical evaluation, agent performance was demonstrated by changing the environment factors; however, the agent has to compare its performance with other agents to prove the accuracy of the strategy.

- Williams *et al.* [115] proposed a negotiation strategy for concurrent negotiations in which the agent negotiates with multiple opponents about one item with multiple issues. This is the most similar existing research with respect to our setting. As with Nguyen and Jennings, the Williams *et al.* strategy has two components: a set of negotiation threads and a coordinator. Each negotiation thread is responsible for two tasks: (a) performing Gaussian process regression to predict the future utility of its opponent and (b) calculating the concession rate for an offer by taking into account the best time and best utility values from the coordinator. The coordinator is responsible for determining the best time and utility for each thread to reach an agreement.

  Williams *et al.* [115] developed two versions of their strategy. The first assumes that each thread has a different utility function from the others, while the second presupposes that all threads should have the same utility function.

  However, there are weaknesses in the Williams *et al.* strategy: (a) both the agent and its opponents have the same deadline (and are aware of each other's), which is not realistic for most practical problems; (b) they model the openness of the market by having agents using the same break-off probability (likelihood of leaving the negotiation) which is a prior knowledge, that is not applicable in real e-markets where buyers and sellers are concurrently negotiating with each other; and (c) their negotiation setting is not completely open since they only allow agents to leave the environment, but

not to enter during run-time. We will use the Williams *et al.* strategy to benchmark our strategy, as it represents the current state-of-the-art in multiple and concurrent bilateral negotiation. We will explain the reasons for choosing this strategy to benchmark at the end of this section.

- An *et al.* [5] presented the design and implementation of agents that negotiate concurrently to allocate multiple resources. The agent specifies the maximum number of reserved offers and what to offer during run time using a heuristic approach. The strategy is calculated based on the following factors: deadline, number of sellers, market competition, multiple resources, and cancellation.

  As in our strategy, they assume that the agent has incomplete information about the environment and the opponent. Also, there is no explanation of how the agent selects actions. The authors' negotiation problem is different from ours since they negotiate multiple resources instead of one. In addition they assume that negotiation is conducted in rounds rather than in real time and the buyer knows the probability distribution of the seller's reservation price. By the same token, Sim [104] and Mansour *et al.* [68] proposed concurrent negotiation strategies to allocate multiple resources. Hence, the An *et al.* , Sim and Mansour *et al.* strategies are not suitable for our negotiation environment because their strategies negotiate to allocate multiple resources while in our strategy we allocate one resource only.

Table 2.1 compares existing work in concurrent negotiation according to the functionalities: (a) Model of Open Market – where the negotiating agent is capable of modelling how sellers/competitors enter and leave the market at any time during negotiation; (b) Self Model – models the progress of ongoing negotiations; (c) Private Deadline – where the buyer's and sellers' deadlines are not known to each other in advance; (d) Explicit Decisions – where clear conditions about when to offer, accept, request-to-reserve, cancel and exit are specified; (e) Different Price Ranges – where the agent is tested against different classes of intersections

Functionality	Nguyen and Jennings [78, 79]	An *et al.* [6]	Li *et al.* [59]	Ponka [83]	Kolomvatsos *et al.* [52]	Ren *et al.* [91]	Williams *et al.* [115]	CONAN
Model of Open Market	✗	✗	✗	✗	✗	✗	✗	✓
Self Model	✗	✗	✗	✗	✗	✗	✗	✓
Private Deadline	✗	✗	✗	✗	✗	✗	✗	✓
Explicit Decisions	✗	✗	✗	✗	✗	✗	✗	✓
Different Price Ranges	✗	✗	✗	✗	✗	✗	✗	✓
Concurrent Protocol	✗	✗	✗	✗	✗	✗	✓	✓
State-of-the-art Opponents	✗	✗	✗	✗	✗	✗	✓	✓
Performance Comparison	✗	✗	✗	✗	✗	✗	✓	✓
Asynchronous Negotiation	✗	✓	✗	✗	✗	✗	✗	✓
Environment Model	✗	✓	✓	✗	✗	✓	✗	✓
Incomplete Information	✗	✓	✗	✗	✗	✓	✓	✓
Continuous Time	✗	✓	✗	✗	✓	✗	✓	✓
Opponent Model	✓	✗	✗	✗	✓	✗	✓	✗
Multi-issue Negotiation	✓	✗	✗	✗	✗	✓	✓	✗

Table 2.1: Functionality comparison of the existing work in concurrent negotiation.

between the price ranges (i.e. agreement zones) between the buyers and sellers (for more details see Section 3.2); (f) Concurrent Protocol – where the agent uses a special protocol that has actions to deal with concurrent negotiations; (g) State-of-the-art Opponents – where the agent negotiates with other opponent agents (i.e. sellers) that exist in the literature; (h) Performance Comparison – where the buyer agent benchmarks other strategies in the literature; (i) Asynchronous Negotiation – where the agent does not have to wait for all the sellers' counter-offers to reply; (j) Environment Model – where the buyer has a model of the market's environment in terms of the number of sellers, its competitors and other parameters that formulate the market's dynamics; (k) Incomplete Information – where there is no a priori knowledge about the negotiation environment and the opponents; (l) Continuous Time – where real time is used as the negotiation time instead of using the number of negotiation rounds or discrete time; (m) Opponent

Model – models the behaviour of the opponents; and (n) Multi-issue Negotiation – where buyers and sellers negotiate multiple issues like price and warranty.

In order to evaluate the performance of our strategy against the strategies proposed in the literature, we make sure that the benchmark strategy uses a concurrent protocol and a similar practical evaluation setting to ours (Chapter 6) without considering the way the strategy is composed (the way that the agent thinks). For instance, buyer $b1$ (adopts a Time-dependent strategy [29] with protocol actions = offer, accept) and $b2$ (adopts a Relative Tit-For-Tat strategy [29] with protocol actions = offer, accept) are negotiating with two sellers, $s1$ and $s2$, concurrently. At the end of the negotiation, $b1$ gains a higher utility than $b2$. So it is fair to say that $b1$ outperforms $b2$ even though their strategies take different factors into account which in this case are time and opponent model.

Another example is when the same buyers mentioned above ($b1$ and $b2$) have different protocol actions where $b1$ adopts a Time-dependent strategy [29] with protocol actions = offer, accept, cancel and $b2$ adopts a Relative Tit-For-Tat strategy [29] with protocol actions = offer, accept. $b1$ and $b2$ concurrently negotiate with two sellers $s1$ and $s2$. At the end of the negotiation, $b1$ gains a higher utility than $b2$. Here the comparison between $b1$ and $b2$ is unfair since one based its strategy on a certain protocol and the other did not. The rules of the negotiation should be the same for all participants.

We compare our strategy with strategies that have the following settings:

- assume private deadline (this can be easily adaptive in the evaluation where we make the deadline public by assigning the same deadline for both buyers and sellers);
- use a concurrent negotiation protocol which includes the cancel action;
- have state-of-the-art opponents;
- have performance comparison;
- have incomplete information about opponents;

- use continuous time.

Since the negotiation settings of Nguyen and Jennings [78, 79], An *et al.* [6], Li *et al.* [59], Ponka [83], Kolomvatsos *et al.* [52] and Ren *et al.* [91] negotiation settings are totally different to our settings based on the functionalities listed in Table 2.1, it is unfair to benchmark their strategies. However, Williams *et al.* [115] is the state-of-the-art in concurrent negotiation and has all the required settings except the deadline being public. For this reason, in our experimental evaluation (Chapter 6) we set the deadline for Williams *et al.* [115] to public by assigning the same deadline for both buyers and sellers, which implies that Williams *et al.* will assume that the sellers will have the same deadline as theirs. Also, the multi-issue feature in Williams *et al.* will be adapted to single issue negotiation by setting the number of issues in Williams's *et al.* strategy to 1.

The justification of any additional strategies used in our experimental evaluation will be provided in Chapter 6, where we will discuss the rationale of our experimental settings in more detail.

2.6.2 Asynchronous Negotiation

The problem that arises in concurrent negotiation is that the buyer has to wait to receive all the offers from the sellers. However, there are many reasons why the buyer should not wait to receive all the proposals from its opponents. First, the buyer is bounded to a limited deadline, and waiting for all the offers will consume the negotiation time and thus will result in fewer negotiation rounds with less probability of reaching an agreement. Second, due to the increased waiting time the buyer will not be preferred in negotiations with other sellers because the other sellers will assume that buyer will always be late.

An *et al.* [6] propose two strategies to make the negotiation more flexible in synchronization: a *fixed waiting time-based strategy* and a *fixed waiting ratio-based strategy*. A fixed waiting time-based strategy determines the time to wait after an agent receives the first counter-proposal. On the other hand, a fixed waiting ratio-

based strategy determines the number of counter-proposals to wait for after an agent receives the first counter-proposal. However, their synchronization solution depends only on the reaction times of the opponents.

Waiting time should not depend on the response time of opponents only and should depend on the negotiation situation of the agent. So, in our negotiation model, the agents negotiate in an asynchronous manner where the agent replies to the sellers based on how many counter-proposals it receives at the same time. For instance, if an agent receives three counter-proposals at time t and there are seven more sellers still to reply, then the agent will reply to these three proposals first. Thus, the agent does not have to wait for other sellers to produce an offer and replies only to sellers whose offers have been received at the same time.

2.6.3 Cancellation Penalties

In this section we will discuss the penalty (i.e. the amount of money) that the agent has to pay to the opponent if it cancels a previously reserved item. A protocol state that describes the reservation of an item needs another action to form an agreement. An agent can cancel a beneficial existing reservation with one seller and move to a new, more preferred offer with another seller. The reason for our interest in cancellation penalties is to justify the cancel action in our protocol in Chapter 3.

The option to cancel a reservation increases the utility to both negotiation parties and saves computation and time [97]. This is because an agent can offer to reserve an item faster than agree to buy it, so that it can explore other options. However, it has to balance the number of reservations that it makes, since if they do not materialize, they can decrease its utility [83], due to cancellation penalties it will have to pay.

There are many types of cancellation penalties:

- Fixed price penalty: the penalty is a fixed price whenever the agent wants to cancel.

- Percentage of the deal: Nguyen and Jennings [80] extended their negotiation model to deal with situations where the seller can renege on agreements by developing a reservation manager. The main contribution of the model is providing an agent with the ability to reason about reservation and cancellation to intermediate agreements (i.e. agreements that have been reserved between a buyer and many sellers, where the buyer must choose one of them as the final agreement). The reservation manager is responsible for helping the thread to decide whether or not to accept an offer or to renege on a reserved deal (agreement). Thus, when a buyer or a seller decides to break a reserved deal, it has to pay a *de-commitment penalty* to its opponent. The de-commitment penalty is calculated dynamically as a percentage of the deal utility and is also based on the time when the contract is broken, as in Equation 2.1.

  In order for the buyer to avoid the risk associated with reserving only one deal at a time, in which the seller can renege near the deadline, leaving the buyer with a short time to find another deal, the buyer can reserve more than one deal simultaneously. Then, the buyer selects the deal that has the highest utility value. The empirical result draws attention to many points. One of the points illustrates that different penalty levels have different impacts on the performance of the model. Also, the more patient the buyer, the better the deal it will obtain. One of the model's strengths is solving a realistic problem occurring in a real-life environment, so it avoids the limitations of other models [79].

$$Penalty_t = U(\alpha, t_\alpha)(Penalty_0 + \frac{t - t_\alpha}{1 - t_\alpha}(Penalty_{max} - Penalty_0)). \quad (2.1)$$

  where $U(\alpha, t_\alpha)$ is the agent utility of the agreement at time t_α, $Penalty_0$ is the penalty at the agreement time, $Penalty_{max}$ is the penalty at the deadline.

  In addition, An *et al.* [4, 5] have developed an agent which negotiates in an

environment that contains concurrent negotiation and cancellation.

$$Penalty_t = 0.1 * Prc(\alpha)((1-t)/\lambda)^{1/2}. \tag{2.2}$$

Where $Prc(Ag)$ is the price of the agreement α, $\lambda = 6$ and t is the negotiation period normalized between $[0, 1]$.

Also, Williams *et al.* [115] assumed a cancellation penalty $Penalty_t$ based on a percentage of the deal, where:

$$Penalty_t = D * Price_{reserved} \tag{2.3}$$

where D is a percentage of the agent's reserved offer $Price_{reserved}$ where $D \in [0, 1]$.

- Penalty that is decided with the negotiation deal. An *et al.* [3] presented a model that negotiates the price and the cancellation penalty of a resource in cloud computing and shows that using cancellation achieves higher social welfare (i.e. both agents win in the negotiation).

- Penalty that is decided at the time of cancellation an agreement.

Ponka [83] investigated the effect of a cancellation penalty on agent behaviour and utility and found that using different types of cancellation penalties can have a significant positive impact on the agent's utility.

We will adopt the model of a percentage of the deal penalty to enable a fair comparison with the benchmark strategy of Williams *et al.* [115].

2.7 Negotiation Simulation

The multi-agent literature is rich in negotiation models and strategies. However, there has not been much work in the area of simulation for negotiating agents. In this context, we may classify the relevant literature according to:

- *proprietary simulators* – which perform specific, closed (i.e. not open source) experiments that are developed by researchers to evaluate their own agent strategies, e.g. in bilateral negotiations [29, 75], concurrent bilateral negotiations [6, 59, 80, 91] and opponent models [22, 46]. These simulators cannot be used in other negotiation settings since their parameters are fine-tuned and the agent types are fixed. In addition, they are hard to re-implement due to time constraints and limited availability of details about the simulator design.

- *public simulators* – which are generic, usually open-source simulators used to evaluate any agent strategy, as long as it is expressed in the system's specification language, e.g. the state-of-the-art GENIUS negotiation environment [61][1].

We will discuss GENIUS because of its wide use and acceptability as a publicly available simulation tool. Also, we will discuss GOLEM and GOLEMLite as base platforms for our simulator.

2.7.1 GENIUS

GENIUS [61] is a negotiation environment that implements an open architecture for heterogeneous negotiating agents. It provides a testbed for negotiating agents that includes a set of negotiation problems for benchmarking agents, a library of negotiation strategies, and analytical tools to evaluate an agent's performance.

To verify the efficacy of GENIUS, the system was used by 65 students, who were each required to design an automated agent for different negotiation tasks. Their agents were evaluated and both quantitative and qualitative results were gathered. These results suggested that GENIUS helps and supports the design process involved in producing an automated negotiator (from the initial design, through to the evaluation of the agent, and redesign and improvements) based on its performance.

[1] http://ii.tudelft.nl/genius/

To the best of our knowledge, GENIUS was used only for evaluating bilateral negotiation, especially of agents in the automated negotiating agents competition (ANAC) [37]. In this context, Williams *et al.* extended GENIUS to provide support for concurrent negotiations [115]. However, this extension addressed a specific experimental setup and was not accessible publicly. Moreover, we are not aware of any work that evaluates the robustness and scalability of GENIUS when using a large number of agents.

2.7.2 The GOLEM Agent Platform

GOLEM[2] is a logic-based agent platform developed to represent agent environments that evolve over time [18, 19]. It provides a middleware that agent developers can use to build multi-agent systems using both Java and Prolog. The platform has been used to deploy agents in a number of practical applications, from ambient intelligence [21, 25] and service negotiation in grid computing [20], to diabetes monitoring and management [50]. As shown in Figure 2.6, a GOLEM environment consists of three main components: containers, agents and objects.

- *Containers* – are logical entities representing a subset of the agent environment and mediating the interactions between agents and objects situated in it. Such interactions are governed by the container's physics, a component that prescribes how the state of the container changes as a result of agent actuators performing actions, including how actions and their effects are perceived by agent sensors. Using containers, an agent environment can be distributed over a network, allowing agents to perceive and interact with entities that are logically near, but physically distributed somewhere else. Containers may also contain sub-containers. A connector component attached to the container allows agents to communicate with each other.

- *Agents* – are cognitive entities that can sense the environment, reason about it and interact with other agents as well as objects. An agent is composed

[2] http://golem.cs.rhul.ac.uk/

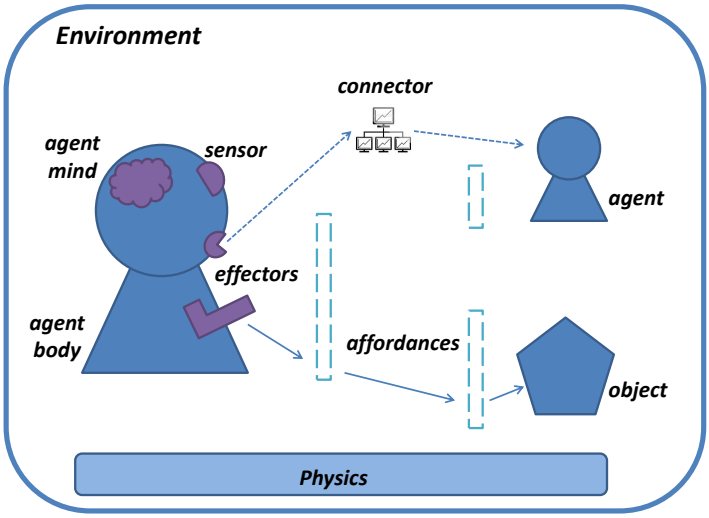

Figure 2.6: GOLEM environment.

of an agent body and an agent mind. The body allows physical interaction with the environment via sensors and effectors, while the mind processes perceived knowledge to decide how the agent should act. An agent's use of body and mind was introduced in Section 4.1.

- *Objects* – are entities situated in the environment that react once agents act upon them.

In GOLEM the way entities make themselves present in the environment is described via the concept of *affordances*. Affordances are defined with ontologies and enable agents to perceive these entities in a systematic manner, without the need for a directory facilitator agent (see [18] for more details).

Agent communication Containers are connected to each other via components called *connectors*. A connector is an abstract service that hides away the low-level details of message transportation. Connectors are registered to containers in order to enable agent communication, implemented by combining Prolog and Java. When an agent needs to send a message to another agent, it simply produces a

specific action described by the keyword *envelope*. Once the agent produces such an action, the connector that is registered with the agent's container picks it up, and transfers it to the recipient agent's connector. A variation of the send message is also used to send serialised objects between containers (e.g. to move an agent to a different physical machine).

2.7.3 GOLEMLite

GOLEMLite [73] is a Java library that maintains the key GOLEM concepts such as the container and agents with body, mind, sensors and effectors. However, in GOLEMLite there is no support for objects and applications can be deployed in one container with a stripped-down physics component that supports communicative actions between agents only (i.e. the system does not support physical actions). In addition, GOLEMLite introduces the notion of *infrastructure agents* that allow us to develop agents that control the environment of the participant's agents, which will be discussed in detail in Section 5.2.2.

Our initial intention was to extend GENIUS with extra functionality for concurrent negotiations as well as realistic e-market parameters. However, the shortcomings of GENIUS as described above discouraged us from trying to extend the system to serve our experimental purposes. We thus decided to develop Recon (Chapter 5) on top of the GOLEM platform as it already supports agent deployment in different settings, including support for specifying declarative and/or imperative strategy negotiating agents. However, GOLEM does not have the specialised tools to support negotiation. In addition, we will use the GOLEMLite library to support our infrastructure agents in Recon.

2.8 Limitations of Existing Work

In general, we consider the limitations that exist in concurrent negotiation agent models. More specifically, we consider the shortcomings of these models in terms of the architectures that they use, their associated protocols and strategies, how

these strategies are implemented, and how the simulations are performed to obtain experimental results. In each of the above dimensions we discuss both the limitations and why overcoming these limitations is important.

- *Architectures:* existing negotiation architectures are for bilateral negotiation only, with protocol- and strategy-dependent features, without explicitly representing the main components of the concurrent negotiation and how these components interact with each other. In addition, such architectures fail to describe the information exchange between the components of the architecture. Developing architectures for concurrent negotiation that is protocol- and strategy-independent is important because it provides adaptable designs that can be used by the agent developer to implement any negotiator concurrent strategies, thus saving the cost of developing the architecture from scratch.

- *Protocols:* since we are negotiating concurrently, the existing protocols are not sufficient to handle the complexity of concurrent negotiations due to their limited actions. Adding more negotiation actions will result in the agent taking advantage of the concurrent negotiation situation and thus increase its negotiation utility.

- *Strategic context:* existing negotiation strategies abstract away from the dynamicity of the market and the progress of ongoing negotiations, thus ignoring information that may decrease the possibility of achieving an agreement and losing opportunities to add value to the agent's utility function. Also, they often make strong assumptions about the domain, e.g. that deadlines are public, thus constraining their applicability in a variety of practical negotiation settings. In addition, none of the concurrent negotiation literature has tackled the problem of different agreement zones during negotiation (for more details see Section 3.2). Moreover, current models lack explanations of how to choose negotiation actions, so that the agent knows when to request-

to-reserve, cancel and exit the negotiation. Furthermore, some of the existing work neglects the process of modelling the negotiation environment, constraining the negotiation information to be complete, and assumes the negotiation time to be discrete, which places such negotiation models far from real-time applications. By overcoming these limitations we can make negotiation strategies more practical.

- *Strategy implementation:* most of the existing agent negotiation strategies, to the best of our knowledge, are implemented procedurally using the constructs of an imperative language, like Java, and are thus accessible only to the developers of the system. Thus, the agent strategy becomes often difficult to explain to a user who might need to understand why the agent has made the specific offers for a particular negotiation. Developing a declarative strategy can help a user understand the rules of the strategy better and the agent to justify better each action taken in the negotiation.

- *Simulations:* existing negotiation simulations only support bilateral negotiation. In addition, most negotiation agent development platforms, such as GENIUS, only support imperative (e.g. Java) agents. It is useful to incorporate declarative strategies that allow developers to specify strategies that can be transparent to a human user, in that the agent can explain why it has taken certain actions during a negotiation.

- *Experimentation:* most existing approaches have not conducted performance comparisons with any benchmark strategies. Benchmarking other strategies will help to advance the state-of-the-art in the concurrent negotiation field. Furthermore, most of the existing concurrent negotiation strategies do not use any state-of-the-art agents as opponents. Using state-of-the-art opponents will make the negotiation between agents closer to real life situations where the opponents' strategies differ from each other.

In summary, to the best of our knowledge, there is no adaptive model for concurrent bilateral negotiations in a dynamic environment overcoming all the limitations listed in Table 2.1, which makes our solution relevant and useful in the agent negotiation field.

2.9 Summary

In this chapter, we presented an overview of the current negotiation literature. Since there are multiple ways to present automated negotiation models, we started the chapter by presenting what we mean by a negotiation model. Then we introduced the attributes of the resource under negotiation. Within our review, we presented our reasons for choosing the concurrent bilateral negotiation (single issue) as our model. We illustrated the current state-of-the-art in negotiation architectures and protocols and why we chose to extend the alternating offers protocol to govern actions in our negotiation models.

We organized negotiation strategies based on game theory, argumentation and heuristic methods. After justifying our choice of a heuristic mechanism to develop our negotiation strategy, we discussed the literature of heuristic strategies. Then, we discussed in detail a wide range of negotiation models based on concurrent bilateral approaches with a focus on the limitations of the concurrent bilateral negotiation area.

As our model is based on heuristic strategies, we presented the current state-of-the-art negotiation simulation and its limitations. At the end of the chapter, we provided a brief overview of all the limitations in the existing concurrent negotiation models. To address these limitations, we present next (Chapter 3) our negotiation model, in terms of its associated agent architecture, protocol and strategy.

Chapter 3

Adaptive Negotiation Model

In this chapter, we will present our concurrent negotiation model, which includes an agent architecture, an agent strategy and the negotiation protocol that the strategy assumes. Our concurrent negotiation model seeks to satisfy the thesis' aim in general and Objectives 1, 2 and 3 in particular, viz., to provide the agent that uses the model with the ability to make decisions on how to negotiate concurrently with other agents on behalf of its user in open and dynamic e-markets for a resource allocation to reach a single agreement that maximises the agent's utility.

In the remainder of this chapter, we will begin by introducing the concurrent negotiation architecture and its components (Section 3.1). Next, we will provide in Section 3.2 a detailed explanation of our negotiation settings, including the number of participants and the recourse under negotiation. Then we illustrate the concurrent negotiation protocol and its actions (Section 3.3). After that, CONAN, our negotiation strategy, will be discussed in Section 3.4. The discussion will explain the heuristics developed to generate an offer, choose the negotiation actions and calculate the cancellation penalties. Section 3.5 analyses the properties of CONAN. Finally, we summarise the chapter in Section 3.6.

3.1 Concurrent Negotiating Agent Architecture

Most descriptions of agent architecture in the literature focus on the general architecture of negotiation [10, 30], protocol or strategy-dependent factors [92], or are related to agent-human negotiation architecture [27]. However, this study focuses on developing an agent architecture that can be used to deploy any concurrent negotiation agent in general, and our negotiating agent in particular. Our approach may be used by any agent developer to implement their concurrent negotiation strategies.

We base our negotiating agents on the KGP model of agency [107]. In this model an agent consists of (a) the knowledge K that the agent uses to reason and act in the environment in which it is situated; (b) the goals G that represent what states of the environment the agent wishes to achieve; and (c) the plans P that represent how the goals of the agent can be achieved as a series of actions and/or further sub-goals.

The reason for choosing the KGP model is the need for (a) declarative strategies to aid the explanation of decisions taken by the agent, hence the choice of KGP as a declarative agent model built using logic programming tools and technologies; (b) the ability of our model to deal with real time, as required in practical negotiation, hence KGP supports temporal reasoning using the Event Calculus (discussed in Chapter 4); and (c) a systematic way to deal with reasoning complexities, hence KGP is appropriate for negotiation.

The following is a brief summary of the KGP model components for agents (formal details can be found in [36]):

- mental state, which holds the knowledge base, the goals and the plans for the agent;
- reasoning capabilities, in particular supporting planning, temporal reasoning, reactivity and goal decisions;
- sensing capability, which allows the agent to observe whether properties of

the agent's environment hold; and to perceive the execution of actions by other agents;

- transition rules, defining how the state of the agent changes;
- selection functions, which provide appropriate inputs to the transitions;
- a cycle theory, which decides the sequence for applying the transitions and when they should be applied.

For negotiation we revisit KGP, in particular, previous work with the KGP as it has been applied to decision-making [36] to equip it for decision-making in multiple and concurrent negotiations. More specifically, we revisit KGP by: (a) allowing the explicit representation of markets via the introduction of an environment model; (b) profiling other participating agents by introducing an opponent model; (c) maintaining a representation of self; (d) revising the structure of the current state component; (e) integrating a strategy model; (f) deleting the planning capability; and (g) restructuring the KGP components.

The revised version is shown in Figure 3.1 and consists of five components: Domain Knowledge, Current State, Physical Capabilities, Cognitive Capabilities and Control. The following is a brief description of each component in the revisited model we construct.

3.1.1 Domain Knowledge

This represents generic and dynamically changing knowledge about the negotiating application at hand. It consists of five sub-components: a *Strategy Model* detailing how the agent selects actions in the application domain; a *Goal Model* outlining which goals the agent can achieve and how they can be achieved using the strategies; an *Environment Model* representing knowledge about classes of different types of environments and the protocols they require; an *Opponent Model* detailing classes of different opponents; and a *Self Model* representing information about the agent itself and its ongoing negotiations. For simplicity, in the rest of

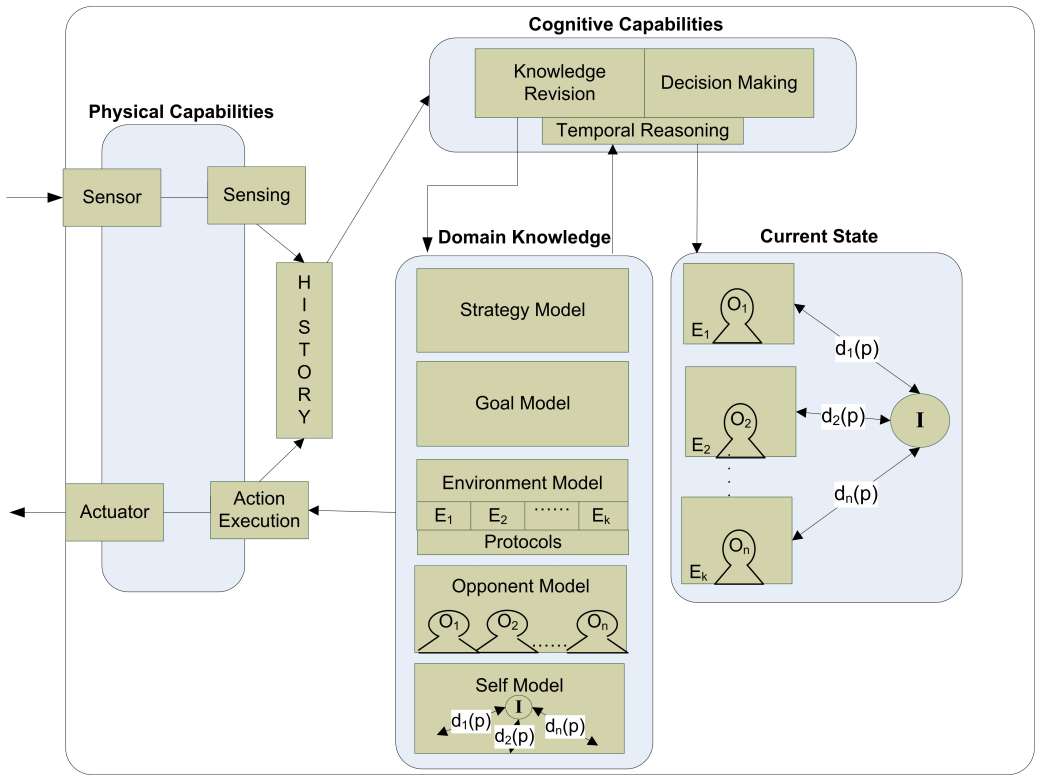

Figure 3.1: Architecture of negotiating agent I that is interacting concurrently in different sub-environments $E_1, E_2, \ldots, E_k$ with opponents $O_1, O_2, \ldots O_n$.

the thesis we will assume that the agent will model only one environment E where $E = E_1$.

3.1.2 Current State

This contains instances of the ongoing negotiations in terms of the participants for each negotiation, the protocol used for each negotiation, and the progress made in these so far.

3.1.3 Physical Capabilities

These situate the agent in an environment by connecting the agent's sensors and effectors to its internal state. They consist of two sub-components: a *Sensing capability* that captures environment information via the sensors; and an *Action*

Execution capability that performs the actions in the environment via the actuators. Both capabilities operate on the *History* of the agent, another component that holds the experience of the agent represented as a series of events that have happened, either in the form of observations (events from the Sensing capability) or actions (events from the Action Execution capability).

3.1.4 Cognitive Capabilities

These allow the agent to reason about the negotiation and take decisions. This component consists of three sub-components: *Decision Making* is responsible for evaluating the *Current State*, and uses *Domain Knowledge* to decide what to do next; *Knowledge Revision* updates the *Domain Knowledge* component either through simple revisions or through learning over time; and *Temporal Reasoning* supports changes in the *Domain Knowledge* due to events happening in the *History* of the agent.

3.1.5 Control

This component details how the capabilities are being invoked and under what circumstances, represented by the outer container and the arrows in Figure 3.1. The typical control cycle is an instance of the more general framework described in [51]. It involves the agent sensing a set of events in the environment, updating its history (which in turn causes a revision of the knowledge and the current state), then making a decision to act. The action is executed in the environment and recorded in the history.

However the protocol and the strategy need to know what information will be known or unknown to the agent and there must be a clear formalization of the negotiation environment and the participants, which will be described in the next section.

3.2 Concurrent Negotiation Setting

We consider open, dynamic and complex environments for concurrent bilateral negotiations. We assume that a market environment contains a set of buyers B where $|B| = n$ ($n \geq 1$) and a set of sellers $S = \{s_1, \ldots, s_m\}$ ($m \geq 1$) that wish to negotiate over a set of resources $R = \{r_1, \ldots, r_l\}$ ($l \geq 1$) using a protocol p. We develop a strategy for a buyer agent $b \in B$, who negotiates concurrently with multiple sellers over a resource $r \in R$ using the protocol p. Negotiation advances in real time t ($t \in \mathbb{R}^+$) where agents take decisions to make offers. At any time t during a negotiation, we identify the following information.

- $Seller_b^t$ is the set of seller agents negotiating with b at time t, where $Seller_b^t \subseteq S$.
- C_b^t is the set of competitor agents for b at time t, where $C_b^t \subseteq B \setminus \{b\}$.
- The negotiation starts at time T_s. Once started, the negotiation can potentially last for time T_b, which is the maximum duration of time that b can negotiate for. We also refer to T_e as the deadline of negotiation for b, where $T_e = T_s + T_b$. In the worst case, at T_e b must terminate the negotiation. We assume that b does not know the sellers' deadlines, and that the sellers do not know T_b.
- b has $\left|Seller_b^t\right|$ negotiation threads, one for each seller $s_i \in Seller_b^t$.
- $\delta^t_{ag_i \to ag_j}$ is a negotiation action from ag_i communicated to ag_j at time t, where either ($ag_i = b \ \wedge \ ag_j \in Seller_b^t$) or ($ag_i \in Seller_b^t \ \wedge \ ag_j = b$), and $\delta \in \{$*offer(x), accept, request-to-reserve, cancel, exit*$\}$ (will be explained next in Section 3.3). Note that $offer(x)$ is a term that contains the price x of the resource under negotiation at time t. The buyer b and seller s_i will generate new offers if their previous offers are not accepted. The value x of the next offer is determined by their negotiation strategy.

- IP_b and RP_b represent the initial and reservation prices of buyer b, while IP_{s_i} and RP_{s_i} represent the initial and reservation prices of seller s_i. In the literature, the intersection between the price ranges (i.e. agreement zones) $[IP_b, RP_b]$ and $[RP_{s_i}, IP_{s_i}]$ is always implicitly assumed to be 100% of $[IP_b, RP_b]$, a very difficult assumption to make when dealing with practical applications. So, in order to make our setting more realistic, we adopt three classes of intersections for the aforementioned price ranges: 10%, 60% and 100%. The reason for exploring different agreement zones is to show how negotiation can move the positions of the agents closer to the areas of the zones where there is agreement and thus improve their utility. We use these three different classes of agreements zones for simplicity in order to test whether this can be verified experimentally for situations where there is roughly speaking little space for agreement (10%), plenty of space for agreement (60%) and no space for disagreement (100%). We chose 100% as the class previously used in the literature, 10% as the lowest level, and a class between 100% and 10% which was 60%. However, we do make the assumption that at any time t if $\forall s_i \in Seller_b^t$ $(IP_b \geq IP_{s_i})$ holds, then there is no need to negotiate. This is a much softer assumption to make.

- $EG_b \in [0, 1]$ represents the eagerness of buyer b, viz., specifying the willingness of b to obtain a resource; this is specified by the user. For instance, if the user needs to buy a laptop to use it in his project in the next week, then the user has a high eagerness to buy a laptop. On the other hand, if the user needs to move to a house in the next two years, then he has a low eagerness to buy a house.

- $U_b(x) : \mathbb{R}^+ \rightarrow [0, 1]$ is the utility function that agent b uses to determine the utility of an offer x made either by b or s_i as follows:

$$U_b(x) = (RP_b - x)/(RP_b - IP_b) \tag{3.1}$$

3.3 Concurrent Negotiation Protocol

Since our focus is on a concurrent negotiation setting where a buyer agent b engages in multiple bilateral negotiations with multiple sellers $s_i \in Seller_b^t$ in order to acquire a single resource, we need a protocol that determines rules of negotiation that are appropriate for concurrent negotiation. We contribute to the state of the art by extending the well-known alternating protocol with additional actions (*request-to-reserve*, *reserve*, *cancel* and *exit*), thus providing the necessary flexibility that was not always possible in previous work with limited actions (*offer(x) and accept*).

3.3.1 Our Negotiation Protocol

In order to decide which protocol would be followed in our negotiation market, we experimented with the alternating offers protocol [95] because it is simple and widely used. However, as discussed in Section 2.4.4, the alternating offers protocol is not able to handle the complexity of concurrent negotiations due to its limited actions (i.e. *offer(x), accept* and *reject*). We extended the protocol to allow for the additional actions *request-to-reserve*, *reserve*, *cancel* and *exit*, as summarised in Figure 3.2. Given that the negotiation time is t:

Where the different elements of the diagram are defined as follows: (1) b represents the buyer agent; (2) s_i represent the seller agent; (3) the arrow represents the name of the agent and the action; (4) the semi-rectangle represents the state of the negotiation; and (5) the circles represent the initial and final states of the negotiation.

- *offer(x)*: describes an offer made either by b or s_i, proposing price x for resource r.
- *request-to-reserve*: provides b with the opportunity to hold a preferred offer from a seller s_i until a better offer is made by another seller s_j. b can send a request-to-reserve to s_i at any time t if b receives an accept from s_i or b

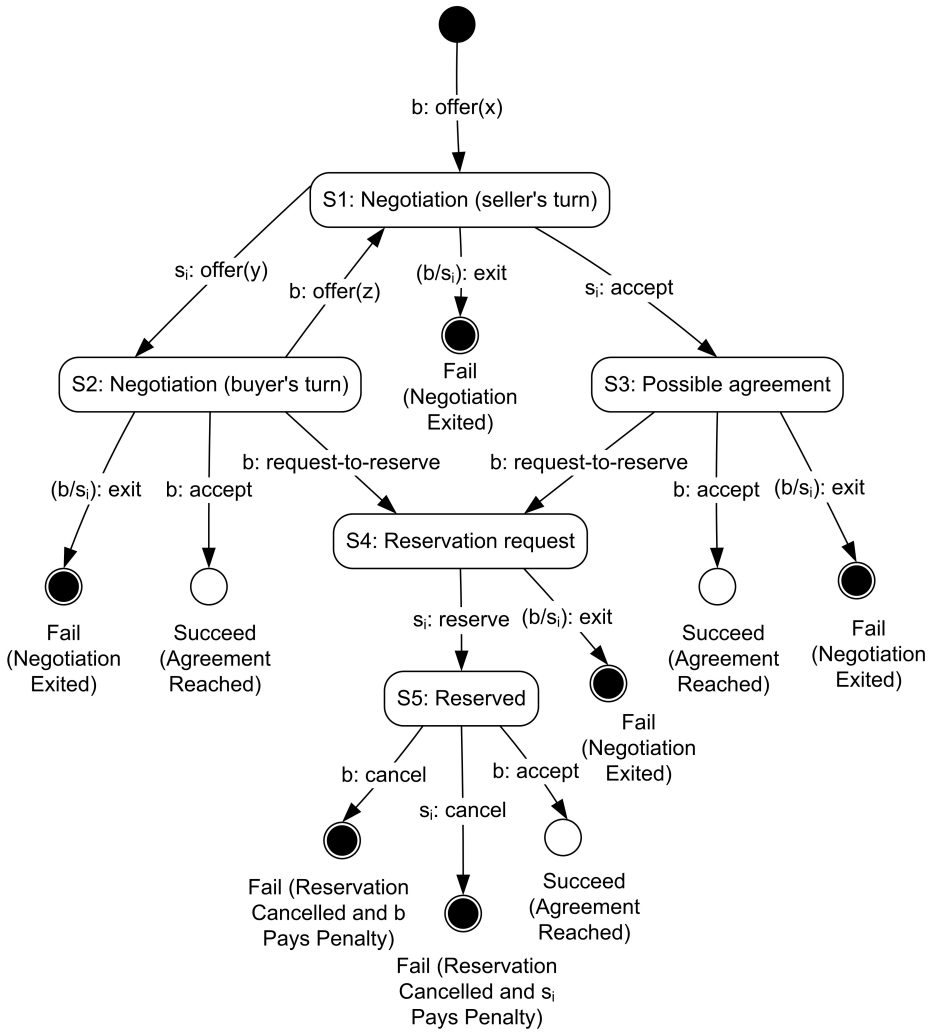

Figure 3.2: Concurrent Alternating Offers State Diagram.

receives an offer from s_i.

- *reserve*: provides s_i with the opportunity to agree on holding a preferred offer from a buyer b. s_i can send a reserve action to b if s_i receives a request-to-reserve action from b.

- *cancel*: allows b or s_i or both to cancel their reserved offers, which means that they retract or withdraw from existing reserved agreements. The agent who cancel the reserved offer pays a penalty $Penalty_{b,s_i}$ to the other agent

involved in the canceled agreement, to ensure fairness and avoid unnecessary cancellations. We will discuss later how to deal with penalties in Section 3.4.2.

- *accept*: implies that b and s_i reach an agreement about the resource r under negotiation being sold at the price of the most current offer with respect to time t. b can send an accept to s_i at any time t if: (a) b reaches its deadline T_b and has a reserved offer with s_i; (b) s_i accepts an offer from b; and (c) b accepts an offer from s_i. In addition, s_i can send an accept for any offer made by b. The negotiation terminates with an agreement only when b sends an accept.

- *exit*: allows b or s_i to withdraw from the course of negotiation at any time, even without notifying other agents. This implies that the negotiation between b and s_i has failed.

Our protocol will be used by our buyer to communicate with multiple sellers concurrently by running the protocol bilaterally at the same time between the buyer and each seller. The negotiation starts when the buyer sends an offer to a seller and then waits for the seller's response. As soon as it receives a response from the seller, the buyer will evaluate the seller's offer based on its strategy. If the buyer does not agree on the offer, it will send a counter-offer. The buyer and the seller will continue sending and receiving offers and counter-offers until either the buyer or the seller quit the negotiation, the offer is reserved, or they reach an agreement. If the seller likes the buyer's offer it will send an accept action and wait for the buyer to:

(a) accept the seller's accept, in which case the negotiation terminates with an agreement. The reason for this is to avoid the situation where b receives accepts from two different sellers;

(b) request-to-reserve to the seller's accept, which implies that the buyer and seller

move to the reservation request state and the buyer will continue to search for other better offers; or

(c) exit from the negotiation with the seller and the negotiation will be considered to have failed.

Likewise, if the buyer likes the seller's offer, it can either:

(1) accept the seller's offer and the negotiation terminates with an agreement; or

(2) request-to-reserve to the seller's offer, so the buyer and seller move to the reservation request state and the buyer will continue to search for other better offers.

In the reservation request state, in cases (b) or (2), the following may happen:

(i) the seller exits from the reservation request state without paying any penalty. This move would be allowed only when the seller does not want to agree to the buyer's request-to-reserve action, and sends an exit;

(ii) the seller sends a reserve action which means that the seller agrees to reserve the item (i.e. accepts the request-to-reserve action);

(iii) the buyer or the seller cancel the reservation state, which results in a penalty being paid by the participant who initiates the cancel action; or

(iv) the buyer can accept the reserved offer and the negotiation succeeds with an agreement.

3.3.2 Overall Negotiation Protocol

This protocol regulates the communication between the agent and the e-market. The overall negotiation, as described in Figure 3.3, starts when the agent b enters the e-market and finishes either when:

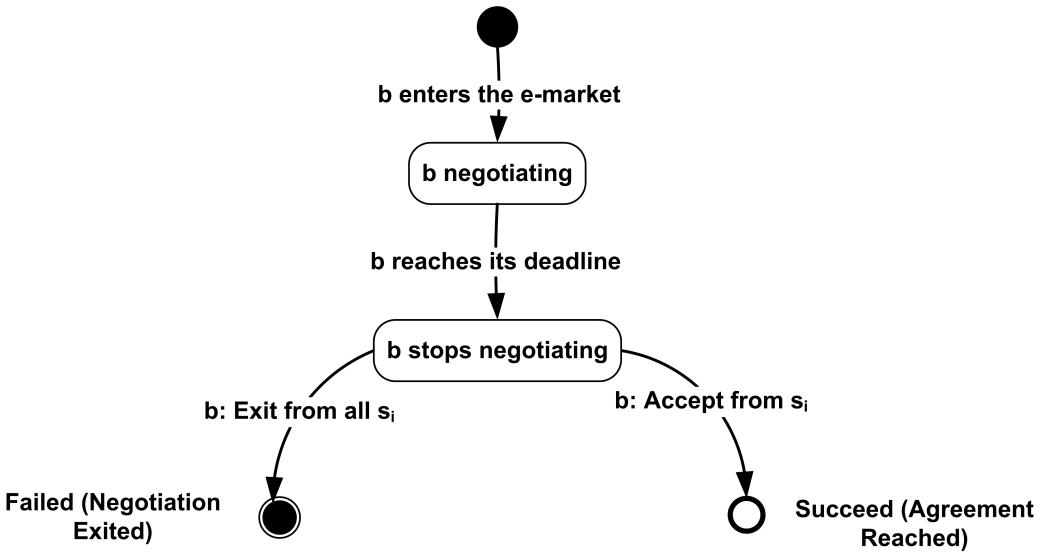

Figure 3.3: Overall Negotiation State Diagram.

E1. b accepts an offer from a seller s_i, thus, agreement is reached and the negotiation succeeds; or

E2. b exits from all bilateral negotiations with all sellers; thus, no agreement was reached, b exits the e-market and the negotiation fails.

After defining this negotiation protocol, the agent needs a strategy to adopt in order to make decisions about which action from the protocol to send to its opponents at each point in the negotiation. Hence, our strategy is protocol-dependent, which means it needs to know what actions are allowed by the protocol to design the negotiation strategy. We make an assumption here that our agent will be *trustworthy*, which implies that the agent will follow the protocol and will not breach it any time during negotiation.

3.4 CONAN: a heuristic-based Agent Strategy

We study the development of heuristic negotiation strategies with an emphasis on the buyer agent in a concurrent negotiation market setting. The existing literature [79, 115] provides answers only about when to offer or accept and what to offer,

but does not explicitly represent reservation and cancellation offers in a concurrent negotiation dialogue. In addition, offers are computed using an opponent's model and do not take into account the environmental and/or self models [79, 115]. Moreover, some existing work assumes complete and certain information about the negotiation environment and the opponents [79]. In most approaches the agent strategy is an isolated component without clear illustration of how the model is designed and how different components interact with each other [79, 115]. To develop an adaptive negotiation decision-making model in a dynamic environment, we must support an agent to deal with an enormous number of environmental and attitude changes at the same time. To address this issue, we develop the CONAN strategy. Since our buyer agent needs to acquire a resource when negotiating with multiple sellers at the same time, the agent will use CONAN to decide (1) what actions to take during negotiation and when to take them; and (2) if the action is an offer, then how to generate an offer. Algorithm 1 gives an overview of CONAN.

CONAN takes T_b, IP_b and RP_b as an input from the agent user and returns either an agreement or no agreement to allocate a resource as output. CONAN starts by initializing its deadline, then calculates the concession rate CR_{t,s_i}, generates the next offer and decides which action to proceed with for each seller s_i. The implementation of Algorithm 1 will be described in the next chapter. However, the designs of each heuristic are discussed in the sub-sections below.

3.4.1 Heuristics to Calculate the Concession Rate

CONAN generates offers $Offer_{t,s_i}$ at time t for seller s_i based on the following equation [29]:

$$Offer_{t,s_i} = IP_b + (RP_b - IP_b) * CR_{t,s_i} \tag{3.2}$$

where

- IP_b: is the initial price;
- RP_b: is the reservation price;

Algorithm 1: CONAN

Input: T_b: negotiation duration,

IP_b: buyer's initial price,

RP_b: buyer's reservation price.

Output: offer (offer > 0 if agreement, offer $= 0$ if no agreement)

$T_e = T_s + T_b$

Repeat

  Get active sellers $Seller_b^t$

  Calculate the CR_{t,s_i} for each seller $s_i \in Seller_b^t$

  Generate b's next offer for each s_i

  Evaluate if $\delta_{b \to s_i}^t$ = exit for each s_i (Section 3.4.3)

  Evaluate if $\delta_{b \to s_i}^t$ = request-to-reserve for each s_i (Section 3.4.3)

  Evaluate if $\delta_{b \to s_i}^t$ = cancel for each s_i (Section 3.4.3)

  Evaluate if $\delta_{b \to s_i}^t$ = accept for each s_i (Section 3.4.3)

  Evaluate if $\delta_{b \to s_i}^t$ = offer for each s_i (Section 3.4.3)

  Send $\delta_{b \to s_i}^t$ for each s_i

Until $t = T_e \quad || \quad \delta_{b \to s_i}^t = accept$ for one s_i

if $\delta_{b \to s_i}^t = accept$ **then**

  Return offer (agreement)

else

  Return 0 (no agreement)

- CR_{t,s_i}: is the concession rate at time t for seller s_i. $CR_{t,s_i} \in [0, 1]$ and $Offer_{t,s_i} \in [IP_b, RP_b]$.

Our evaluation of the concession rate depends on the environment and self factor:

$$CR_{t,s_i} = \begin{cases} 0, & \text{if } t = T_s \ ; \\ \lambda, & \text{if } t = T_e - \alpha \text{ where } \alpha = \text{ time to complete one negotiation round;} \\ CR_{t',s_i}, & \text{if } CR_{t,s_i} - CR_{t',s_i} < 0; \\ CR_{t',s_j} * \mu, & \text{if offer of } CR_{t',s_j} \text{ reserved and } j \neq i; \\ w_{Env_t} E_t + w_{Self_t} S_t, & \text{otherwise} \end{cases} \tag{3.3}$$

where

- λ, μ: are real constants where $\lambda \in [0.9, 1)$ and $\mu \in [0.8, 1)$;
- E_t: is the effect of the environment factor where $E_t \in [0, 1]$;
- S_t: is the effect of self factor where $S_t \in [0, 1]$;
- w_{Env_t}, w_{Self_t}: are the environment and self weights, normalised in the following sense: $w_{Self_t}, w_{Env_t} \in [0, 1]$ and $w_{Self_t} + w_{Env_t} = 1$.

Before explaining the environment and self factors E_t and S_t, we demonstrate how we calculate each case of the concession rate CR_{t,s_i} in Equation 3.3 above.

Case 1: When b starts the negotiation, it will offer its initial price first.

Case 2: When time is α time units close to the deadline and the buyer b has no reserved offer, then it needs to secure an offer even if it is near to its reservation price. Therefore, b concedes near to its reservation price by setting the concession rate $CR_{t,s_i} = \lambda$. We set λ in Chapter 6 to $\lambda = 0.99$.

Case 3: When b generates an offer that is non-monotonic (i.e. $CR_{t,s_i} - CR_{t',s_i} < 0$) at t, then, to ensure its own rationality, b will offer its previous offer CR_{t',s_i}. This case needs a first generation of the value CR_{t,s_i} with cases 1 or 5.

Case 4: If b has a reserved offer at t', there is no need for b to concede more to other sellers. Thus, b will generate an offer which is less than 20% ($\mu = 0.8$) of its reserved offer. This case needs a first generation of the value CR_{t,s_i} with cases 1

or 5. We set μ in Chapter 6 to $\mu = 0.9$.

Case 5: Otherwise, b will generate an offer that takes into consideration the environment and self factors. To give a flavour of these, the following are some special cases illustrating how environment and self factors affect the concession rate.

a. Adding and/or removing a seller to/from the negotiation: while agent b is negotiating with agent $s1$ and agent $s2$, a new seller agent $s3$ enters the market and starts a new negotiation thread with b. This creates a new option for b. As a result, b will decrease its concession rate to see how the negotiation process proceeds with agent $s3$. On the other hand, when $s1$ exits the negotiation, it means b lost an option, thus, it has to increase its concession.

b. Adding and/or removing negotiation competitors: consider agent b negotiating with agents $s1$ and $s2$ as before, but now b perceives that a competitor $c1$ is negotiating with agent $s1$. Hence, b may concede by a larger amount to attract $s1$ to sell the resource. In contrast, if $c1$ leaves the market, b may concede by a smaller amount since it has no other competitors while negotiating with $s1$.

c. Change in the negotiation situation: if b is in a bad negotiation situation (a notion to be defined by a function when we explain the self factor S_t), then b has to concede more in order to secure the resource before the deadline. On the contrary, when in a good negotiation situation, b has a higher chance of winning the negotiation, which in turn leads to b reducing its concession rate.

Now, we will explain the environment and self factors E_t and S_t.

Self Factor

We compute the self factor using the following formula:

$$S_t = (\frac{1}{CO + 1} + NS + TE + EG_b)/4 \tag{3.4}$$

a) *CO:* is the number of reserved offers, obtained from $|CO|$ and updated each time the buyer reserves or cancels an offer. If the number of reserved offers increases, then the agent decreases the concession rate for generated offers. We normalise CO between $[0, 1]$ by dividing the number 1 by the number of reserved offers.

b) *NS:* is the negotiation situation for all threads. The evaluation of the negotiation situation in each thread is used for deciding the next offer for each thread. It is calculated using the following two criteria: $Criteria_1$ and $Criteria_2$.

- $Criteria_1$ is the opponent's *response time*. Given an offer from the buyer b to seller s_i at time t_l, denoted as $\delta^{t_l}_{b \to s_i}$, and a response to that offer from s_i to b at time t_m ($t_m > t_l$), denoted as $\delta^{t_m}_{s_i \to b}$, then $Criteria_1$ is as described below:

$$Criteria_1 = (t_m - t_l)/T_b \tag{3.5}$$

  $Criteria_1 \in (0, 1]$. We map the range of possible $Criteria_1$ values to one of three groups: incompatible, moderately compatible and compatible. In other words, we classify the opponent depending on how compatible/incompatible is the opponent's behaviour in relation to our expectations:

  - compatible $\in (0, 0.33]$;
  - moderately compatible $\in (0.33, 0.66]$;
  - incompatible $\in (0.66, 1]$.

- $Criteria_2$ is the opponent's *concession rate* at the current time $t = t_k$, which is based on a window of consecutive opponent offers $\phi > 1$, containing first the last offer made to b by s_i, denoted as $Offer^{t_{k-1}}_{s_i \to b}$, and ending with $Offer^{t_{k-\phi}}_{s_i \to b}$. We can then compute the opponent's concession rate as follows:

$$Criteria_2(\phi) = 1 - \left(\frac{Offer^{t_{k-\phi}}_{s_i \to b} - Offer^{t_{k-1}}_{s_i \to b}}{RP_b - IP_b} \right) \tag{3.6}$$

For example, if $\phi = 3$ and the previous consecutive opponent offers of s_i to b at time t_k are 770, 790 and 800, then the value of the concession will be $(800 - 770)/3$. This, however, is normalised so that $Criteria_2$ is a real number such that $Criteria_2 \in [0, 1]$. Similar to $Criteria_1$, the value of $Criteria_2$ is mapped to incompatible, moderately compatible and compatible.

From these two criteria, our goal is to derive a *local situation* δ_i for each negotiation thread i (where i represents the thread that the buyer b uses for representing the negotiation with seller s_i), which also takes the values incompatible, moderately compatible and compatible. Since the agent has a real time deadline (usually the duration of negotiation T_b in seconds) and limited computational resources, the speed of the negotiation is a key element to obtain an agreement. Thus, to minimise the computation time needed to calculate the criteria, we choose three discretization levels (compatible, moderately compatible and incompatible). Although this choice seems arbitrary, we propose this as an heuristic way to classify opponents according to our expectations. The intuition is that this approach is sufficiently discriminating if compared with a more coarse-grained classification of compatible/incompatible, while at the same time it is equally simple to explain with the added advantages of distinguishing behaviours that lie in the middle using 'moderately compatible'.

To compute this, we have chosen the multi-criteria decision-making process known as the Borda method [17]. The Borda method gives a value to every qualitative decision based on its position, as in Table 3.1. We select the Borda method for the following reasons: Borda (1) supports proportional representation of the situation of each thread; (2) is neutral and monotonic [16]; (3) produces a complete, transitive ranking for a set of alternatives [26]; (4) preserves the decisiveness of the labels: incompatible, moderately compatible and compatible [38]; (5) uses straightforward calculation; and (6) employs feasible computation.

Range	**Qualitative Decision**	**Score**
$(0, 0.33]$	compatible	1
$(0.33, 0.66]$	moderately compatible	2
$(0.66, 1]$	incompatible	3

Table 3.1: Mapping procedure using the Borda method.

Then, we calculate δ_i as the sum of the scores $Criteria_1$ and $Criteria_2$:

$$\delta_i = Criteria_1 + Criteria_2. \tag{3.7}$$

Given the score combinations of $Criteria_1$ and $Criteria_2$, the domain of δ_i is $[2, 6]$, with 2 implying that the opponent is compatible with our style of negotiation and 6 implying that the opponent is incompatible. We then derive the global negotiation situation δ for buyer b, which we calculate as the sum of all the local negotiation situations δ_i, as shown in Equation (3.8). However, δ needs to be normalised, so that we can use the normalised value to generate the concession rate. So NS is the normalised global negotiation situation given by Equation 3.9, where 2 and 6 are the minimum and maximum values that a thread can score respectively, while y is the number of sellers $y = |Seller_b^t|$.

$$\delta = \sum_{i=1}^{y} \delta_i \tag{3.8}$$

$$NS = \frac{\delta - (2 * y)}{(6 * y) - (2 * y)} \tag{3.9}$$

c) TE: is the effect of the passage of time on the concession rate for buyer b. We use Equation 3.10 to normalise TE between $[0, 1]$.

$$TE = \frac{t - T_s}{T_b} \tag{3.10}$$

d) EG_b*:* is the eagerness of buyer b to obtain the resource and is specified by the user. As discussed in Section 3.2, EG_b is between $[0, 1]$.

Since the value of $S_t \in [0, 1]$, and the values of each S_t's sub-factors (CO, NS, TE, EG_b) $\in [0, 1]$, then we need a weighting method to ensure that the value of the sub-factors summation will be $= 1$. Thus, because we have four sub-factors we choose to divide the summation of all sub-factors by four to obtain a value for $S_t \in [0, 1]$.

Environment Factor

We compute the environment factor using the following formula:

$$E_t = (\frac{1}{Se_t} + C_t + R_{ds})/3 \tag{3.11}$$

a) Se_t*:* is the number of sellers that are actively negotiating with b. Note that the value is $1/Se_t \in (0, 1]$ (see case 5(a)).

b) C_t*:* is the number of active competitor agents. The competitors are other agents who are trying to obtain an agreement from the sellers that are negotiating with b for the same resource. We assume that this number is obtained from the market. The lower the number of competitors during negotiations, the higher the possibility of b reaching an agreement by conceding less (see case 5(b)). We normalise the value of C_t between $[0, 1]$ by using the following function:

$$C_t = \frac{|No.Competitors|}{MaxNo.Competitors} \tag{3.12}$$

Where $|No.Competitors|$ is the total number of buyers in the market and $MaxNo.Competitors$ is the maximum number of buyers that will be in the e-market.

c) R_{ds}*:* Demand/supply ratio is the ratio of the number of buyers to the number of sellers. The higher the ratio, the higher the price for the resource and,

hence, the more difficult it is to reach an agreement. Since both numbers are known by b, the ratio can be calculated. There are many e-markets in the literature that provide demand/supply ratios [101], including Tete-a-Tete, Kasbah, AuctionBot and the Fisher market. We normalise the value of R_{ds} between $[0, 1]$ by using the following function:

$$R_{ds} = \frac{DemandSupplyRatio - MinDemandSupplyRatio}{MaxDemandSupplyRatio - MinDemandSupplyRatio} \tag{3.13}$$

Where $DemandSupplyRatio$ is the e-market demand supply ratio and Max $DemandSupplyRatio$ and $MinDemandSupplyRatio$ are the maximum and minimum demand supply ratios in the e-market.

Since the value of $E_t \in [0, 1]$, and the values of each E_t's sub-factors (Se_t, C_t, R_{ds}) $\in [0, 1]$, then we need a weighting method to ensure that the value of the sub-factors summation will be $= 1$. Thus, because we have three sub-factors we choose to divide the summation of all sub-factors by three to obtain a value for $E_t \in [0, 1]$.

Assigning Weights to Factors

We develop a method to assign weights to the environment and self factors presented earlier. First, in Equation 3.14, we assign a weight to the self factor S_t depending on the value of the factor itself. Then, in Equation 3.15, we calculate the impact of the price range DM. The functions below present this method. We classify the value of S_t as low, medium or high, where:

- low $\in (0, 0.33]$;
- medium $\in (0.33, 0.66]$;
- high $\in (0.66, 1]$.

The reason for the classification as low, medium or high is the same as that for choosing to classify as incompatible, moderately compatible or compatible in

Section 3.4.1. The buyer always looks for its self factor S_t first, to see how satisfied it is with the current progress of the negotiation. If the value of S_t is low, it means that the buyer is in a good negotiation position because if it focuses on S_t only, then b will concede only by a small amount. Because of this, b must put more weight on the self factor, as it is now more important because it reflects the fact that b wants to concede very little, which in turn means that the environment factor should be assigned a lower weight. As a result, a high weight will be assigned to w_{Self_t}. This is captured by Equation 3.14, where if the value of S_t is low, we multiply by 0.75 to make it more important; if the value of S_t is medium, by 0.5 to make it indifferent; and if the value of S_t is high, by 0.25 to make it less important. In this way, w_{Self_t} is adjusted according to the negotiation progress.

$$w_{Self_t} = \begin{cases} DM * 0.75, & \text{if } S_t = \text{low} \\ DM * 0.5, & \text{if } S_t = \text{medium} \\ DM * 0.25, & \text{if } S_t = \text{high} \end{cases} \tag{3.14}$$

We also need to cater for the situation where the initial price of the seller is far away from the reservation price of the buyer. In this case, even if S_t is low, b will need to concede more, in order to close the price gap in fewer negotiation cycles. We therefore introduce a price distance multiplier DM in Equation 3.14, which measures the negotiation gap between b and s_i and is defined by Equation 3.15.

$$DM = (IP_{s_i}/RP_b) * TE \tag{3.15}$$

The ratio estimated by the initial price of the seller IP_{s_i} and the reservation price of the buyer RP_b is multiplied by TE, which is the effect of time passage, to remove a small percentage from the distance. The domain of TE is within the interval $[0, 1]$. We are now in a position to evaluate the weight of the environment factor, as shown in Equation 3.16.

$$w_{Env_t} = 1 - w_{Self_t} \tag{3.16}$$

3.4.2 Cancellation Penalty

As discussed in Chapter 2, the cancellation penalty is important to ensure fairness between negotiation participants, to save money and resources and to allow the system to be stable in terms of avoiding unwanted canceled deals.

The penalty amount needs to be determined by the agent's user before the start of the negotiation in order to decide when to cancel an offer. $Penalty_b$ represents the total amount of penalties for all the reserved offers. Thus, if the agent, during negotiation, finds a better offer, then the agent will cancel the least preferred offer (i.e. one cancel only) and request-to-reserve to the better offer. At the end of the deadline, the buyer accepts its most preferred reserved offer and cancels the rest of the reserved offers. The buyer keeps an MCO (max number of reserved offers) in case the opponents cancel the reserved offer.

As mentioned in Section 2.6.3, we opted for the percentage of the reserved offer penalty. The penalty for CONAN $Penalty_b$ is calculated based on the difference between the amount of money the agent has to pay to cancel a negotiation and the amount of money the agent has already received from opponents that canceled their negotiation from the buyer, i.e.:

$$Penalty_{b,s_i} = Penalty_{s_i,b} = D * Price_{reserved} \tag{3.17}$$

$$Penalty_b = (Penalty_{b,s_i} * |CO|) - (Penalty_{s_i,b} * |DO_{Seller_b^t}|) \tag{3.18}$$

where $Penalty_{b,s_i}$ is the penalty paid to the canceled offer to/from seller s_i, D is a percentage of the canceled offer $\in [0, 1]$, CO is the number of reserved offers so far, $DO_{Seller_b^t}$ is the number of canceled offers that occur from sellers $Seller_b^t$ negotiation with b.

The reason for choosing a percentage of the reserved offer is for fair comparison with the benchmark, being the strategy used by Williams *et al.* [115].

3.4.3 Heuristics to Decide Actions

In order for the buyer b to decide which action to choose at each point in time, we developed a rule-based strategy. We start with the request-to-reserve and cancel and then discuss the accept conditions. In these rules, we will use three parameters determined by the user: (a) MCO - this is the maximum number of reserved offers; (b) D - this is the percentage of the deal for canceling negotiation threads (more details in Section 3.4.2); and (c) MAN - this is the maximum number of active threads (ensures that the agent has limited computational resources).

Conditions to Request-to-reserve

b request-to-reserve to s_i when:

- offer $\delta^t_{s_i \to b} + Penalty_b \leq \delta^{t+1}_{b \to s_i} <$ any reserved offers and b is not near the deadline.
- s_i accepts the last offer from b and b's last proposed offer + $Penalty_b \leq$ any reserved offers and b is not near the deadline.

Conditions to Cancel

b Cancels from:

- the least preferred offer when $|CO| \geq MCO$;
- all reserved offers immediately after b has accepted an offer.

Conditions to Accept

b accepts:

- $\delta^t_{s_i \to b}$ when offer $\delta^t_{s_i \to b} + Penalty_b \leq \delta^{t+1}_{b \to s_i} <$ any reserved offers and b is nearing the deadline.
- s_i's acceptance when s_i accepted the last offer from b and b's last proposed offer + $Penalty_b \leq$ any reserved offers and b is nearing the deadline.

- the highest utility reserved offer when b is nearing the deadline $t = T_e$.

Conditions to Exit

We study next the conditions that the buyer b needs to check before exiting from an individual thread or from the negotiation as a whole.

Conditions to exit from a thread

b exits from:

- any thread if the deadline has been reached, i.e. $t = T_e$;
- δ_i if seller s_i is incompatible (i.e. based on the evaluation NS mentioned above) and there is a new seller s_j and $|Seller_b^t|{=}MAN$ (which means that the current opponent is incompatible and the buyer has a new opponent to negotiate with, but due to the limited number of active negotiations, the agent has to terminate the negotiation with the incompatible opponent).
- δ_i if s_i has accepted an offer but a higher utility offer has been accepted at the same time by s_j (which means that the buyer receives two accept actions from two different sellers at the same time, leading the buyer to choose the highest utility offer from s_j and exit from s_i).

Conditions to exit from the whole negotiation

b exits from the whole of the negotiation

- if the deadline has been reached, i.e. $t = T_e$.
- if $\delta^t_{b \to s_i}$ = Accept.

3.5 Analysis of Properties

Following the presentation of the CONAN strategy, we now discuss some obvious properties of our strategy CONAN. Such properties are typically studied in game theoretic models of negotiation [31, 32] because of the assumptions these models

make about agents' full rationality and the availability of complete information about the environment and the opponents. The presentation of CONAN's properties here aims to complement the results from the experiments (Chapter 6) and to show that it is possible for a strategy to have properties, even if the strategy is heuristic.

We derive the strategy properties from the behaviour of the strategy in different market settings. For instance, if the market has many sellers and fewer competitors, then the buyer will concede less since it has more opportunities to maximize its utility. On the other hand, if the market has few sellers and many competitors, then the buyer will concede more since it has less opportunities to maximize its utility.

Property 1: CONAN *will concede less as the number of sellers increases.*

When the number of opponents Se_t increases, then the environment factor E_t will decrease, assuming that the number of competitors C_t and the demand/supply ratio R_{ds} are stable (i.e. not changing during the change of Se_t). After that, E_t will be multiplied by the environment factor's weight w_{Env_t} which will also decrease the value of E_t. Thus, assuming that the value of the self factor S_t is stable, the total amount of concession CR_{t,s_i} will decrease.

Property 2: CONAN *will concede more as the number of competitors increases.*

When the number of competitors C_t increases, then the environment factor E_t will increase, assuming that the number of sellers Se_t and the demand/supply ratio R_{ds} are stable. After that, E_t will be multiplied by the environment factor's weight w_{Env_t} which will take a percentage of the value of E_t. Thus, assuming that the value of the self factor S_t is stable, the total amount of concession CR_{t,s_i} will increase.

Property 3: CONAN *will make more reserved offers with opponents when the penalty is very low.*

CONAN's rules to decide whether to request-to-reserve (offer $\delta^t_{s_i \to b} + Penalty_b \leq \delta^{t+1}_{b \to s_i}$ < any reserved offers and b is not near the deadline) to an opponent check if the opponent's offer $\delta^t_{s_i \to b}$ covers the amount of the penalty $Penalty_b$ that has

to be paid for cancellation from another opponent. If the canceling penalty is low, and assuming $\delta^{t}_{s_i \to b}$ and $\delta^{t+1}_{b \to s_i}$ are stable, then the agent can request-to-reserve to another opponent and pay a penalty which will not have a notable effect on the negotiation budget. Thus the agent can make more reserved offers with opponents.

Property 4: CONAN *behaves rationally*

CONAN produces monotonic offers in the sense that the agent will offer its previous offer or increase its concession rate (rational behaviour) but it will never decrease its concession rate, which it would consider irrational.

When CONAN generates an offer that is non-monotonic (i.e. $CR_{t,s_i} - CR_{t',s_i} < 0$) at t, then, to ensure its own rationality, CONAN will offer its previous offer CR_{t',s_i}.

3.6 Summary

In this Chapter, we proposed our concurrent negotiation model (architecture, protocol and strategy), which satisfies Objectives 1-3 listed in Section 1.3. Firstly, we proposed a concurrent negotiation architecture (Objective 1), designed as a specialized extension of previous work with the KGP model [107] to satisfy a complete concurrent negotiation model. Secondly, we presented a concurrent negotiation protocol (Objective 2). The protocol is a revised version of a well-known alternating protocol that can support concurrent negotiations for open e-markets. Thirdly, we developed a novel strategy, CONAN, to reach one agreement to allocate resources by concurrently negotiating with multiple opponents in open, dynamic e-markets (Objectives 3). CONAN relaxes some of the strong assumptions made in the existing negotiation literature. The strategy employs a weighted combination of modelling the e-market environment and observing the progress of concurrent negotiations in which the agent is involved. An offer is generated using heuristics, allowing an agent to decide when to act and what action to consider. These actions are based on our version of the alternating offers protocol. Also, we discussed some of the properties of CONAN.

In the following chapter, we will specify CONAN by using a logic-based knowledge representation method.

Chapter 4

Strategy Implementation

In Chapter 3, we presented the theoretical design of the strategy CONAN, without discussing the implementation issues. This chapter addresses the modelling and implementation of the negotiation strategy. We will present how we implemented the negotiation strategy CONAN by studying the implementation language and knowledge representations that satisfy Objective 4.

In the remainder of this chapter, we will use the architecture presented in Section 3.1 to illustrate how we implement our negotiating agent's mind, which holds the strategy CONAN. We first describe the GOLEM agent platform which will enable our experiments (Section 4.1). Then, we will introduce the implementation language based on logic programs specified in Prolog and extended for temporal reasoning with the Event Calculus (EC) in Section 4.2. After that, we will explore the knowledge representation in CONAN using the EC, with emphasis on how to implement the offer generation and action selection parts (Sections 4.3, 4.4 and 4.5). Finally, the chapter is summarised in Section 4.6.

4.1 Agent Development in GOLEM

Implementing our strategy requires the embedding of the action selection it offers into deployable software agents, so that we can test the strategy and experiment

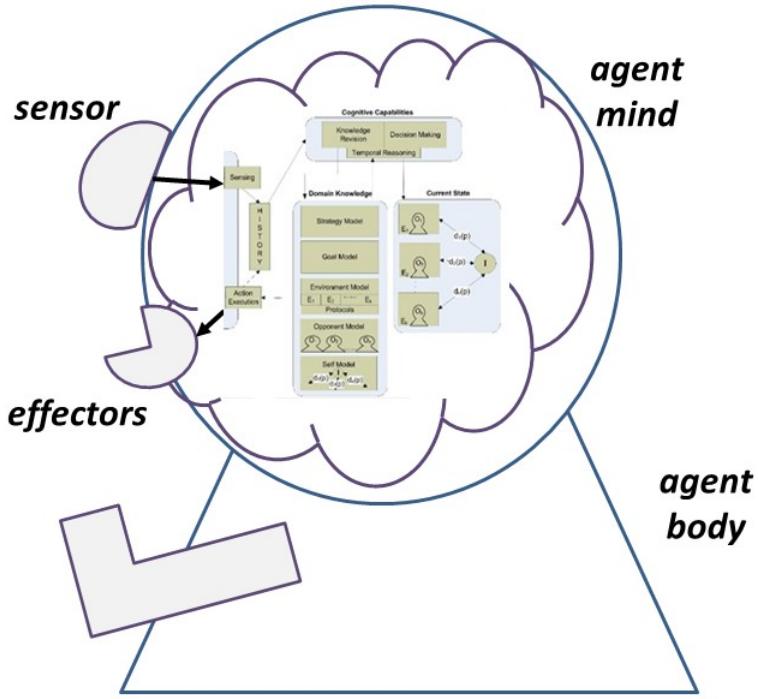

Figure 4.1: Negotiating Agent in GOLEM, where the agent mind is built with the architecture of CONAN.

with its properties. This in turn requires the selection of an agent platform that would allow us to deploy agents that communicate with each other and negotiate for the purchase of items. The platform we selected for the implementation of our strategy is GOLEM as discussed in Section 2.7.2. The reason for this is that GOLEM supports the development of agent models that require a logic-based implementation, like ours. The platform also supports agent interactions and communications, as well as distribution of the agent environment [19], features that are very important for negotiating agents in general and our experimental plans in particular.

A GOLEM agent consists of two core components, as shown in Figure 4.1. An

agent body situates the agent in the environment and aggregates the sensors and actuators that the agent needs to access and effect the environment. An agent mind then allows percepts to be extracted from the body and takes decisions about which actions need to be carried out using the body's actuators. The agent body is developed in Java, while the mind is implemented using logic-based techniques in Prolog. As in CONAN, the agent's mind is represented by our developed architecture in Section 3.1.

Listing 4.1 presents a simple cycle theory [51] implemented in Prolog as an illustration of agent control. When a new percept is received via the body, the mind revises the internal state of the agent; this revision in our work will be constructed as the assimilation of an observation in EC style, which will be discussed later. Then, the mind decides what action to perform based on its goal which is defined implicitly in the action selection rule. The agent's implicit goal is to maximize the agent's utility. For simplicity, the agent selects the first action whose conditions succeed in the state of the agent; the call to *once* ensures this (using the Prolog ! cut operator). The ordering of the `select` rules describe which action must be preferred over others. The mind executes the action internally to revise the agent's goals and returns it to the body. Again, action execution involves the addition of an event indicating that an action has been attempted. This cycle will be adapted later in Section 4.4 to represent CONAN.

```
1  % cycle step
2  cycle_step(Percept, Action, T):-
3        revise(perceived(Percept), T),
4        decide(Action, T),
5        execute(Action, T).

7  decide(Action, T):-
8        once(select(Action, T)).

10 % domain-dependent strategy
```

```
11 select(Action, T):-
12         Conditions [T].
13 ...
14 execute(Action, T):-
15       revise(attempted(Action), T).

17 revise(Event , T):-
18       assert(happens(Event, T)).
```

Listing 4.1: Simple agent mind in GOLEM.

GOLEM allows agent developers to make their own design choices about the implementation of the cycle step, e.g. develop more complex agent decision-making [36] by overwriting the definitions of `decide/2` and `execute/2`. In addition, the mind can be developed in another programming language (like Java), known as an imperative agent, for developers who do not require logic-based strategies.

We develop CONAN in GOLEM using a logic-based approach, thus giving our strategy a *declarative* flavour. Declarative strategies allow developers to specify strategies that can be transparent to a human user, in that the agent can explain why it has taken certain actions during a negotiation. The reason for choosing to implement CONAN declaratively is because declarative strategies: (1) can be expressive and transparent to model negotiation decisions; (2) provide abstract and executable representation; and (3) have been successful in modelling bilateral negotiations [89, 105].

In the next section, we will describe the use of Prolog and EC in building the CONAN declarative implementation.

4.2 Event Calculus (EC)

We represent the state of an agent during negotiation as a temporal logic program. We specify the rules of such a program in Prolog, which we assume the reader is familiar with. Prolog as a computational logic language has been successful,

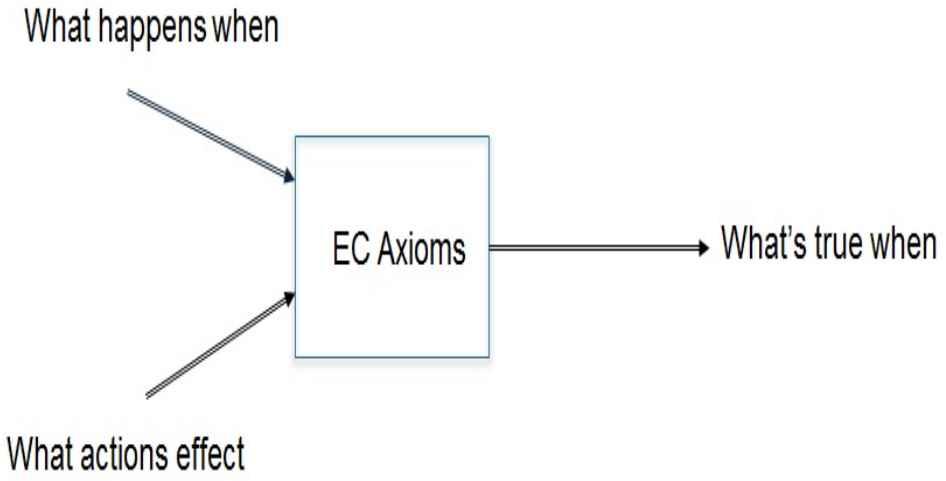

Figure 4.2: How the Event Calculus functions [98].

as described in the literature [34]. Informally, Prolog uses the convention that a constant is an identifier starting with a lower-case letter, while a variable is an identifier starting with an upper-case letter. A constant stands for a specific entity, and different constants stand for different entities. In contrast, a variable can stand for any entity, and different variables can stand for the same entity. The predicate name must be a constant, while each argument can either be a constant or a variable. As Prolog can be used to answer queries, or to achieve goals, the prompt "?-" denotes a query whose truth will be computed by the program.

As an agent negotiation strategy has to deal with time issues, such as deadlines, our strategy has a temporal part which we represent using the Event Calculus (EC) [53]. The EC is a logical language for reasoning about actions and their effects. Figure 4.2 illustrates the reasoning in the EC. As explained in [98], the logical mechanism of the calculus infers what is true when given what happens when and what actions do. More specifically, the "what happens when" part of Figure 4.2 is a narrative of events, and the "what actions effect" part describes the effects of actions.

The reasons behind choosing the EC for formalizing the CONAN strategy are:

- the EC formalization is based on a flexible, solid theoretical basis [98];
- the EC provides a practical means of implementing an executable system specification [8];
- the EC reduces ambiguity and enables rigorous reasoning [109];
- the EC has been applied successfully to the formalisation of market interactions in the financial sector [109];
- the EC has been successfully used to represent a contract net protocol (Section 2.4.1) [9], which is very similar to our concurrent negotiation protocol, argumentation based models [15] and multi-agent systems in general [53];
- the EC represents the negotiation environment as a structure evolving over time [19];
- the EC has been successfully applied to represent negotiation protocols in the literature [96];
- to the best of our knowledge, we believe there has been a lack of using EC to formalize the temporal needs of negotiation strategies.

4.2.1 EC Predicates

We will use a dialect of the EC based on temporal variables known as multi-valued fluents due to the fact that these variables can take different values at different times [9]. Table 4.1 summarizes the basic predicates in EC [8, 98].

4.2.2 The Axioms

In this thesis we will use the following domain-independent axioms for describing the Event Calculus in Prolog:

1. ```
   holds_at(F = V, Tn):-
           Tn >= 0,
   ```
   ```

Predicate	**Meaning**
initially(F =V)	at time 0, the value of fluent F is V.
initiates(Act, F =V, T)	at time T, the action Act occurs which initiates a period of time for which the value of fluent F is V.
terminates(Act, F =V, T)	at time T, the action Act occurs which terminates a period of time for which the value of fluent F is V.
happens(Act, T)	at time T, Action Act occurs.
holds_at(F =V, T)	at time T, the value of fluent F is V.

Table 4.1: Main Predicates of the Event Calculus.

```
initially(F = V),
\+ broken(F = V, 0, Tn).
```

This axiom states that the fluent `F` still holds at time `Tn` if a fluent `F` initially holds and is not terminated (broken) between time `0` and time `Tn`.

2.
```
holds_at(F = V, Tn):-
        happens(Act, Ti),
        Ti < Tn,
        initiates(Act, F = V, Ti),
        \+ broken(F, Ti, Tn).
```

This axiom states that the fluent `F` still holds at time `Tn` if the fluent `F` is initiated at some earlier time `Ti` (i.e. `Ti` is less than `Tn`) and `F` is not terminated (broken) between `Ti` and `Tn`.

3.
```
broken(F = V, Ti, Tn):-
        happens(Act, Tj),
        Ti < Tj,
        Tj < Tn,
        terminates(Act, F = V, Tj).
```

This axiom states that fluent `F` is broken between time `Ti` and time `Tn` if

an action `Act` occurs at time `Tj` and terminates the fluent `F` and time `Tj` is between time `Ti` and time `Tn`.

4.
```
terminates(Act, F = V, T):-
        initiates(Act, F = Vnew, T),
        holds_at(F = V, T),
        V \= Vnew.
```

   This axiom states that fluent `F = V` is terminated at time `T` if an action `Act` initiates a fluent `F = Vnew` and `V` is not equal to `Vnew`.

The first two axioms will be used to formalize CONAN in order to find out if a fluent holds at a certain time. The third and fourth axioms will be used to further define the first two axioms (`holds_at`). With the EC, through the use of the predicates `initiates/3`, `terminates/3` and `happens/2`, we imply that all the negotiation actions and the effects of these actions will be known to the agent. This allows the agent to spend more time deciding its actions using its knowledge of the effect of these actions.

4.3 Knowledge Representation in CONAN

In this section, we will show how to use the EC to describe the internal state of the agent in order to participate in negotiation with other agents using our version of the concurrent alternating offers protocol (Section 3.3). We will describe how fluents are represented, how events resulting from percepts or attempts of an action will be specified and how the effect of these actions will be carried out.

We chose the EC based on the reasons outlined in Section 4.2. In CONAN, the fluents of the EC are used to describe the state of the negotiation with each seller and the concurrent negotiations in general. `happens/2` will be used to describe the actions of CONAN and the sellers (i.e. offer, request-to-reserve, cancel, accept and exit). The effects of a negotiation action on the negation fluents are described by the `initiates/3` and `terminates/3` predicates.

4.3.1 Fluents in CONAN

During a negotiation a software agent needs a number of fluents to represent what changes during the negotiation. This information normally will contain details about the item/resource to be negotiated, the price negotiated in different threads, and thread specific information. For example, we write:

```
turn_of(thread1) = seller
```

to say that it is the turn of the seller in the negotiation thread `thread1` and not the buyer. The following is a list of some of the fluents that will be used in CONAN.

F1. `number_competitors = CNumber`.

This fluent states the number of competitors `CNumber` in the market. `CNumber` $\in Integer$.

F2. `result(ThreadId) = TStatus`.

This fluent reports the result of the negotiation in a thread id `ThreadId` in terms of whether it has failed, is under negotiation, successful, reserved (hold) or the seller has accepted CONAN's offers. `TStatus` $\in$ {*failed*, *onnegotiation*, *success*, *hold*, *seller_accept*}.

F3. `total_status = Status`.

This fluent reports the status of the overall negotiation in terms of whether it has ended or not. `Status` $\in$ {*finished*, *negotiating*}.

4.3.2 Actions

We have already seen in Section 3.3 that our version of the alternating offers protocol requires the agent to be capable of interacting using a number of actions. There are two types of actions: (a) *market protocol actions* due to the interaction of the agent with agents that control the e-market (i.e. infrastructure agents in

Section 5.2.2); and (b) *negotiation protocol actions* due to the interaction of the agent with other negotiating agents. We represent these actions as follows:

Market Protocol Actions

MC1. `notify_about_new_seller(ThreadId)`.

This action is an announcement from the market (Section 5.2.2) to notify the agent about the arrival of a new seller with thread id `ThreadId` that is selling the target resource that the agent wants to buy.

MC2. `notify_about_change_in_number_of_competitors(NoCompetitors)`.

This action is an announcement from the market (Section 5.2.2) to notify the agent about the total number of competitors `NoCompetitors` (i.e. other buyers aiming to buy the same target resource that the agent wants to buy) currently in the market.

MC3. `notify_about_change_in_demand_supply_ratio(DemandSupplyRatio)`.

This action is an announcement from the market (Section 5.2.2) to notify the agent about a change in the demand/supply ratio in the market. The new ratio is `DemandSupplyRatio`.

Negotiation Protocol Actions

C1. `offer(ThreadId, Item, Price)`.

This action is an offer from the opponent with thread id `ThreadId` to sell/buy the resource `Item` for a price `Price`. This action can be used by both the buyer and the seller. For instance, if CONAN (the buyer) wants to offer to the seller `s1` with thread id `5` to buy an item `laptop12` for a price `500`, then the action will be `offer(5, laptop12, 500)`.

C2. `request_to_reserve(ThreadId, Item)`.

This action is to request-to-reserve (i.e. hold) an offer from the opponent with thread id `ThreadId` to buy the resource `Item`. As we discussed in our protocol Section 3.3, this action can be done only from the buyer side.

C3. `cancel(ThreadId, Item).`

This action is to cancel (i.e. unhold) an offer from the opponent with thread id `ThreadId` to sell/buy the resource `Item`. This action can be used by both the buyer and the seller.

C4. `accept(ThreadId, Item).`

This action is to accept an offer from the opponent with thread id `ThreadId` to sell/buy the resource `Item`. This action can be used by both the buyer and the seller.

C5. `exit(ThreadId, Item).`

This action is to exit from negotiating with the opponent with thread id `ThreadId` to sell/buy the resource `Item`. This action can be used by both the buyer and the seller.

When such an action is perceived by a sensor within the agent it is implemented by the `revise/2` predicate in the cycle step (see Listing 4.1) as the happening of an event. For example, if an offer of `500` from a seller with thread id `5` about `laptop12` is perceived at time `30` it is represented as:

```
happens(perceived(offer(5, laptop12, 500)), 30).
```

Similarly, if CONAN has selected an action to a seller with thread id `5` to request-to-reserve an offer, we write:

```
happens(attempted(request_to_reserve(5, laptop12)), 30).
```

4.3.3 Initial State of the Fluents

We initialize the fluents for a negotiation by using the `initially/1` declaration. For example, in CONAN we write:

S1. `initially(turn_of(ThreadId) = buyer).`

Initially the agent who makes the first move in a negotiation is the buyer.

S2. `initially(number_competitors = 0).`

Initially the number of competitors in the market is 0.

S3. `initially(total_status = negotiating).`

Initially the status of the overall negotiation is *negotiating*.

We also need to initialize the simulation parameters. In order for CONAN to start the negotiation, we need to set : (a) the name of the resource the user wants to buy; (b) the resource's initial price; (c) the resource's reservation price; and (d) the duration of the negotiation. So, (a)-(d) are set by the simulation platform (Chapter 5). The list of the simulation parameters is as follows:

SP1. `initially(item = Item).`

Initially the resource under negotiation is `Item`.

SP2. `initially(ip = Min).`

Initially the resource's initial price is `Min`.

SP3. `initially(rp = Max).`

Initially the resource's reservation price is `Max`.

SP4. `initially(deadline = Deadline).`

Initially the negotiation duration is `Deadline`.

The above fluents change as a result of observations being perceived and attempted actions being executed by the agent. We are now in a position to discuss how the initial state of the agent evolves as a result of the agent engaging in negotiations.

4.3.4 Evolution of the Agent's State

When an action is performed by an agent, the action results in an event happening in the state of the agent. Events change the values of fluents that describe the

agent's state as described by `initiates/3` and `terminates/3` rules. In our model, we distinguish between fluents that describe the overall negotiation and fluents that describe what holds in specific negotiation threads with individual sellers. Next we will classify the effect of the negotiation actions on (1) the overall negotiation; and on (2) a specific negotiation thread.

E1. Effect on the overall negotiation:

EW1.
```
initiates(attempted(exit_all(Item)),
          total_status = finished,
          T).
```

The `exit_all(Item)` action at time `T` initiates that the overall negotiation is finished by setting the value of the fluent to `finished`.

E2. Effect of actions `offer` and `exit` on specific thread:

EE1.
```
initiates(perceived(offer(ThreadId, Item, New_price)),
          turn_of(ThreadId) = buyer,
          T).
```

The `offer` action initiates the buyer's turn at time `T` when an opponent with a thread id `ThreadId` offers a `New_price` for the resource `Item`.

EE2.
```
initiates(perceived(exit(ThreadId, Item)),
          result(ThreadId) = failed,
          T).
```

The action `exit` indicates a negotiation has failed on the thread `ThreadId` at time `T`, which implies that the negotiation ends, from the buyer/seller to/from the opponent for buying/selling the resource `Item`.

4.4 The Representation of the CONAN Strategy

CONAN, introduced in Chapter 3, has two types of heuristics: heuristics to decide actions and heuristics to generate offers. In the next sections, we will explain how

to present a rule-based knowledge representation of these heuristics.

4.4.1 Deciding Actions

To deal with the fact that we need to model decisions in many threads at the same time, we propose a different decision-making process to the one in Listing 4.1, as follows:

```
decide(Actions, T):-
      findall(Action, select(Action, T), Actions).
```

`findall/3` is a Prolog primitive that finds every action on each negotiation thread at time `T` because of the fact that the agent is participating in many threads. The thread id will be embedded with `Action` and the list of actions to be executed will be collected in the variable `Actions`.

Recall that, from Listing 4.1, action selection rules take the form:

```
select(Action, T):- Conditions [T].
```

Such rules state that the `Action` is selected if the `Conditions` hold at time `T`. The following is an example of when to decide to exit from a negotiation thread:

```
1 select(exit(ThreadId, Item), T):-
2    holds_at(self_deadline = Td, T),
3    T >= Td,
4    holds_at(result(ThreadId) = onnegotiation, T),
5    holds_at(item = Item, T).
```

Listing 4.2: Exit Action.

The agent selects the action `exit/2` for the negotiation thread `ThreadId`, if the current time `T` is greater than or equal to the agent's deadline `Td`, which indicates that the deadline has been reached. Appendix B lists more rules for the selection of any action.

4.4.2 Offer Generation

As in the previous section (Section 4.4.1), we explain next how the agent decides what actions to perform. Here, the agent needs to calculate the next offer it plans to propose, because the agent uses the intended offer to evaluate the opponents' actions and decide on a new action. Our approach requires different conclusions to be represented using `holds_at/2` to formulate the mathematical presentation of the offer generation in Section 3.4.1, as follows:

```
1 intended_offer(ThreadId, Offer, T):-
2    concession(CA, ThreadId, T),
3    holds_at(ip = Min, T),
4    holds_at(rp = Max, T),
5    Offer is Min + (Max - Min) * CA.
```

Listing 4.3: Offer Generation.

The predicate `intended_offer` returns an `Offer` for thread id `ThreadId` at time `T`. The offer is calculated based on the initial price `Min`, the reservation price `Max` and the concession rate `CA`. The `Min` and `Max` values retrieved using the predicate `holds_at` and the `CA` is calculated using the predicate specified below:

```
1  concession(CA, ThreadId, T):-
2     findall(Cycle_Times, neardeadline(Cycle_Times), List),
3     list_max(List, Near_Deadline),
4     holds_at(self_deadline = Td, T),
5     (
6     T >= (Td - (Near_Deadline)) ->
7     CA is 0.99
8     ;
9     selfFactors(ThreadId, Self, T),
10    w_Self(ThreadId, W_Self, Self, T),
11    envFactors(ThreadId, Env, T),
12    w_Env(ThreadId, W_Env, W_Self, T),
13    CA is (W_Env * Env + W_Self * Self)
```

```
14 |     ).
```

Listing 4.4: Concession Rate Calculation.

The predicate `concession(CA, ThreadId, T)` returns the concession rate `CA` for each `ThreadId` at `T`. The value of `CA` will be 0.99 if the agent is near its deadline. `Near_Deadline` is the maximum time the agent takes to complete one cycle (i.e. round) of negotiation. Otherwise, `CA` will be a summation of weighted combinations of the environment factor `Env` and the self factor `Self` for each thread. The predicates `w_Env` and `w_Self` retrieve the weights `W_Env` and `W_Self`, respectively.

To determine the self factor we define the rule:

```
1 | selfFactors(ThreadId, Self,  T):-
2 |    holds_at(self_deadline = Td, T),
3 |    holds_at(startTime = Ts, T),
4 |    DeadlineF is float(T-Ts)/float(Td-Ts),
5 |    reserved_offer(N, T),
6 |    ReservedF is (float(1)/float(N+1)),
7 |    holds_at(eagerness = EagernessF, T),
8 |    generalNegotiationStatus(NegotiationStatusF, T),
9 |    Self is 0.25*(DeadlineF + ReservedF + EagernessF +
  |       NegotiationStatusF).
```

Listing 4.5: Self Factor Calculation.

Self factor `Self` is calculated based on the sub-factors: deadline, reserved offers, eagerness and negotiation status. In line 2-3, we access the deadline `Td` and the starting time `Ts` in order to evaluate the sub-factor `DeadlineF` in line 4. In line 5-6, the sub-factor `ReservedF` is calculated based on the number of reserved offers. `EagernessF` is returned in line 7 which was initialized by the user. In line 8, the final sub factor `NegotiationStatusF` is returned.

After obtaining all the sub-factors, in line 9, `Self` is computed based on the summation of the 0.25 of each sub factor. Using 0.25 is the same as dividing the summation of the sub-factors by 4 ($1/4 = 0.25$) as explained in Section 3.4.1.

To compute the environment factor we use the rule:

```
1  envFactors(ThreadId, Env, T):-
2     number_sellers(NoSellers, T),
3     NoSellersF is float(1)/float(NoSellers),
4     holds_at(number_competitors = NoCompetitors, T),
5     holds_at(max_number_competitors = MaxNoCompetitors, T),
6     NoCompetitorsF is float(NoCompetitors)/float(MaxNoCompetitors),
7     holds_at(demandSupplyRatio = DemandSupplyRatio, T),
8     holds_at(min_demand_supply_ratio = MinDemandSupplyRatio, T),
9     holds_at(max_demand_supply_ratio = MaxDemandSupplyRatio, T),
10    DemandSupplyRatioF is float(DemandSupplyRatio -
          MinDemandSupplyRatio)/float(MaxDemandSupplyRatio -
          MinDemandSupplyRatio),
11    Env is 0.33 *(NoSellersF + NoCompetitorsF + DemandSupplyRatioF).
```

Listing 4.6: Environment Factor Calculation.

By the same token, the environment factor `Env` is calculated based on three sub-factors: number of sellers, number of competitors and demand/supply ratio. In line 2-3, `NoSellersF` is normalized based on the actual number of sellers that negotiate with the agent. The predicates in line 4-6 retrieve and normalize the sub-factor `NoCompetitorsF`. Similarly, in line 7-10, the predicates retrieve and normalize the sub-factor `DemandSupplyRatioF`.

After obtaining all the sub-factors, the environment factor `Env`, in line 11, is computed based on the summation of 0.33 of each sub-factor. The reason for using 0.33 is the same as dividing the summation of the sub-factors by 3 ($1/3 = 0.33$) as explained in Section 3.4.1.

4.4.3 Action Execution

In the execute step, the agent will compile and update its knowledge based on all the actions selected to be sent to the opponents. The negotiation platform Recon, which is explained in Chapter 5, is responsible for sending actions between agents after being executed. As the `decide/2` predicate returns a list of actions, we need to overwrite the definition of `revise/2` to deal with lists of actions rather than a

single one as specified in Listing 4.1.

```
revise(attempted([]), T).
revise(attempted([Action|Actions]), T):-
      assert(happens(attempted(Action), T)),
      revise(attempted(Actions), T).
```

In this way the agent is cognizant of the actions it has carried out after the execution step. Note that there is an extended version of the revision process, where instead of asserting an event in the EC fashion we compile it. However, the compiler is part of the implementation of the GOLEM platform and its technical details are beyond the scope of this thesis.

4.5 Example Run

Time	Observation	Deliberation	Decision
t1	$s1$ & $s2$	`decide(Actions, t1)`	[`offer(5, laptop12, 700)`, `offer(6, laptop12, 700)`]
t2	`offer(5, laptop12, 950)`		
t3	`offer(6, laptop12, 920)`		
t4		`decide(Actions, t4)`	[`offer(5, laptop12, 730)`, `offer(6, laptop12, 720)`]

Table 4.2: Observations, deliberations and decisions of b during a negotiation.

Table 4.2 illustrates how the b agent interacts with sellers in the market in order to purchase a laptop. The scenario is presented from a buyer's perspective and assumes the user has specified that (a) the laptop's price will be in the range [700-900] and (b) the maximum negotiation duration will be 5 minutes. At time $t1$, b is negotiating concurrently with $s1$ and $s2$. At time $t2$, $s1$ sends a counter-offer to b while at time $t3$, $s2$ sends back an offer to b. In $t4$, b will follow the skeleton strategy in Section 4.4.1 to select the best action.

Listing 4.7 describes the action selection rule that is triggered in `decide(Actions, t1)`. When the negotiation status is `onnegotiation`, not near the deadline `T <` `Td`, the seller has not yet offered `Opponent_offer == 0` and it is the buyer's turn `turn_of(ThreadId) = buyer` then the buyer will offer its initial price `ip = Offer`.

```
1 select(offer(ThreadId, Item, Offer), T):-
2         holds_at(result(ThreadId) = onnegotiation, T),
3         holds_at(self_deadline = Td, T),
4         T < Td,
5         holds_at(neg_price(ThreadId, Item) = Opponent_offer,
              T),
6         Opponent_offer == 0,
7         holds_at(turn_of(ThreadId) = buyer, T),
8         holds_at(item = Item, T),
9         holds_at(ip = Offer, T).
```

Listing 4.7: Action selection rule triggered in `decide(Actions, t1)`.

Listing 4.8 describes the action selection rule triggered in `decide(Actions, t4)`. When the negotiation status is `onnegotiation`, not near the deadline `T <` `Td`, the seller has already made offers `Opponent_offer =\= 0`, it is the buyer's turn `turn_of(ThreadId) = buyer` and there are no reserved offers then the buyer will offer a price based on the predicate `next_offer(T, Offer, CounterIntendedOffers)`.

```
1 select(offer(ThreadId, Item, Offer), T):-
2         holds_at(result(ThreadId) = onnegotiation, T),
3         holds_at(self_deadline = Td, T),
4         T < Td,
5         holds_at(neg_price(ThreadId, Item) =  Opponent_offer,
              T),
6         Opponent_offer =\= 0,
7         holds_at(turn_of(ThreadId) = buyer, T),
8         holds_at(item = Item, T),
9         next_offer(T, Offer, CounterIntendedOffers),
10        reserved_offer(NoReservedOffers, T),
```

```
11          NoReservedOffers == 0,
12          Offer < Opponent_offer.
```

Listing 4.8: Action selection rule triggered in `decide(Actions, t4)`.

4.6 Summary

In this chapter, we started by introducing how we implemented CONAN by studying its implementation language, which is a logic program implemented with Prolog and extended with the EC. We described the agent GOLEM's architecture which includes the mind and body of the agent, and how we adopted our agent's architecture presented in Chapter 3 as the agent's mind. Then we explained in depth the EC predicates and axioms which handle the actions and the effects of the actions that occurs in the environment. We also showed the knowledge representation of CONAN, which includes how fluents, actions and the effects of the negotiation actions are represented. After establishing the basics of the EC, we explored in detail how to define and formalize each step in CONAN.

In the following chapter, we will present the design, implementation and evaluation of the simulation platform RECON, which will be used to evaluate the performance of CONAN in an open and dynamic environment.

Chapter 5

Experimentation Simulator: RECON

In the previous chapter, we presented the logic-based implementation of the strategy CONAN. However, CONAN needs to be evaluated in an open and dynamic environment in order to test whether the agent maximizes its utility and under which circumstances. To address the evaluation requirements of CONAN, in this chapter we introduce our negotiation simulator RECON: a Robust multi-agent Environment for simulating COncurrent Negotiations, which satisfies Objective 5. We propose the design, implementation and evaluation of RECON.

In the remainder of this chapter, we start by providing the background of negotiation platforms in Section 5.1. In Section 5.2, we explain the RECON components in detail. Then, in Section 5.3 we empirically evaluate the performance of RECON with increasing numbers of agents in a market, and present our experimental results. Sample screens from the RECON graphical user interface are displayed in Section 5.4. Finally, the chapter is summarised in Section 5.5.

5.1 Background

One major issue in the design of a negotiation agent that participates in e-markets is how to evaluate the performance of its strategy. There is a huge interest in developing simulations to model and evaluate the performance of agents in an open and dynamic environment [77]. In negotiation, some researchers test their agents theoretically by setting theorems and proofs of these theorems [32, 33], but most agent developers use simulation platforms [29, 46, 59, 80, 115] especially for evaluating heuristic-based strategies. This gives rise to the need for a standardised simulation environment to provide fair and objective comparisons between negotiating agents. A successful negotiation platform should be (i) able to provide an open and dynamic environment for its participants to negotiate concurrently; (ii) robust to the changes that occur in the market; (iii) reliable for communication among agents; and (iv) scalable in terms of the number of agents it can support.

Recent negotiation literature suggests that most simulators are designed for specific domains with limited protocol and agent types [29, 46, 59, 80, 110, 115]. GENIUS [61] is a state-of-the-art negotiation platform that provides a competitive framework to test and compare bilateral negotiation strategies. Developers can implement their agent strategies using Java, following the provided negotiation protocol. However, the periodic extension of the GENIUS platform (see Williams *et al.* [115]) is not provided as open-source software. The system does not provide an open and dynamic e-market setting where buyers and sellers can enter/leave the system. The e-market options in GENIUS are very limited in their ability to describe a realistic setting. For example the system imposes fixed deadlines for the end of negotiation, which are publicly available to both buyers and sellers. In addition, although GENIUS has been thoroughly tested using the ANAC competition series [37], it is unclear whether it can support markets with a large number of negotiating agents.

Motivated by the current limitations in GENIUS we developed Recon: a Robust multi-agent Environment for simulating COncurrent Negotiations. Recon is

build on top of GOLEM (Section 2.7.2) with the use of the GOLEMLite library (Section 2.7.3) and supports the development of software agents (both buyers and sellers) negotiating concurrently with other agents over various issues. In contrast to most agent development platforms such as GENIUS, which only support imperative agents, RECON supports also declarative agents. Declarative agents allow developers to specify strategies that can be transparent to a human user, in that the agent can explain in the future why it has taken certain actions during a negotiation.

Functionality	RECON	GENIUS
Open & dynamic	✓	✗
Support concurrent protocols	✓	✗
Large number of agents	✓	✗
Support logic programming	✓	✗
Private deadlines	✓	✗
Continuous-valued issues	✓	✗
Human negotiators	✗	✓
Multiple issues	✗	✓
GUI	✓	✓

Table 5.1: Comparison of functionalities in RECON and GENIUS.

5.2 RECON Development

RECON supports the development of negotiating agents for testing both buyer and seller strategies. These strategies can be developed either declaratively (implemented in Prolog) or imperatively (implemented in Java). Also, the system supports concurrent negotiations and dynamic markets, where agents can enter/leave

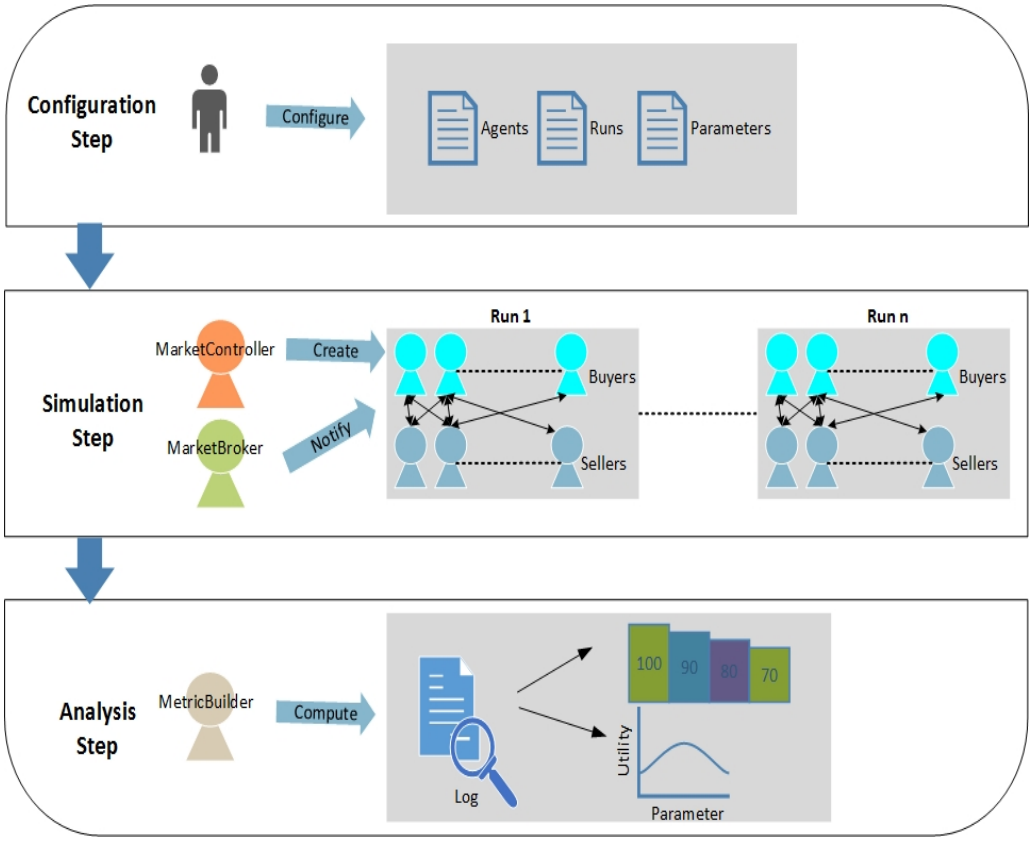

Figure 5.1: The RECON architecture.

the e-market during negotiation. In addition, agents can have a private deadline during a negotiation, instead of a common public one, to reflect what happens in practice. Moreover, realistic applications require negotiations with both continuous and discrete issues, rather than supporting discrete and integer-valued issues only. Table 5.1 compares the functionalities of RECON and GENIUS.

A simulation in RECON assumes three steps (see Figure 5.1):

1. In the configuration step, the user describes the simulation setting using the configuration tool.

2. In the simulation step, buyers and sellers negotiate with each other. The *MarketController* agent controls the execution while the *MarketBroker* agent acts as a yellow pages service for the buyers.

3. In the analysis step, simulation logs are inspected by the *MetricBuilder* agent to calculate related evaluation metrics.

Next, we describe each step in detail.

5.2.1 Configuration Step

In this step, the user defines the simulation parameters, the number of repeated runs in each parameter combination (RC), and the market agents. The simulation parameters, which describe the e-market setting, are summarised in Table 5.2.

Variables	**Description**
Buyer's Initial Price	Starting price IP_b of buyer b.
Buyer's Reservation Price	Maximum price RP_b buyer b will offer.
Seller's Initial price	Starting price IP_{s_i} of seller s_i.
Seller's Reservation Price	Maximum price RP_{s_i} seller s_i will offer.
Market update time	The time that elapses for the market to change (its density and demand/supply ratio due to new agents entering or existing agents leaving the market).
Change in market density	The rate at which the number of agents increases or decreases.
Change in demand/supply	The rate at which the ratio of buyers to sellers increases or decreases.
Deadline	Duration of the negotiation T_b for buyer b.

Table 5.2: Simulation parameters.

Each parameter has a set of default qualitative values drawn from the *ValueBuckets* such as small, medium, or large. These correspond to a range of quantitative values. The number of parameters and their corresponding value buckets are adjustable depending on the user's preferences. Let us now review each parameter:

- *Selected Buyers*: contains a list of buyer strategies where the MarketController adds them to the e-market to measure their performances.

- *Buyer/seller pool*: contains a pool of strategies where the MarketController selects randomly to add new buyers and sellers.

- *Change in market density (MD)*: determines the number of market agents in the e-market at any given time point. Note that this does not impose any constraints on the composition of agent types.

- *Change in demand/supply (MR)*: determines the ratio of buyer agents to seller agents in the e-market (i.e. demand(X)/supply(Y) ratio). The values of market ratio and market density are used jointly to determine the number of the buyers and sellers in the e-market based on the following formula:

$$NoBuyers = (MD * X)/(X + Y) \tag{5.1}$$

$$NoSellers = (MD * Y)/(X + Y) \tag{5.2}$$

  For instance, if the value of Market Ratio is 2:1 and the market density is 30, then the number of buyers is $(30 * 2)/3 = 20$, and the number of sellers is $(30 * 1)/3 = 10$.

- *Market update time (MC)*: represents the number of seconds at which the e-market has to change. The change involves adding/removing buyers and sellers according to MD and MR. This parameter ensures an open environment, where participants are not necessarily known at design time.

- *Deadline (MDL)*: represents the deadline for all market agents. All buyers have the same deadline, which is different from the seller's deadline. Moreover, the deadlines are kept private to the agents, so that buyers and sellers do not know each other's deadlines. The buyer's negotiation timer starts when it enters the e-market. On the other hand, the seller has many negotiation timers: one per negotiation and another which starts when it receives the first offer from a buyer. Both buyers and sellers leave the e-market when they reach their deadlines.

- *Initial price (IP_b)(IP_{s_i})*: represents the ideal price for an agent to buy/sell an item. All buyers/sellers in each run will have the same initial price IP_b/IP_{s_i} to provide fairness when comparing performance.

- *Reservation price (RP_b)(RP_{s_i})*: represents the maximum/(minimum) price the agent is willing to pay for (get from) an item. Similarly, all buyers/sellers have the same reservation price RP_b/RP_{s_i} for fairness.

5.2.2 Simulation Step

The actual negotiation between the market agents takes place in this step. In general, the number of negotiation simulations RECON will conduct in each experiment depends on the parameters and RC from the configuration step. The following equation calculates the number of simulations:

$$NoSimulations = (\prod_{i=1}^{n} NoValueBuckets_i) * RC \tag{5.3}$$

where n is the number of parameters. For instance, to conduct a simulation for four parameters MD, MR, MC and MDL where each parameter has three values, and $RC = 100$, the simulator will conduct 8100 (3 * 3 * 3 * 3 * 100) simulations. Thus, this step will terminate when all the 8100 simulations finish. Likewise, each simulation ends when all of the selected buyers (i.e. the buyers that the user wants to measure the performances of) leave the e-market, either because they have reached their goal or because they have reached the deadline. This step is managed by the infrastructure agents *MarketController* and *MarketBroker*, and played by *MarketAgents*. Infrastructure agents were explained in Section 2.7.3.

Market Controller

The *MarketController* is an infrastructure agent that is responsible for managing all the simulations that will be run as part of the overall experiment. The MarketController runs the simulator for a fixed number of times defined by the user. At

the end of each simulation, it logs the negotiation messages between MarketAgents. The MarketController has the following responsibilities:

- it creates the MarketBroker;
- it uses the configuration to initialise each simulation and creates all the required MarketAgents;
- during each simulation, it controls the number of MarketAgents in the e-market by adding and removing them according to the market density and demand supply ratio;
- it asks all MarketAgents about their decisions at fixed time intervals;
- at the end of each simulation (i.e. when all Selected Buyers reach their goal or deadline), it stops and removes all agents, saving the negotiation history to a log file;
- when all simulations finish, the MarketController stops and removes the MarketAgents and the MarketBroker, and finally terminates itself.

Market Broker

The *MarketBroker* is an infrastructure agent created by the MarketController to match buyers with sellers. It acts as a 'yellow pages' service for the buyers where it notifies the existing buyers when a new seller enters the e-market, and maintains a register of all MarketAgents in the e-market. The MarketBroker maintains two registers internally: a list of buyers and a list of sellers. When a new seller enters the e-market, the MarketBroker will send that seller's thread id to every buyer that has registered interest in purchasing the item that the new seller is willing to sell.

Market Agents

The MarketAgents are agents constructed by the MarketController. Their goal is to allocate/offer a resource (i.e. product or service) from other MarketAgents by

negotiating the resource issues (i.e. price). MarketAgents announce themselves when they enter the e-market, specifying their class (buyer or seller) as well as the item they are demanding/offering. We followed the architecture of MarketAgents proposed in Section 3.1, as market agents are negotiating agents.

Buyers are MarketAgents with the goal of maximising their utility from purchasing an item. They are liable for initiating negotiation with sellers notified by the MarketBroker. A typical buyer will remain in the e-market until it reaches a successful negotiation, is told to leave by the MarketController, or reaches its deadline. Listings A.1 and A.2 in the Appendix A describe part of the reasoning process of the buyer agent both in Java and Prolog.

Sellers are MarketAgents with the goal of maximising their utility from selling an item. Each seller negotiates with buyers that send it offers. A seller will leave the e-market when the MarketController requests it or if its strategy dictates it must leave (e.g. because its inventory is empty). Sellers are responsible for terminating negotiations when their deadline is reached. We can develop strategies for a seller as we have with buyers.

5.2.3 Analysis Step

This step is controlled by the *MetricBuilder* agent, which is responsible for analysing the log files generated from the simulation step[1]. The purpose of this step is to observe how market agents reason during negotiation, and to determine why an agent is winning/losing. The MetricBuilder calculates a number of performance metrics defined by the user. The metrics are then plotted against various configuration parameters that describe the e-market setting. To reduce complexity, only the logs of selected buyer agents will be parsed to provide results.

Currently, the following metrics are supported by RECON:

- *Number of successful runs (SR_j)*: is the number of simulations per parameter combination j where a negotiation ends in an *accept* action from the buyer.

[1]Listing A.3 in the Appendix A illustrates an example of a log file

- *Average utility per parameter combination (AU_j)*: is the sum of the utility from each simulation divided by the number of runs (RC) in each parameter combination j. $AU_j \in [0, 1]$, where 0 indicates an unsuccessful negotiation and 1 denotes that the buyer was able to acquire the item at the initial price.

$$U_i^j = (RP_i^j - AP_i^j)/(RP_i^j - IP_i^j) \tag{5.4}$$

$$AU_j = \sum_{i=1}^{RC} U_i^j / RC \tag{5.5}$$

where U_i^j is the buyer's utility for the jth parameter combination and ith repeated run, RP_i^j is the reservation price, IP_i^j is the initial price, AP_i^j is the accepted price.

- *Average utility over successful runs (AUS_j)*: is the sum of the utility from each simulation, divided by the number of successful runs (i.e. when the buyer reaches an agreement). $AUS_j \in [0,1]$.

$$AUS_j = \sum_{i=1}^{RC} U_i^j / SR_j \tag{5.6}$$

5.3 RECON Evaluation

To evaluate the performance of RECON and show that it provides a robust environment for negotiating agents, we conducted experiments to evaluate the cycle time of agents (e.g. the time it takes for the agent to generate an offer) in the following settings:

- The first set of experiments report the average cycle time of agents throughout a simulation. The aim of these experiments is to show that agents are not affected by the execution of the simulation framework itself, i.e., they do not take a longer time to decide on their actions towards the end of a simulation.

- The second set of experiments report the average cycle time of agents for increasing numbers of agents in the market. The aim of these experiments

is to show that there is no exponential growth in agents' cycle times, i.e., having other agents in the environment has a linear effect on the time it takes for an agent to decide on an action.

RECON has already been used as a simulator to evaluate the negotiating strategy CONAN (Chapter 3). In the planned experiments, we observed that the performance of RECON is steady and it can handle different agent types and market settings. Next, we provide the details of these experiments for the above settings and report the results.

5.3.1 Experimental Methodology

We ran the simulation for five diverse market densities (MD) with up to 200 agents. For each density, 100 runs were performed. Therefore, a total of 500 runs were executed, with each run having buyers and sellers negotiating concurrently. We set the deadline to long ($MDL \in$(151-210 seconds)), the change in demand/-supply to high ($MR \in$ (10:1, 1:1, 1:10)), and the market update time to high ($MC \in$ (10 seconds)). The computer used for the experiment had the following specifications: Intel i5-3450S 2.80GHz processor with 8GB RAM running on Windows 7 Enterprise Service Pack 1 (64-bit Operating System).

Agent selection: For our experiments, we developed an extended version of Faratin's strategies [29] that allows sellers to concurrently negotiate with different buyers. For the buyers, we selected three declarative and three imperative agents. We recorded the average cycle times for each agent type.

5.3.2 Results

The results for three of the five market densities are shown in Figure 5.2. Low density implies that there are 8-12 agents in the e-market. For high, there are 30-50 agents and for very high there are 100-130 agents. In each plot, the average cycle times for the three declarative and imperative buyers were consistent throughout all runs. Changes in the market density did not affect the average cycle time. This

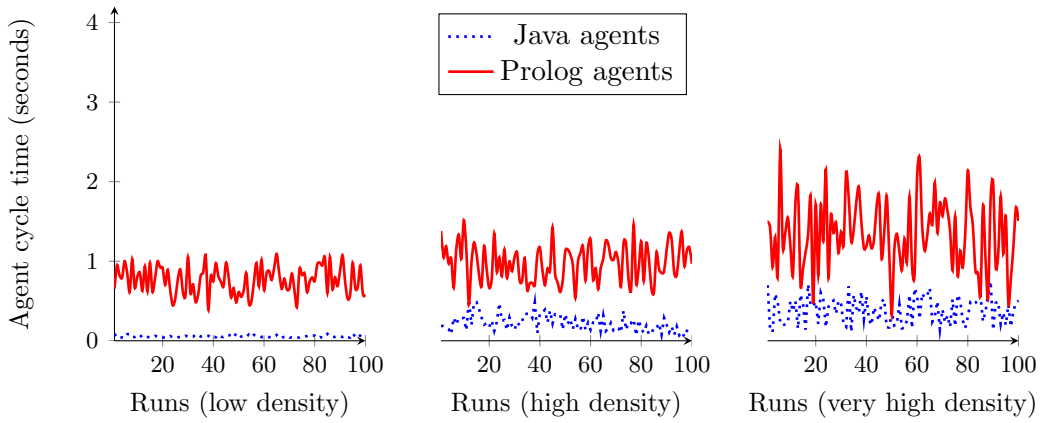

Figure 5.2: Average cycle time for increasing number of runs.

conclusion was derived from the three plots where the average cycle time for the Java buyer stayed within 0.03-0.7 seconds, and for the declarative buyer it stayed within 0.3-2.4 seconds. We expected this difference as the Prolog agents required an extra interface between Java and Prolog, plus the Prolog agent performed more computation, in terms of searching for solution over the strategy rules, which required more options to consider.

The fluctuations in the average cycle time were not caused by Recon, but were rather an outcome of the buyers themselves. Since Recon supports asynchronous negotiation (i.e. where the buyer can negotiate at any time without waiting for all sellers to reply), the buyers replied to different numbers of sellers. For instance, while the agent replies to one seller in one cycle, it might have to deal with three sellers in the next cycle due to the sellers' reply times. Moreover, our buyers have heuristic strategies. Consequently, they may check all the negotiation threads in one simulation, and not in another.

Finally, Figure 5.3 shows that the average cycle time for the three declarative buyers in the 100 runs grew linearly with increasing numbers of agents. We plan to further increase the number of agents in our experiments. This will be carried out in future work.

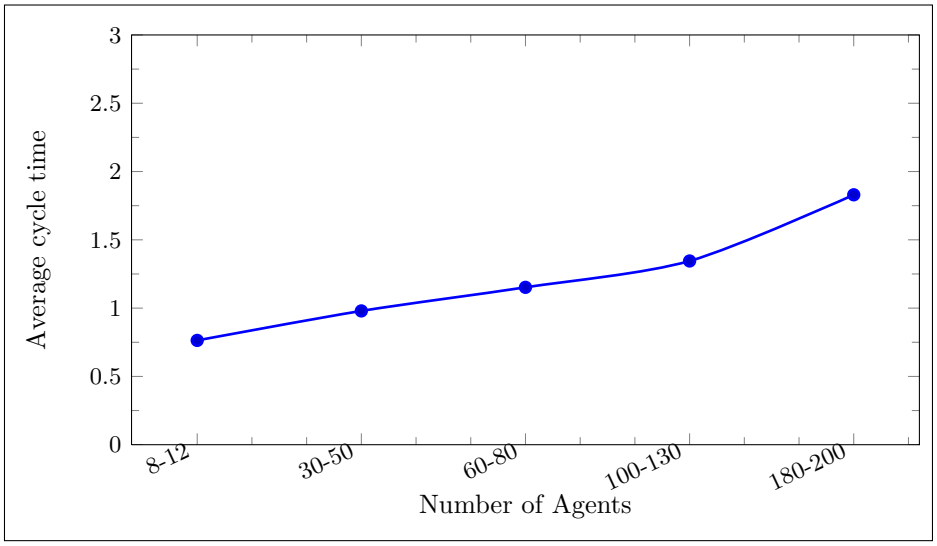

Figure 5.3: Average cycle time for increasing number of agents.

5.4 RECON Graphical User Interface

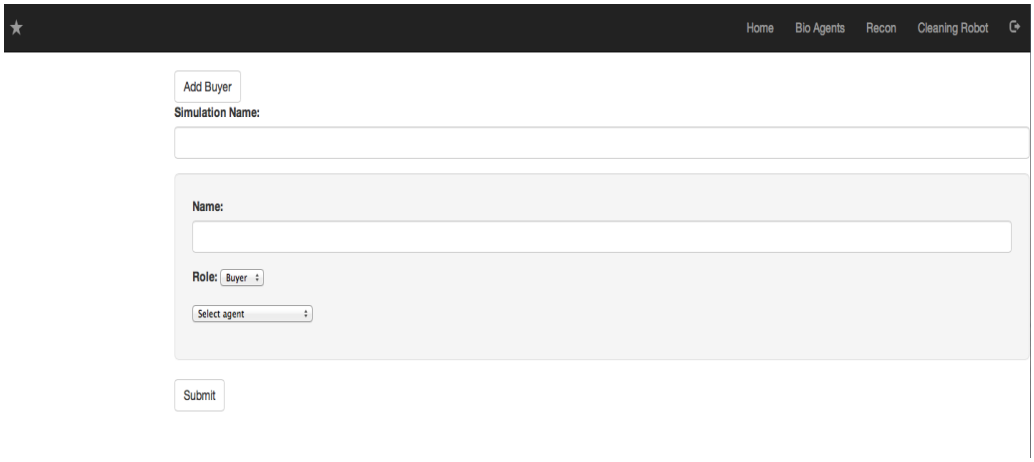

Figure 5.4: RECON supports buyer agents.

We developed RECON using a graphical user interface to allow a distribution version of RECON on the Internet. Users from all around the world can upload their concurrent negotiation agent and benchmark its performance against the latest agents in state-of-the-art systems. The results of the performance will be sent

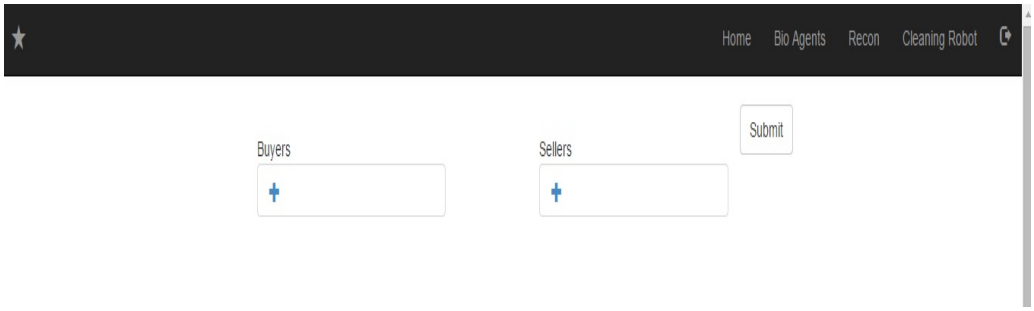

Figure 5.5: RECON supports adding competitors and seller agents.

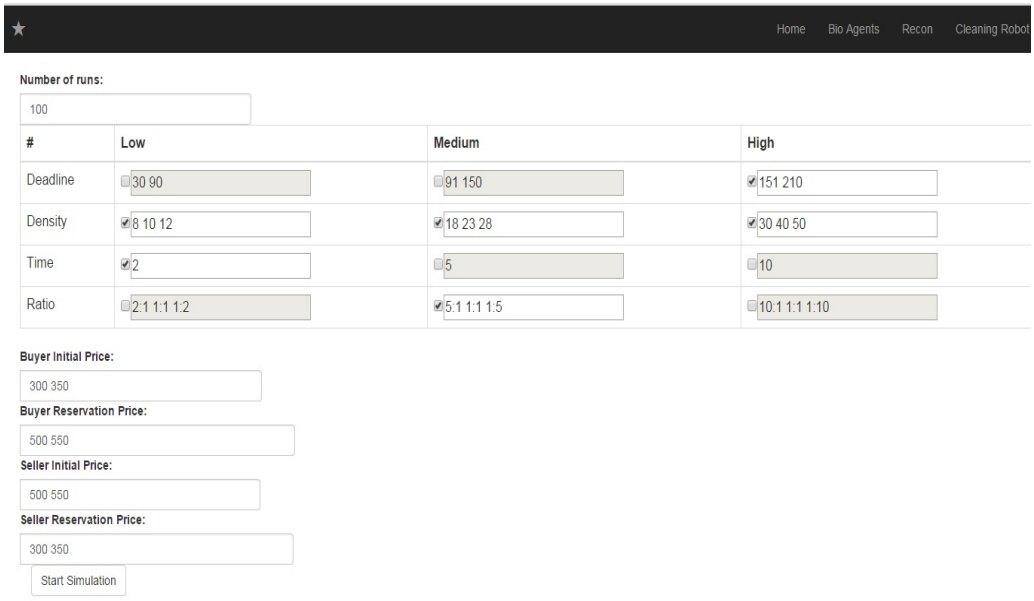

Figure 5.6: RECON simulation parameters.

to the user by email. Figure 5.4 represents a screen-shot of the RECON interface supporting a user to specify the e-market participants of buyer agents that the user wants to measure their performance. Figure 5.5 supports determining all competitors and sellers in the e-market. Figure 5.6 specifies the simulation parameters as described in Table 5.2. In addition, Figure 5.7 shows the results obtained after a specific experiment. The experiment compares the performance of the declarative strategy *Alrayes_w* and the imperative strategy *WilliamsIAmHagglerBuyer* and shows that *Alrayes_w* outperforms *WilliamsIAmHagglerBuyer* in terms of the

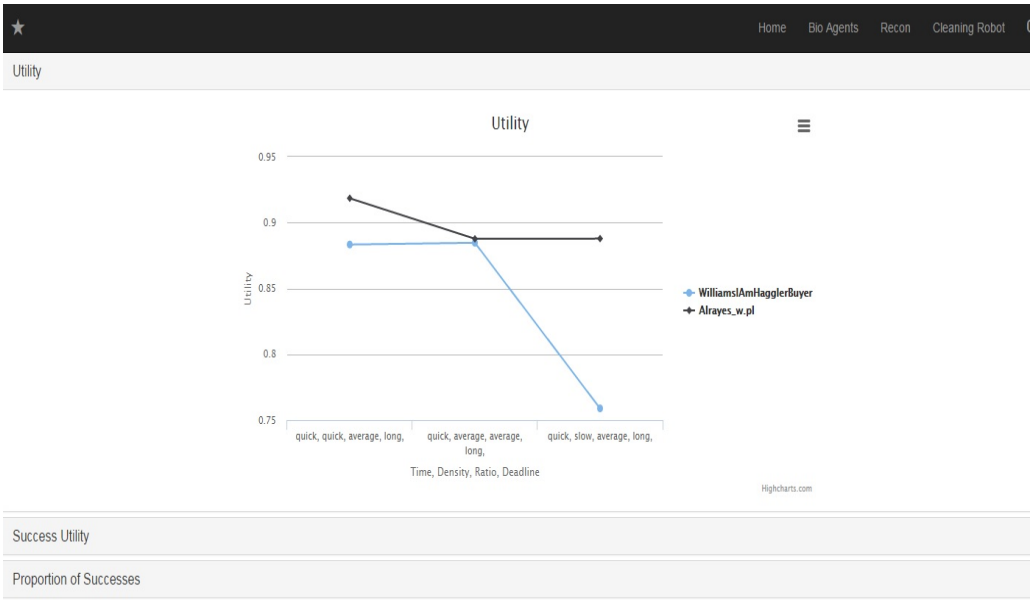

Figure 5.7: RECON results.

average utility in four different market settings, where the:

- market update time is quick (low);
- change in market density are quick (high), average (medium) and slow (low);
- change in demand/supply is average (medium);
- deadline is long (high).

5.5 Summary

RECON is an environment that supports the simulation of concurrent negotiations amongst multiple agents in e-markets. We have shown that RECON was built using a simplified version of the GOLEM agent platform and specialised with a set of infrastructure agents that manage an e-market. We evaluate the performance and robustness of RECON using agents developed imperatively and declaratively. Our experiments with RECON reported on e-market simulations with up to 200 agents negotiating with very reasonable offer-generation times. Moreover, we evaluated

the system under different market settings by testing different simulation scenarios that experimenters might explore in practice and showed that its performance is stable, reliable and scalable under different simulation settings.

In the following chapter, we will evaluate the performance of CONAN compared to the strategy of Williams [115], Random and the extended versions of Faratin's [29] as the benchmark buyers strategies in different e-market settings using RECON, and show that we outperform Williams using statistical measures. In addition, CONAN outperforms Random strategy and the extended versions of Faratin's strategies.

Chapter 6

Empirical Evaluation

In the previous chapter, we proposed our negotiation simulator RECON, which has been developed to evaluate the performance of our negotiation strategy CONAN. In this chapter, we present two large experiments to test CONAN which satisfy Objective 6. We report their results and evaluate their significance.

The first experiment is presented in Section 6.1, where we evaluate the performance of CONAN in different market settings against the performance of the state-of-the-art concurrent negotiation strategy proposed by Williams *et al.* [115]. Then we report the results and show that CONAN outperforms significantly Williams *et al.* Also, to make sure that CONAN also outperforms other buyers, we conduct a second experiment which is presented in Section 6.2, in which we use different settings from the first experiment, with new benchmark strategies (*Random* and *Faratin's et al. [29]*). Similar to the first experiment, we show that CONAN outperforms significantly the benchmark strategies. Finally, we summarize our findings in Section 6.3.

6.1 Experiment 1

The aim of this experiment is to determine whether CONAN can outperform the state-of-the-art and if so, under what settings. This section starts by explain-

ing the experimental setting in detail, which includes the choice of opponents, the benchmark strategy that we are comparing our work with, the parameters of the proposed experiments, and the performance measurements characterising our results. After that, we report on the experimental results.

6.1.1 E-Market Setting

We followed Chapter 4 by implementing a RECON agent that adopts the CONAN strategy. Interaction between agents is handled using inter-agent communication. As explained in Chapter 5, RECON supports both the imperative definition of agent strategies in Java and the declarative specification of logic-based strategies in Prolog, within the same environment. This feature is important, because we can develop simulations with CONAN developed declaratively and then compete with imperative agents, such as those currently representing the state-of-the-art in the negotiation literature, without difficulty. In addition, RECON supports concurrent negotiation which, at the time of our research, existing negotiation platforms did not support [61].

Choice of opponents: For our evaluation, we developed extended versions of Faratin's [29] and ANAC 2011 (International Automated Negotiating Agents Competition) [37] strategies that allow opponents (sellers) to concurrently negotiate with different buyers. In the simulation, we divide these strategies into three groups:

- The first group, *Faratin's Sellers* [29], contains three combinations of strategies which we combine into one group as explained in Section 2.5.2. The first combination is called time-dependent strategies, where the value of the counter-offers and the acceptance value for the offers depend on the remaining negotiation time. The second and third combinations are the resource-dependent and behaviour-dependent strategies. In resource-dependent strategies, sellers concede more rapidly as the quantity of resources becomes limited. As in Faratin *et al.* [29], resources are the number of buyers partici-

pating in the negotiation and the negotiation time. In behaviour-dependent strategies, the seller imitates the behaviour of the buyers, so the seller will compute the next offer based on the previous attitude of the buyer.

- The second group is the *ANAC Sellers* [37]. We chose this group for fair comparison with the benchmark strategy, which we will describe once we have completed this discussion on opponents. In addition, the ANAC competition provides the most recent and most competitive state-of-the-art practical negotiation agents. It is important to note that in our experiments we noticed, in the 10% agreement zone (for more details see Section 3.2), that three ANAC sellers were accepting offers with negative utility for them. This was due to the fact that ANAC sellers always assume that there will be a 100% agreement zone between them and the buyers. To make the comparison fair with ANAC sellers in the 10% agreement zone, we adjusted their strategies to offer or accept only when they obtained positive utility.

- The third group, *All Sellers*, is a combination of all *Faratin's Sellers* and *ANAC Sellers*.

Each seller has two fixed private deadlines. The first is the deadline for how long it will be staying in the market. The second is the deadline related to the negotiation with each buyer. When a seller starts to negotiate with each buyer, it assigns a fixed duration for the negotiation, for instance, 180s. Hence, when the seller starts a new negotiation with another buyer, it negotiates for a fixed time, 180s only. The seller will exit the market if (a) the seller sells all its resources, or (b) the seller reaches the deadline for staying in the market, or (c) the market forces the sellers to exit their negotiations to alter the demand/supply ratio of the market.

Benchmark strategy: We will use the state-of-the-art strategy developed by Williams *et al.* [115] as the benchmark strategy. Williams *et al.* also validated their strategy empirically. The opponents' strategy was obtained from *ANAC*

Sellers only. They compare their strategy with two other strategies: a random strategy and Nguyen and Jennings' strategy [79]. Their results show that their strategy outperforms both. Here, we compare CONAN with the second version of their strategy, where all negotiation threads have the same utility (see Section 2.6.1) and this is due to our assumption that all the sellers provide the same identical resource to be negotiated. We set the simulation parameters based on Williams *et al.* settings in their empirical evaluation in terms of public deadline, break-off probability and penalties.

Simulation parameters: The market allows the agents to enter and leave the market at any time. The market simulates the entrance and the exit of the opponents. For each buyer, the market announces the arrival of new sellers and the total number of buyers. The time of the announcement is based on the parameter *Market update time* as shown in Table 6.1. The number of new sellers and buyers added will depend on the parameters *Change in market density* and *Change in demand/supply ratio*. For instance, if the change rate in market density is 8 and the demand/supply ratio is 1:1, and the market situation has 5 buyers and 7 sellers, then the market terminates 1 buyer and 3 sellers to reach the correct desired change in the market. These terminated buyers and sellers send an *exit* action to their opponents.

Performance Measurements:

To evaluate the performance of CONAN and the state-of-the-art strategy, we use the following performance measurements:

- *Average utility* [29, 79, 115]: is the sum of all average utilities of that agent $U_{ag}(x)$ (see Equation 3.1) over the number of negotiation runs N:

$$Average\ utility = \frac{\sum_{i=1}^{N} U_{ag}^{i}(x)}{N} \tag{6.1}$$

- *Percentage of successful negotiations:* is the total number of negotiation runs where the agent reaches an agreement with one of the sellers over all runs.

Variables	Values
Buyer's Initial price	[300-350]
Buyer's Reservation price	[500-550]
Seller's Initial price	10%[750-800] - 60%[580-620] - 100%[500-550]
Seller's Reservation price	10%[450-500] - 60%[380-420] - 100%[300-350]
Market update time	[2s,5s,10s]
Change in market density	high[30,40,50], average[18,23,28], low[8,10,12]
Change in demand/supply	high[10:1, 1:1, 1:10], average [5:1, 1:1, 1:5], low [2:1, 1:1, 1:2]
Deadline	short[30s-90s], average[91s-150s], long[151s-210s]
Eagerness	[0.1,0.3] [0.4,0.7] [0.8,1]

Table 6.1: Simulation parameter values.

$$Percentage\ of\ successful\ negotiations\ = \frac{number\ of\ successful\ runs}{N} * 100 \tag{6.2}$$

- *Average number of rounds* [47]: is the average number of negotiation rounds until the accept agreement is reached.

$$Average\ number\ of\ rounds = \frac{number\ of\ rounds\ per\ run}{N} \tag{6.3}$$

6.1.2 Experimental Setup

In our evaluation, we allow the *demand/supply ratio* and market density to change during negotiation to create more realistic settings. In addition, as indicated in Table 6.1, we set the eagerness for all threads to 0.5, the deadline to long $[151s - 210s]$ and the market update time to $10s$. We set the same deadline for all buyers and sellers in the e-market for fair comparison with the benchmark strategy since it assumes public deadlines (both buyers and sellers have identical deadlines). To test that CONAN negotiates using private deadlines, CONAN does

not know that buyers and sellers have the same deadline. Also, as we explained in Section 3.4.3, we set the $\lambda = 0.99$, $\mu = 0.9$, $MCO = 1$ and $MAN = 100$. We also set the penalty $D = 0.1$ for fair comparison with Williams's *et al.* [115] strategy.

Similarly, we choose three ranges for the sellers' initial price and reservation price to create three different agreement zones: 10%, 60% and 100%. The reason for this choice is to simulate real-life situations, where the buyer does not know the price range of the sellers. CONAN and the benchmark agent run concurrently within the same simulation as competitors, while in the literature the buyer agents run separately in individual simulations. We run three agents for each strategy (state-of-the-art and CONAN) and run the simulation 100 times for 81 different settings (3 different demand/supply ratio values * 3 different market density values * 3 price ranges * 3 opponent groups)= 8100 runs. We report the average utilities, percentage of successful negotiations and average number of negotiation rounds for those simulations. We also conduct t-tests to report the statistical significance between CONAN and the state-of-the-art utilities for each setting and use ANOVA test (Analysis of variance) to calculate the statistical significance of the resulting utility between each price range for each group of sellers.

6.1.3 Results

The results for *Faratin's Sellers* are shown in Figure 6.1. In the top plot, market density is kept at average and we change the rate for the demand/supply ratio. In the bottom plot, the rate for the demand/supply ratio is kept at average and we change the market density. It can be seen from the plots and from the p value $<$ 0.05 that our strategy outperforms the state-of-the-art according to experimental evidence that is statistically significant. However, the utilities of CONAN and the state-of-the-art are stable during the changes in the demand/supply ratio among the price ranges. On the other hand, the utility of CONAN increases when the market density increases, while the utility of the state-of-the-art decreases when the market density increases. We can observe from Figure 6.1 that:

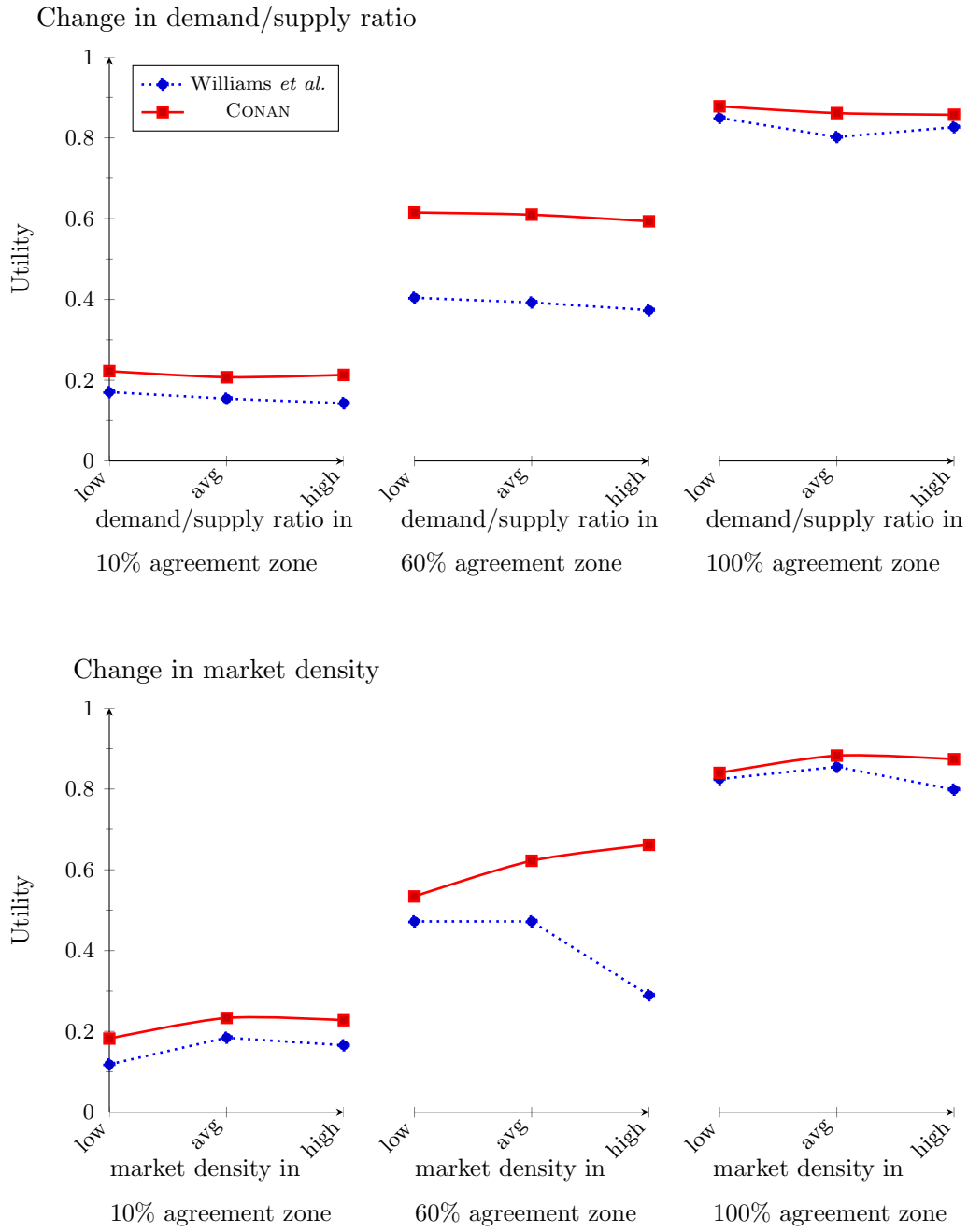

Figure 6.1: Average utility for *Faratin's Sellers*.

- **observation 1:** Conan outperforms the state-of-the-art in all price ranges,

demand/supply ratios and market densities when negotiating with *Faratin's Sellers*;

- **observation 2:** CONAN obtains a significantly higher utility than Williams *et al.* when negotiating in a high density market and a 60% agreement zone with *Faratin's Sellers*. The most likely explanation for this result is that CONAN takes into consideration that the price ranges between buyers and sellers may differ.

- **observation 3:** CONAN achieves the highest utility when the change in demand/supply ratio is low as a result of the market being less competitive, which increases CONAN's opportunity to negotiate with more sellers. Also, in terms of market density, CONAN obtains the highest utility when the market density is high. As a result of the number of sellers increasing, the buyer's opportunity to get an agreement also increases.

The next result is about negotiating with *ANAC Sellers* and is shown in Figure 6.2. Our strategy still outperforms the state-of-the-art ($p<0.05$). The top plots indicate that each strategy gains the same amount of utility when the rate of demand/supply ratio changes, while in the bottom plots, when the market density is average, CONAN has the highest utility. We can observe from Figure 6.2 that:

- **observation 4:** CONAN outperforms the state-of-the-art in all price ranges, demand/supply ratios and market densities when negotiating with *ANAC Sellers*;

- **observation 5:** CONAN obtains significantly higher utility than Williams *et al.* when negotiating in an average density market and 60% agreement zone with *ANAC Sellers*. We observed in the experiments that *ANAC Sellers* cannot cope with high numbers of agents in the market, so as the number of agents rises they become slower in producing conceded offers. As a result

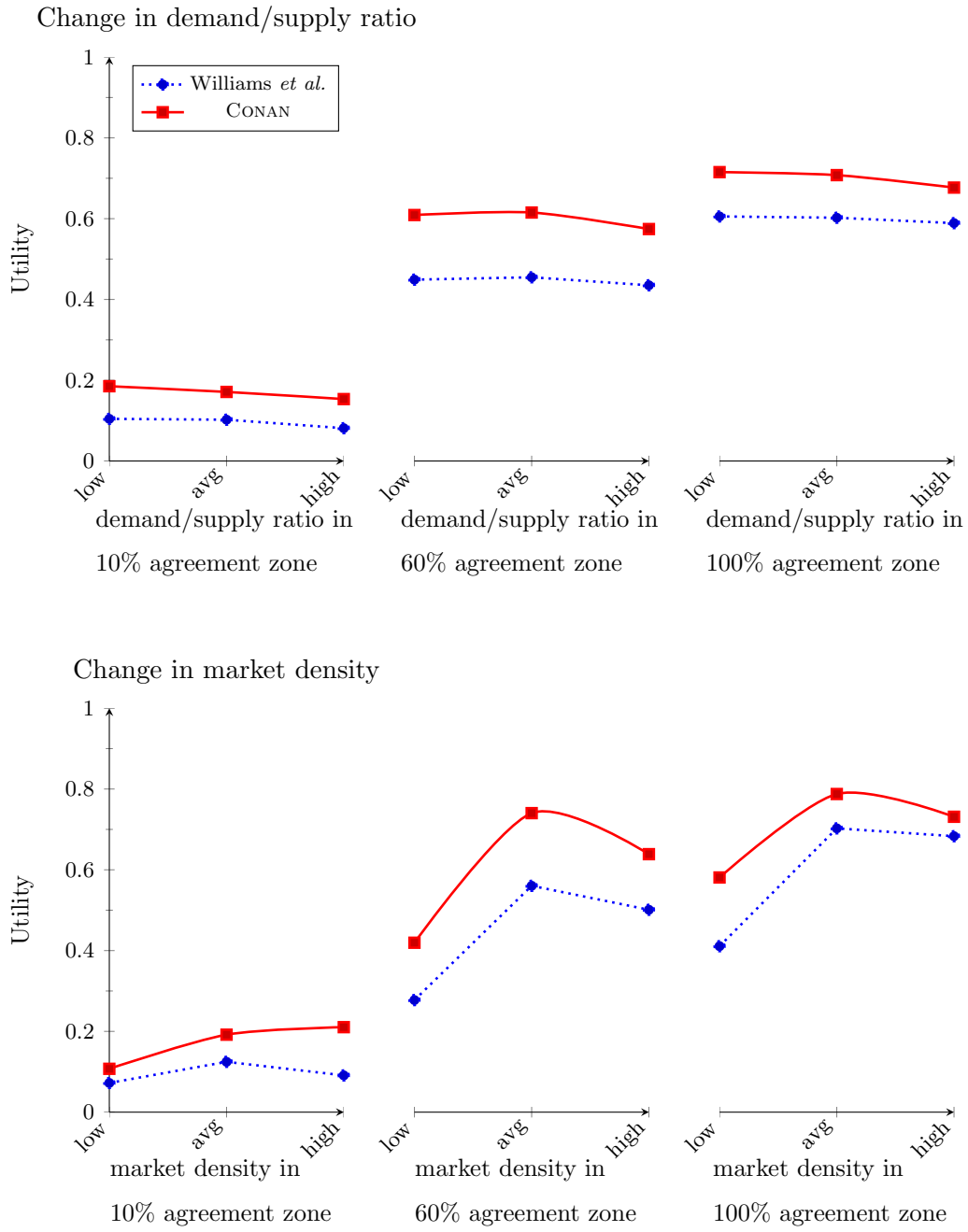

Figure 6.2: Average utility for *ANAC Sellers*.

Conan accepts offers from the *ANAC Sellers* before the deadline since the utility of the offers > 0;

- **observation 6:** CONAN gets the highest utility when the change in demand/supply ratio is low as in *observation 3*. Also, in terms of market density, CONAN obtains the highest utility when the market density is average, as explained in *observation 5*.

In the third group of sellers, *All Sellers*, the results are shown in Figure 6.3. As before, CONAN outperforms statistically significantly the state-of-the-art ($p<0.05$) within each setting and for each group of settings. CONAN obtains the highest utility when the market density is high. We can observe from Figure 6.3 that:

- **observation 7:** CONAN outperforms the state-of-the-art in all price ranges, demand/supply ratios and market densities when negotiating with *All Sellers*;

- **observation 8:** CONAN obtains significantly higher utility than Williams *et al.* when negotiating in an average density market and 60% agreement zone with *All Sellers*, as explained in *observation 5*;

- **observation 9:** CONAN obtains the highest utility when the change in demand/supply ratio is low. Also, in terms of market density, CONAN obtains the highest utility when the market density is high, as explained in *observation 3*.

We can see from Figure 6.4 that CONAN gains more successful negotiations than Williams *et al.* in all the simulation settings when negotiating with *All Sellers*. The top plots indicate that each strategy gains the same percentage of successful runs when the rate of demand/supply ratio changes, while in the bottom plots, when the market density is high, CONAN has the highest percentage of successful runs. We can observe from Figure 6.4 that:

- **observation 10:** CONAN outperforms the state-of-the-art in all price ranges, demand/supply ratios and market densities when negotiating with

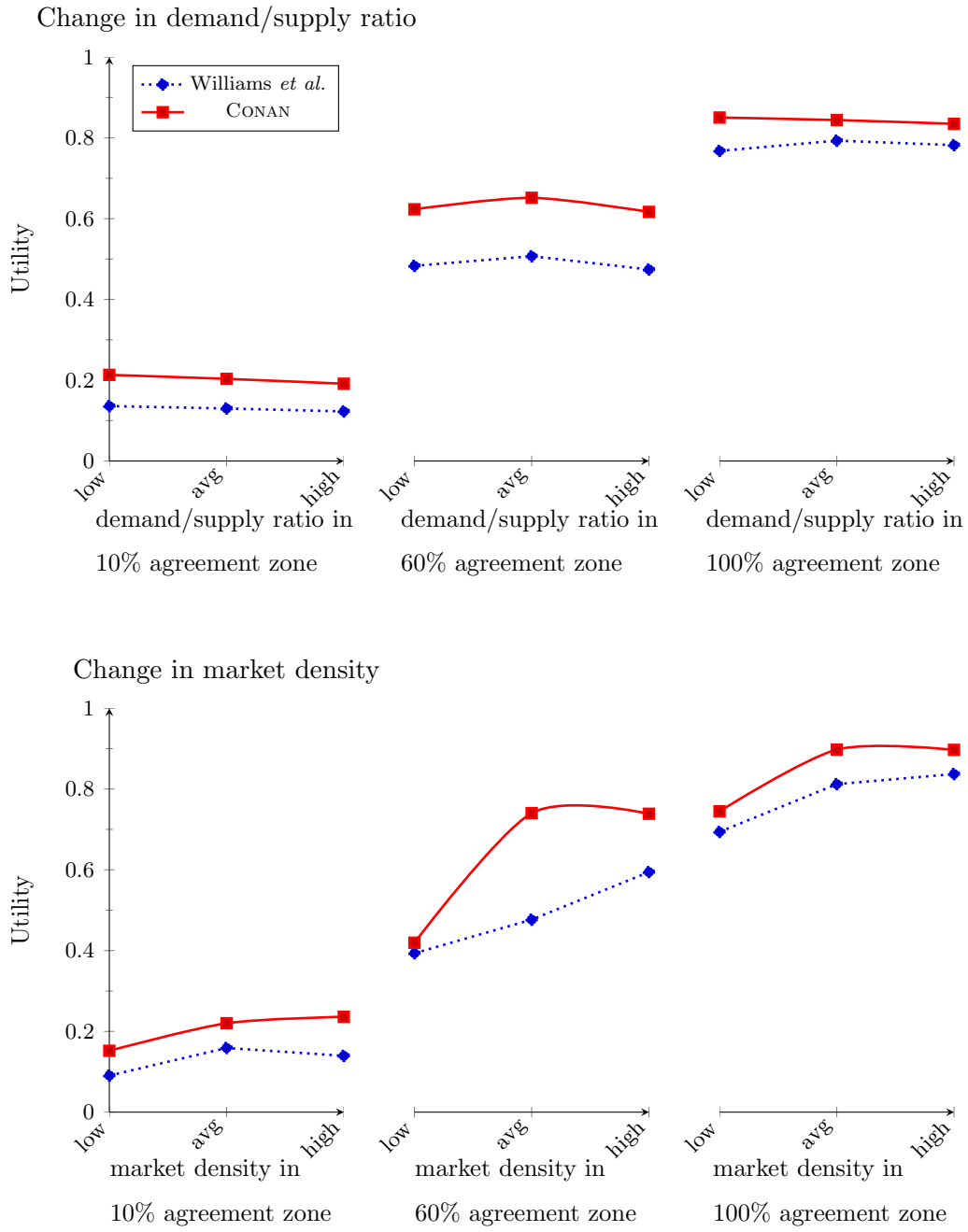

Figure 6.3: Average utility for *All Sellers*.

All Sellers, especially in the 10% and 60% agreement zones. This is because it is hard for any agent to get an agreement in the 10% and 60% zones,

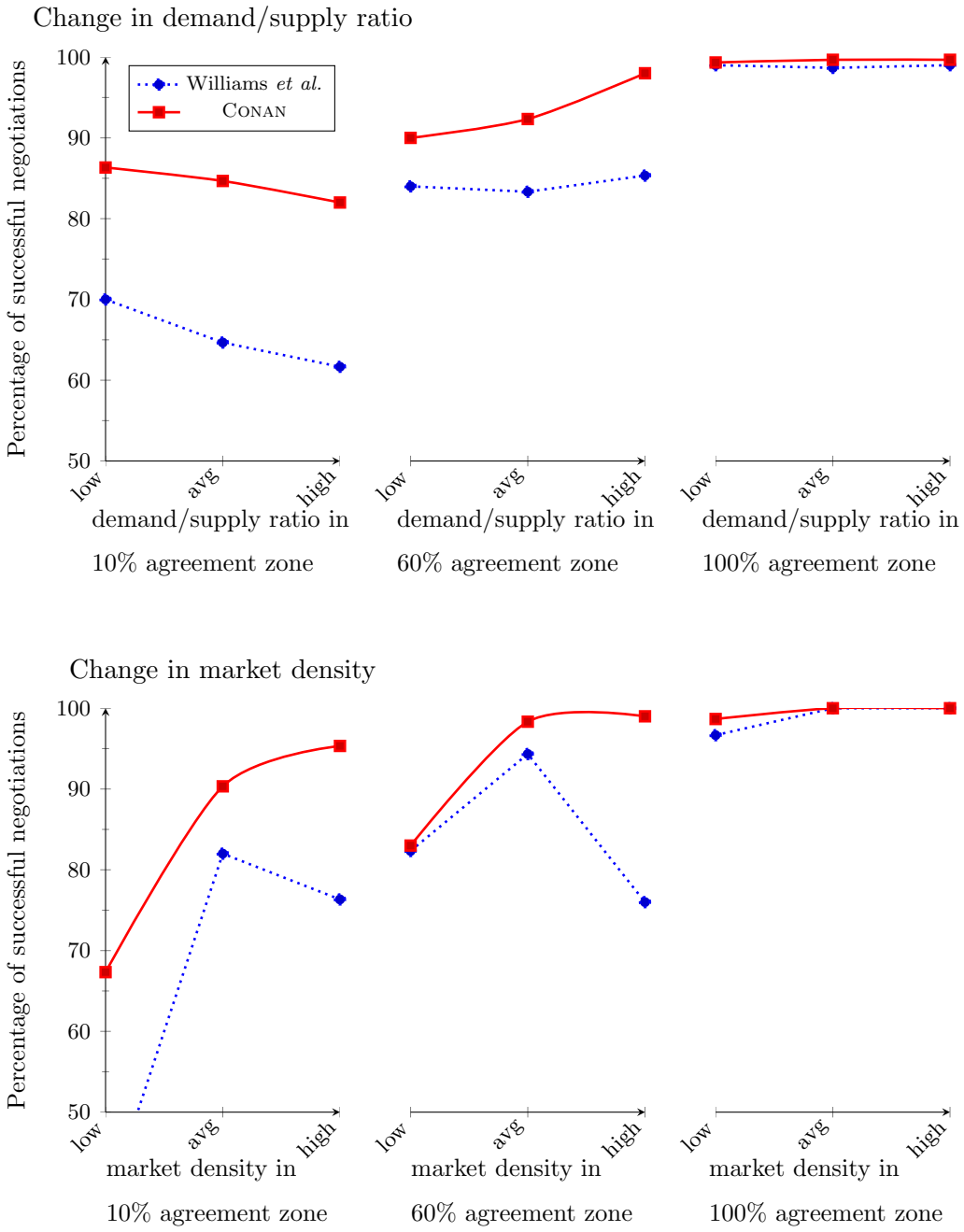

Figure 6.4: Percentage of successful negotiations for *All Sellers*.

while it is easier to get an agreement in the 100% zone, since in the 100% agreement zone, the buyer's initial price $\geq$ the seller's reservation price so

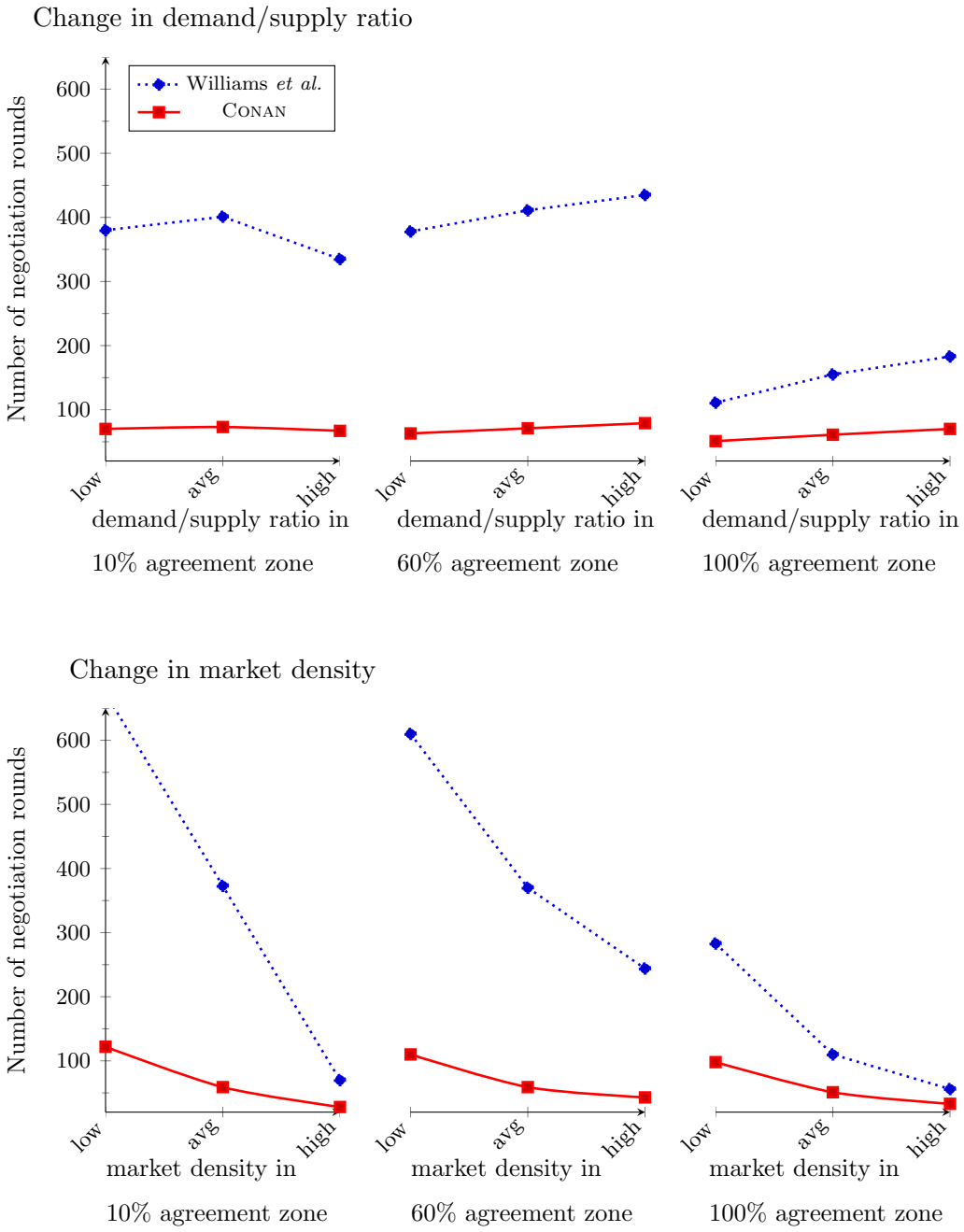

Figure 6.5: Number of negotiation rounds for *All Sellers*.

the utility > 0.

- **observation 11:** CONAN obtains more successful runs when the market density is high in all agreement zones as in *observation 3*.

We can see from Figure 6.5, that CONAN conducts less negotiation rounds compared to Williams *et al.* when negotiating with *All Sellers*. This is because CONAN needs more time to think than Williams *et al.* Thus, CONAN talks less but has more time to think about the best agreement and the best opponent to negotiate with, which minimises the agent's use of communication resources.

The number of rounds are stable (around an average of 100 rounds at low density) for every market density and every demand/supply ratio. On the other hand, Williams *et al.* have a high number of rounds when the market density is low and a low number of rounds when the e-market density is high.

- **observation 12:** CONAN conducts less negotiation rounds than the state-of-the-art in all price ranges, demand/supply ratios and market densities when negotiating with *All Sellers*;

- **observation 13:** CONAN has a stable number of negotiation rounds in all price ranges, demand/supply ratios and market densities when negotiating with *All Sellers*.

The following is a general observation about the results presented in Figures 6.1 to 6.5:

- **General Observation 1:** CONAN is a robust and adaptive strategy that is not affected by changes in the market density and/or in the demand/supply ratio in any price ranges. Experimental evaluation of the strategy produces statistically significant results in terms of average utility compared to the state-of-the-art Williams *et al.* [115]. CONAN also outperforms the state-of-the-art in term of the percentage of successful negotiations, with fewer number of rounds.

6.2 Experiment 2

Given the results of Experiment 1, the aim for this experiment is to determine whether CONAN can outperform other negotiation strategies. In this experiment, we will benchmark more strategies and we will keep the simulation parameters as in Experiment 1 (Section 6.1). The reason for the second experiment is to make the experimentation more credible and convincing by checking the behaviour of CONAN compared to other, equally famous and competing strategies. We did not combine Experiments 1 and 2 because:

- Experiment 1 considers many performance measurements comparing CONAN with Williams *et al.* as the key benchmark strategy. In Experiment 2, we focus on a general overview of CONAN's performance, thus it has only one performance measurement.
- Experiment 1 makes sure that CONAN outperforms Williams *et al.* by running 3 agents for each. In Experiment 2, since we have many buyers to assess, we cannot run more than one agent per strategy in order to keep the low density setting of the e-market as low as possible.
- Experiment 1 tests the performance of Williams *et al.* and CONAN with different groups of opponents (*Faratin's Sellers*, *ANAC Sellers* and *All Sellers*) since Williams *et al.* is the key benchmark strategy. In Experiment 2, we need a general overview of CONAN's performance since *All Sellers* represents both *ANAC Sellers* and *Faratin's Sellers*.

6.2.1 E-Market Setting

As in Experiment 1, we use our negotiation stimulator RECON with the following settings:

Choice of opponents: For our evaluation, we use *All Sellers* strategies. *All Sellers*, as in Experiment 1, is a combination of all extended versions of *Faratin's Sellers* [29] and *ANAC Sellers* [37] strategies that allow opponents (sellers) to

concurrently negotiate with different buyers. The reason for choosing *All Sellers* is that it combines all the opponents that we have, to give a general overview of the performance of CONAN.

Benchmark strategy: We use the *Random* strategy, and the extended versions of *Faratin's strategies et al.* [29] as the benchmark buyers strategies, since there are no more concurrent strategies that fit to our benchmark selection criteria based on the background review in Section 2.6.1.

- The *Random* strategy offer generation function is obtained from the GENIUS platform [1]. The offer is randomly generated and the utility of the offer should be > 0.

- *Faratin's strategies*: There are eight strategies proposed by Faratin *et al.* [29]: *Linear*, *Conceder*, *Boulware*, *Resource Dependent*, *Resource Time Dependent*, *Relative Tit-For-Tat*, *Random Absolute Tit-For-Tat* and *Average Tit-For-Tat*. Faratin's bilateral strategies have already been explained in detail in Section 2.5.2. We chose to benchmark them because they: (a) are simple, popular and widely used [5, 31, 116]; (b) have a clear strategy which is easy to reimplement; (c) have incomplete information about opponents; and (d) use continuous time. However, they use an alternating offer protocol (Section 2.4.3) which is different to our concurrent protocol.

  To make a fair comparison with CONAN, we extended both the Random and Faratin's strategies action selection methods to be able to use the concurrent negotiation actions (Section 3.3), which are request-to-reserve and cancel.

Performance Measurements: We use the average utility as stated in Section 6.1.1. The reason for this is to focus on a general overview of CONAN's performance.

Simulation parameters: As in Experiment 1 (Section 6.1).

[1] http://ii.tudelft.nl/genius/

6.2.2 Experimental Setup

As in Experiment 1, we allow the *demand/supply ratio* and market density to change during negotiation to create more realistic settings. We expand the low density range in Table 6.1 to $[12, 14, 16]$ in order to ensure that the market accommodates the ten benchmark strategies. Also, as in Experiment 1, (1) eagerness =0.5; (2) deadline $\in [151s - 210s]$; (3) market update time =10s; (4) $MCO = 1$; (5) $D = 0.1$; (6) $MAN = 100$; and (7) there are three different agreement zones: 10%, 60% and 100%. CONAN and the benchmark agents run concurrently within the same simulation as competitors. We run one agent for each strategy and run the simulation 100 times for 27 different settings (3 different demand/supply ratio values * 3 different market density values * 3 price ranges * 1 opponent groups)= 2700 runs. We report the average utilities for those simulations. We conduct t-tests to report the statistical significance between CONAN and the benchmark strategies utilities for each market density (high, average and low) and use ANOVA test (Analysis of variance) to calculate the statistical significance of the resulting utility between each price range.

6.2.3 Results

Figure 6.6 depicts the result of negotiation with *All Sellers* in the 10% agreement zone. As in Experiment 1, in the top plot, market density is kept average and we change the rate for the demand/supply ratio. In the bottom plot, the rate for the demand/supply ratio is kept average and we vary the market density.

It can be seen from the plots and from the p value < 0.05 that our strategy CONAN outperforms the Random and Faratin's strategies according to experimental evidence that is statistically significant. From the results we can say that the utility of CONAN, Random and Faratin's strategies is stable during changes in the demand/supply ratio. On the other hand, the utility of CONAN increases when the market density increases, while the utility of *Linear, Conceder, Boulware and Resource Time Dependent* decreases when market density increases.

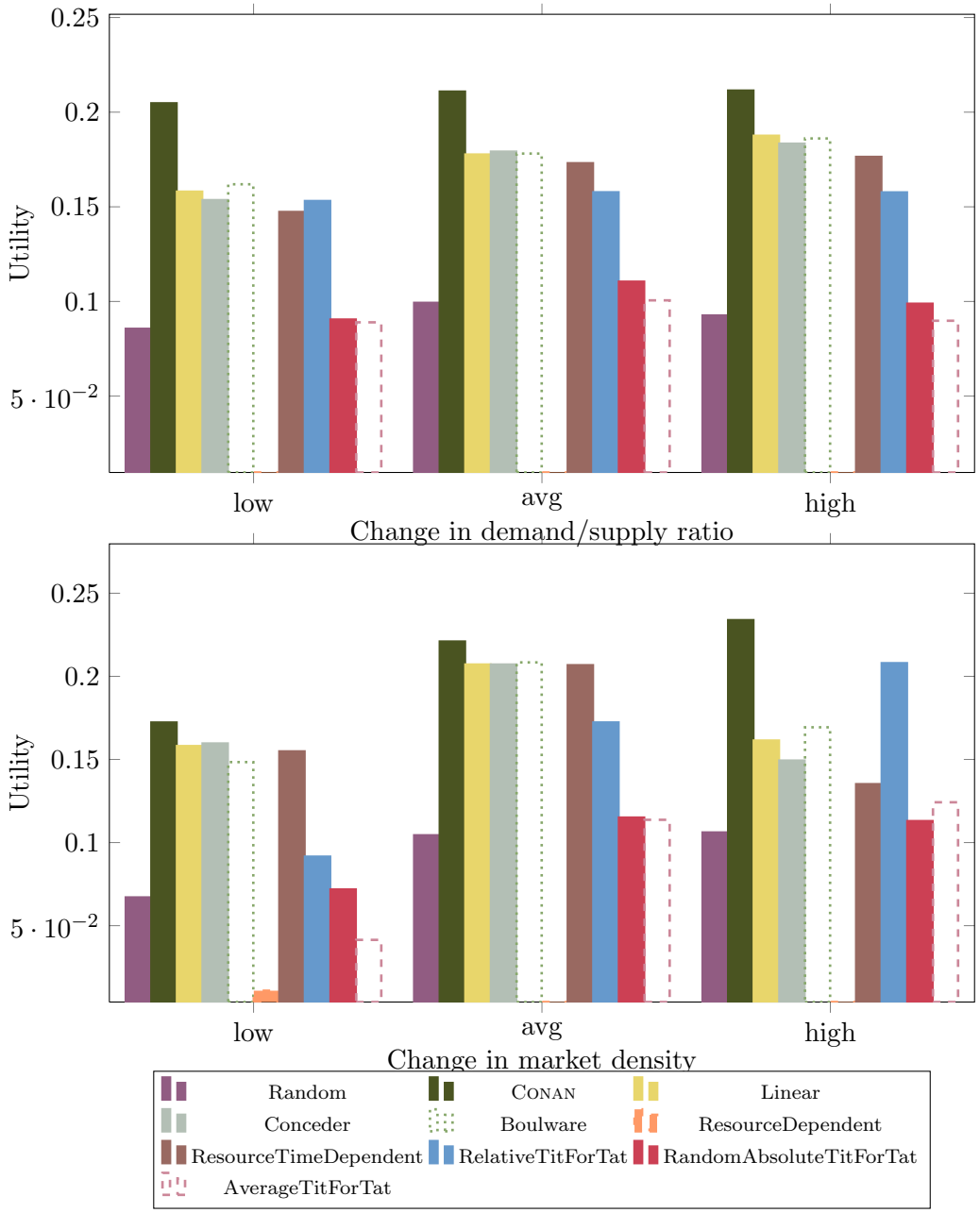

Figure 6.6: Average utility for *All Sellers* with 10% agreement zone.

Figure 6.7 shows the result of negotiation with *All Sellers* in the 60% agreement zone. As in Figure 6.6, Conan still outperforms the benchmark strategies ($p<0.05$) and the utility of Conan, Random and Faratin's strategies is stable

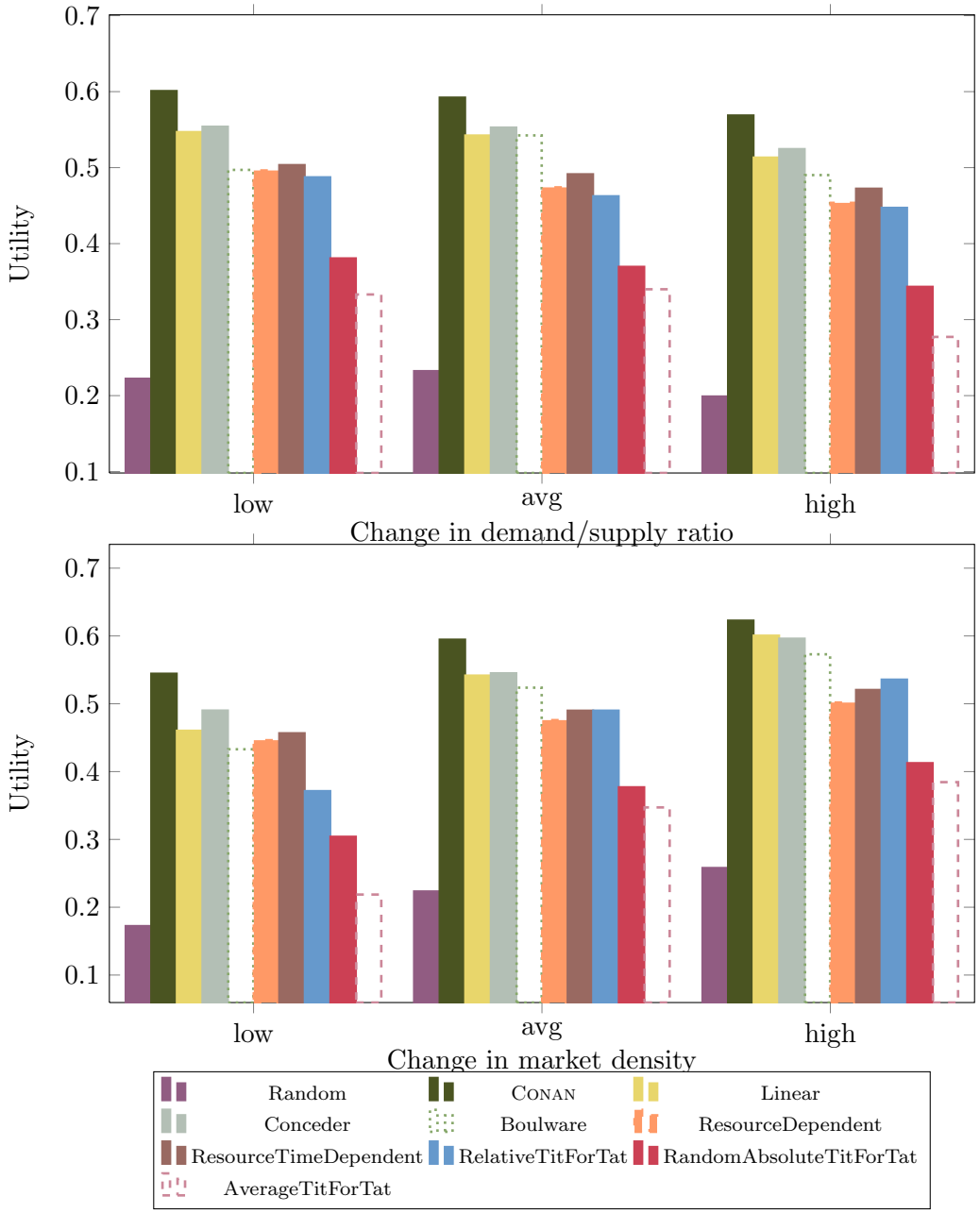

Figure 6.7: Average utility for *All Sellers* with 60% agreement zone.

during changes in the demand/supply ratio. The utility of CONAN increases when the market density increases, and the utility of Random and Faratin's strategies increases when the market density increases.

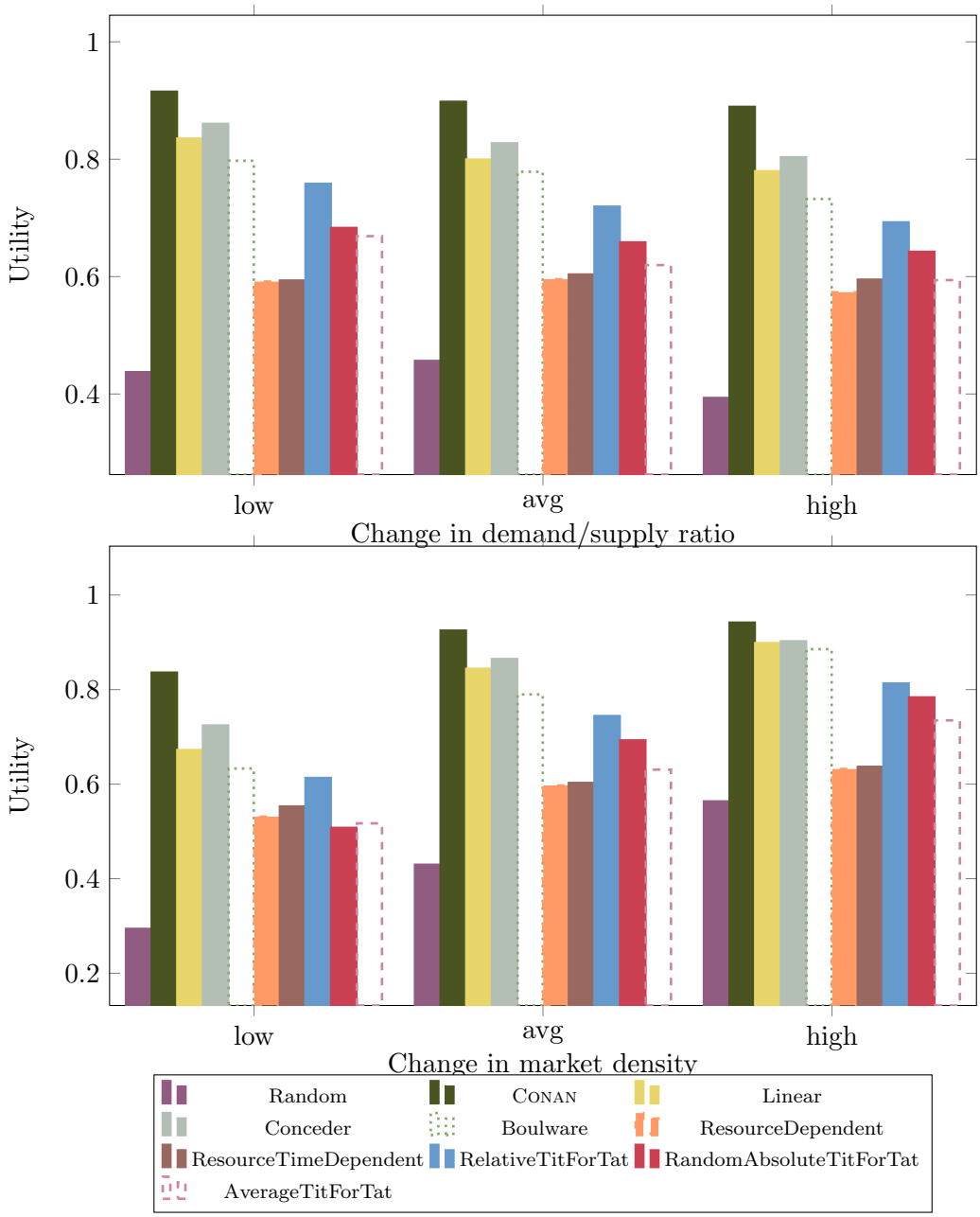

Figure 6.8: Average utility for *All Sellers* with 100% agreement zone.

Figure 6.8 shows the results for negotiation with *All Sellers* in the 100% agreement zone. As before, Conan outperforms significantly the benchmark strategies ($p<0.05$) and the utility of Conan, Random and Faratin's strategies is stable

during changes in the demand/supply ratio. The utility of CONAN increases when the market density increases, and the utility of Random and Faratin's strategies increases when the market density increases.

For this section we combine all the observations from Figures 6.6 to 6.8:

- **observation 14:** CONAN outperforms the Random and Faratin's strategies in the 10%, 60% and 100% price ranges, demand/supply ratios and market densities when negotiating with *All Sellers*;

- **observation 15:** CONAN obtains significantly higher utility than the Random and Faratin's strategies (except *Relative Tit-For-Tat*) when negotiating in a high density market in the 10% agreement zone in Figure 6.6. As with the Williams *et al.* strategy, this is because it is easy for any agent to negotiate in the 100% agreement zone and get a better agreement rather than negotiating in the 10% agreement zone, which is more difficult. Also, CONAN obtains significantly higher utility than the Random and Faratin's strategies when negotiating in a low density market in the 100% agreement zone in Figure 6.8. This shows that CONAN can perform well even in a simple market setting where the number of agents is low in a 100% agreement zone;

- **observation 16:** In Figures 6.7 and 6.8, CONAN obtains the highest utility when the change in demand/supply ratio is low as a result of the market being less competitive, which increases CONAN's opportunity to negotiate with more sellers. Also, in terms of the market density, in Figures 6.6 to 6.8, CONAN obtains the highest utility when the market density is high, as a result of the number of sellers increasing, which increases the buyer's opportunity to reach an agreement.

The following is a general observation about the results presented in Figures 6.6 to 6.8:

- **General Observation 2:** We can see that modelling random behaviour,

time of negotiation or opponent behaviour is not enough to increase the negotiation utility. Experimental evaluation of CONAN shows that it outperforms significantly the Random and Faratin's strategies in terms of average utility.

6.3 Summary

We evaluated the performance of CONAN in two experiments. In the first experiment, with various settings, we showed that CONAN performed significantly better than the state-of-the-art [115] in terms of average utility. We also showed that CONAN's utility is not affected by changes in the market density, the demand/-supply ratio, and/or the price range. In addition, the performance of CONAN was tested in the 10%, 60% and 100% agreement zones. We also showed that CONAN outperforms the state-of-the-art [115] in terms of the percentage of successful negotiations, with fewer negotiation rounds.

In the second experiment, we showed that CONAN outperforms significantly both Random and Faratin's [29] strategies in terms of average utility when negotiating with all sellers under the 10%, 60% and 100% agreement zones with different market densities and demand/supply ratios.

The experimental results validate the properties of CONAN mentioned in Chapter 3. We can see from the results that Property 1 has been confirmed in Figures 6.6-6.8 where CONAN concedes less (more utility) as the number of sellers (high market density) increases. Property 2 has been confirmed in Figures 6.6-6.8 where CONAN concedes more (less utility) as the number of competitors increases. In addition, from the results we can see that CONAN behaves rationally by producing monotonic offers.

Chapter 7

Conclusions and Future Work

This Chapter provides final remarks, along with an overview of the thesis work and a summary of its contribution to the field of negotiation. It also presents a discussion of future research directions.

7.1 Review and Discussion of the Achievements

We began this thesis by introducing the challenges involved in automated negotiation in Chapter 1. Chapter 2 presented the background to our negotiation model.

We introduced our negotiation model in Chapter 3 where we proposed a concurrent negotiation architecture (Objective 1) designed as a specialized extension of previous work with the KGP model [36, 107] to satisfy the requirements of concurrent bilateral negotiation. We integrate models of the environment, the opponents and self in a revised agent architecture to aid the symbolic representation of decision-making in negotiations. Existing work has provided architectures for bilateral negotiation only, with protocol- and strategy-dependent features. Another limitation is lack of description of the behaviour of the information through the architecture components. Addressing these limitations is important to provide adaptable designs to be used by the agent developer to implement any negotiator concurrent strategies, thus saving the cost of developing the architecture from

scratch.

We presented a concurrent negotiation protocol (Objective 2) by producing a revised version of the well-known alternating protocol that can support concurrent negotiations for open e-markets. However, in realistic applications, and especially in concurrent negotiation settings where a buyer agent engages in multiple bilateral negotiations in order to acquire a resource, agents need more complex negotiation actions to handle different negotiation situations. Thus, we contribute to the state of the art by: (a) developing two protocols to rule the concurrent negotiation: the individual bilateral negotiation protocol and the overall negotiation protocol; and (b) adding more action to the well-known alternating protocol (*request-to-reserve*, *reserve*, *cancel* and *exit*), thus providing the necessary flexibility that was not always possible in previous work with limited actions (*offer(x) and accept*).

To test our negotiation protocol we studied the development of agent strategies with an emphasis on the strategy of a buyer agent in an open and dynamic environment. The existing literature (e.g. [79, 115]) provides answers only on when to offer or accept and what to offer, and thus lacks explanations of how to choose negotiation actions, so that the agent knows when to request-to-reserve, cancel and exit the negotiation. Offers are computed using an opponent's model but ignoring both the environmental and self models [79, 115], while other work assumes complete and certain information about the negotiation environment and the opponents [79]. In addition, existing models often make strong assumptions about the domain, e.g. that deadlines are public, negotiation time is discrete and there is one agreement zone (Section 3.2), thus constraining their applicability in a variety of practical negotiation settings.

To overcome these shortcomings, we developed a heuristic negotiation strategy, Conan (Objective 3). Conan considers a weighted combination of modelling the e-market environment and the progress of concurrent negotiations in which the agent is involved. Also, it provides explicit decisions about when to offer, accept, request-to-reserve, cancel or exit a negotiation. This development will increase the possibility of achieving an agreement and minimise lost opportunities to add value

to the agent's utility function and better imitate real life negotiations.

Furthermore, in order to provide a high level of abstract reasoning, we formalized our strategy CONAN in a logic-based knowledge representation using the Event Calculus (objective 4). To the best of our knowledge, we believe there is a lack of using Event Calculus to formalize negotiation strategies. Our formalization can help the agent in the future to justify each action taken in the negotiation.

We built a negotiation framework, RECON (Objective 5), an experimental simulation platform that supports the development of software agents interacting concurrently with other agents in negotiation domains. Unlike existing negotiation simulation toolkits that support only bilateral negotiation and imperative negotiation strategies [61], RECON also supports concurrent declarative strategies, for applications where logic-based agents need to explain their negotiation decisions to a user. RECON is built on top of the GOLEM [1] [18] agent platform, specialized with a set of infrastructure agents that can manage an e-market and extract statistics from the negotiations that take place. We evaluated the performance and robustness of RECON using agents developed imperatively and declaratively. Our experiments with RECON report on e-market simulations with up to 200 agents negotiating with very reasonable offer-generation times. Moreover, we evaluated the system under different market settings by testing different simulation scenarios that experimenters might explore in practice and showed that its performance is stable, reliable and scalable under different simulation settings. To the best of our knowledge, RECON is the only platform that simulates any concurrent negotiation agent that has imperative or declarative strategy.

Our experimental results confirm that it is not enough to model only the opponent(s) in an open e-market; we need also to model the environment and the status of the agent's concurrent negotiations (both individually and as a whole). By varying (a) the density of the market, (b) the demand/supply ratio and (c) the zones of agreement between the negotiating agents, the results show that our strategy outperforms the state-of-the-art strategy in terms of average utility and

[1] http://golem.cs.rhul.ac.uk/

successful negotiations gained from negotiations (Objective 6). In addition to the most competitive strategy in the state-of-the-art, CONAN gained a higher utility compared to Random and Faratin's strategies. We also demonstrated that our experimental results for the average utility were statistically significant. Most existing approaches have not conducted performance comparisons with any benchmark strategies. Benchmarking other strategies will help to advance the state of the art in the concurrent negotiation field.

The implications of our work are that the **proposed model opens up new possibilities for developing more realistic and flexible models for negotiating agents in e-markets** and contributes to the long tradition of the multi-agent systems area by reporting results on the role of agent technology for automated negotiation in practical settings.

7.2 Future Research

This thesis lays the ground for some future work, as discussed below.

7.2.1 Malicious Negotiation Actions

In our negotiation protocol we assume that the agents are trustworthy, in that they follow the protocol without breaching it at any time during negotiation. However, in an open and dynamic negotiation environment, an agent may behave maliciously and untrustworthily. For instance, if the agent did not follow the negotiation protocol by using a new action (i.e. wait) that would be considered a malicious behaviour. One potential way to avoid inappropriate actions from these agents is to develop a strategy to deal with malicious actions. It would be useful to design a strategy as suggested by Neville *et al.* [76] to enable agents to cope with malicious actions. Neville *et al.* [76] present a computational socio-cognitive framework that formalises social theories of trust, reputation, recommendation and learning from direct experience, which increases the system's protection from such undesirable behaviour. In addition, in CONAN, penalties for canceling an offer

are based on a percentage of the deal. It would be interesting in further work to study the impact of different cancellation penalties (as introduced in Section 2.6.3) on the average utility gained from the negotiation. For instance, An *et al.* [3] negotiated cancellation penalties in conjunction with the price of the resource under negotiation.

7.2.2 Weights for Self and Environment Factors

In our strategy CONAN, we used a heuristic method to assign weights to the environment and self factors. Another direction that could be taken is the use of machine learning techniques to improve the function that assigns weights to the environment and self factors, thus making the negotiation more adaptive, without compromising the increase in the agent's average utility.

In addition, in CONAN, we used equal weights for each sub-factor in the environment and self factors. In the self factor S_t we used 0.25 and in the environment factor E_t we used 0.33 to weight the sub-factors. To maximize the agent's utility, it would be interesting to develop a strategy to assign different weights for each e-market setting for each sub-factor.

Also, it would be of interest to measure the effect of the environment factors on the opponent's behaviour. For instance, if the number of sellers increases in an e-market, will that affect the concession rate of the sellers?

7.2.3 Opponent Model

Opponent models have recently become an important part of any negotiation model [12, 45, 62]. However, Baarslag *et al.* [11] note that opponent models do not have a huge effect (low importance) on the agent's utility compared to the offer generation and action selection strategies, which are more important. A future direction is therefore to study the effect of including the opponent model in the concession rate (see Equation 3.3) on the amount of negotiation utility. However, it is important to keep the opponent model as simple as possible in terms of

computational complexity since the opponent model will not have a major effect on the agent's utility [11].

Adopting a suitable opponent model may improve the negotiation performance of CONAN. Machine learning can be used to predict the opponent model during and across negotiation runs. Possible directions could be based on works like the one of Hindriks *et al.* [46] who used Bayesian learning to predict the issue preferences of an opponent and their values, or like the one of Ren *et al.* [90] who predicted opponent strategy using regression analysis.

7.2.4 Declarative Strategy

One of the reasons why we implemented CONAN as a declarative strategy is that it offers a transparent model for decision making for the agent's user. It also offers the possibility to formally state and verify the properties of agents' behaviour [51]. In future research, it would be important to prove the properties of CONAN, check the consistency of CONAN's rules for selecting negotiation actions to achieve its goal of maximizing the agent's utility and verify that CONAN is a compliant strategy. Verification refers to checking the actions of an agent at design time to ensure that the agent will behave as desired in all simulation runs. Compliance means checking the behaviour of an agent at run time to determine if it behaves as desired [24, 40].

In addition, it would be interesting to prove the concurrent protocol properties. For example, Guerin and Pitt [39] provided a procedure by which they could verify the properties of the protocol. It is also important to make sure that CONAN handles conflicts arising from applying the negotiation rules. For instance, Bikakis *et al.* [14] proposed a formal method to resolve potential inconsistencies that may arise from the interaction of contexts. As result of proving the consistency of agent behaviour, it may be possible for an agent to explain to its own user why it takes a particular action.

7.2.5 Computational Time

Since the negotiation agents have limited computational resources, and we assume that the e-market is open and dynamic, with different numbers of sellers and competitors entering and exiting the e-market, an agent consumes time in replying to all sellers simultaneously. It is therefore important to develop an algorithm to minimise the agent's computation time by:

- determining the most promising sellers (sellers with good behaviour based on specific criteria) as negotiation progresses, and replying to them as shown in Algorithm 2. Then, gradually decreasing the number of concurrent negotiation sellers by exiting from negotiations with sellers that fall into the third and fourth groups in Algorithm 2.

Algorithm 2: Promising CONAN

Input: TE, $ThreadIds$

Output: agreement or no agreement

No.promising threads $= (1 - TE) * |ThreadIds|$
if *No. promising threads* > 0 **then**
    Reply using Equation 3.3 to following Promising Threads:
    First respond to all compatible sellers with High eagerness
    Second respond to all moderately compatible sellers with High eagerness
    Third respond to all incompatible sellers with High eagerness
    Fourth respond to any seller with Medium eagerness
else
    Use CONAN

- improving the matching strategy between buyers and sellers, so the buyer will be able to choose the best opponents before the start of the negotiation. The agent will thus have to negotiate with some of the sellers available in the

e-market. For instance, Munroe *et al.* [74] proposed an opponent matching mechanism based on trade-offs made between conflicts the seller expected to bring to a negotiation and the expected cost when negotiating with the seller.

7.2.6 Negotiation Over Multiple Issues

Our current agent only considers one issue (e.g. price) rather than multiple issues (e.g. price, warranty or colour). By considering multiple issues an agent will be better positioned to simulate real-life applications, such as negotiating with users. Multiple issues relating to the resource under negotiation require agents to deal with multiple challenges, the most important challenge being whether the value of an issue is dependent on other issues. For instance, if the price of a laptop depends on the length of the warranty, then the longer the warranty, the higher the price of the laptop. This challenge is still an open problem in automated negotiation.

In addition, our negotiation simulator Recon only supports single-issue negotiation between agents. As the agent's strategy will in the future deal with negotiating multiple issues, there will be a need for Recon to be improved to handle multi-issue negotiation. This will include (1) support of quantitative and qualitative issues; and (2) handling different ways of negotiating issues either sequentially or as a package deal (Section 2.2.2).

7.2.7 Performance Metrics

For the negotiation platform Recon presented in Chapter 5, there are a number of future research directions. First, in the current design, Recon provides some performance measurements (e.g. the average utility). In future work, we will include additional performance metrics in the analysis step (Section 5.2.3). Providing more performance measurement functions: (1) helps the agent designer to improve their agent's performance, which will develop the automated negotiation field in general; and (2) gives a better insight into why agents behave in certain

ways during negotiation.

Performance metrics will include: (1) Time of Agreement - the normalized time at which an agreement is established; (2) Best Acceptable Bid - the utility of the best bid offered to the agent; (3) Number of concession moves - the number of moves where the agent increases its opponent's utility and decreases its own utility; (4) Number of reserved and canceled offers - the total number of reserved and canceled offers; and (5) Penalties paid - the value of penalties paid as a result of the canceled offers.

In RECON, we ran the simulation for various market densities with up to 200 agents. Another direction is to conduct experiments with larger numbers of agents (above 200). The reason for this is that the more agents RECON can handle, the more the e-market becomes competitive and popular and can be used to simulate real e-markets on the Internet.

7.2.8 Negotiation With Humans

In our model all the participants in negotiations are agents. In the future, agents will have to negotiate with humans [60] since in real-life situations not all people can afford to deploy an agent to negotiate on their behalf. However, humans negotiating with agents will introduce many more challenges into the negotiation. Emotion, culture and negotiation duration will be some of the factors that agents must take into consideration when negotiating with humans. The main path for future investigation is research into adapting human avatars to take the place of negotiating buyers and sellers. Human avatars have already been developed in GOLEM to provide human interaction with agents, which need to be improved to take emotion and more complex negotiation language into consideration. One interesting area of research proposed by Rosenfeld *et al.* [94] is to provide an agent that is able to negotiate with agents and humans at the same time to model a real application.

Appendix A

Agents in RECON

Appendix A: Agents in Recon

This appendix briefly describes the basic methods used by the market agents. Listings A.1 and A.2 presents examples of a Java buyer agent and a Prolog buyer agent, respectively. Note that the offer generation processes of the agents solely depend on their strategies and not on RECON. Seller agents can be implemented similarly.

```
1  public class SimpleBuyer extends AbstractBuyerAgent{
2   public SimpleBuyer(AgentBrain brain, AgentParameters params, String
        product){
3    super(brain, params, product);
4   }

6   @Override
7   protected List<Action> decideActionBasedOnOffer(NegotiationAction
        offer){
8    List<Action> actionsToPerform = new ArrayList<>();
9    double utilityOpponentOffer = getUtility(Double.parseDouble(offer.
        getValue()));
10   double counterOffer = generateNextOffer(offer.getDialogueId());
11   double utilityMyCounterOffer = getUtility(counterOffer);
12   boolean isOfferAcceptable = isOfferAcceptable(
```

```
13              utilityOpponentOffer,
14              utilityMyCounterOffer,
15              getNormalisedTime(getStartTime()));
16   if (isOfferAcceptable){
17    actionsToPerform.addAll(super.acceptOpponentOffer(offer));
18   }
19   else{
20    actionsToPerform.addAll(super.sendCounterOffer(offer,
          counterOffer));
21   }
22   return actionsToPerform;
23  }

25  @Override
26  protected List<Action> decideActionBasedOnAccept(NegotiationAction
        accept){
27   switch (getRandom().nextInt(NUM_RANDOM_ACTIONS)){
28    case ACCEPT:
29     return super.acceptOpponentOffer(accept);
30    case RequestToReserve:
31     return super.commitToMyLastBid(accept);
32    case EXIT:
33     return super.exitFromDialogue(accept.getDialogueId());
34   }
35   return new ArrayList<>();
36  }

38  @Override
39  protected double generateNextOffer(String dialogueId){
40   return getInitialPrice() + getRandom().nextInt(getReservationPrice
         () - getInitialPrice());
41  }

43  private boolean isOfferAcceptable(double utilityOpponentOffer,
        double utilityMyCounterOffer, double time){ }
```

Listing A.1: Java buyer agent.

In Listing A.1, the Java buyer inherits the basic methods from AbstractBuyerAgent (line 1). The method `decideActionBasedOnOffer()` (line 7) returns an action based on the seller's offer. The method `getUtility()` (line 9) returns the utility of the seller offer. The method `generateNextOffer()` (line 10) calls the method in line 39 to generate the buyer next offer. This is part of the buyer strategy. The method `isOfferAcceptable()` (line 12) calls the method in line 43 to decide if the opponent's offer is acceptable by comparing the buyer offer with the opponent offer. This is part of the buyer strategy. The method `decideActionBasedOnAccept()` (line 26) returns an action based on the opponent accept action.

In Listing A.2, is part of the agent strategy. The predicate `concession(ThreadId, CA, T)` (line 1) calculates the concession rate. `calc_next_offer(ThreadId, Offer, T)` (line 5) generates the counter-offer at time `T` for thread id (`ThreadId`) based on: initial price (`Min`), reservation price (`Max`) and concession rate (`CA`). The predicate `select(exit(ThreadId, Item), T)` (line 11) returns the action exit for all sellers with thread id(`ThreadId`) that negotiate for item `Item` at time `T`. This decision will be taken if the buyer reaches its deadline. The predicate `select(offer(ThreadId, Item, Offer), T)` (line 17) returns an offer (represented by the variable `Offer`) for seller with thread id (`ThreadId`) and item under negotiation (`Item`). This predicate calls a function to generate the counter-offer at line 5 after checking the buyer's deadline.

```
1 concession(ThreadId, CA,  T):-
2       CA is 0.75.

4 % calculate an offer
5 calc_next_offer(ThreadId, Offer, T):-
6      concession(ThreadId, CA, T),
7      holds_at(ip(Id, Min)=true, T),
8      holds_at(rp(Id, Max)=true,  T),
9      Offer is Min + (Max - Min) * CA.
```

```
11 select(exit(ThreadId, Item), T):-
12       deadline(Deadline),
13       ourdeadline(StartTime),
14       Td is (StartTime + Deadline),
15       T > Td.

17 select(offer(ThreadId, Item, Offer), T):-
18       deadline(Deadline),
19       ourdeadline(StartTime),
20       Td is (StartTime + Deadline),
21       T < Td,
22       calc_next_offer(ThreadId, Offer, T).
```

Listing A.2: Prolog buyer agent.

```
TIMESTAMP:::FROM:::TO:::EVENT

1389744273493:::s_1:::buyer_1:::
offer(0, laptop12, 768.3)
1389744273509:::buyer_2:::s_0:::
offer(13, laptop12, 333.0)
1389744273509:::buyer_2:::s_1:::
offer(1, bananas, 332.0)
1389744273509:::buyer_2:::s_2:::
offer(7, laptop12, 333.0)
1389744273509:::buyer_2:::s_3:::
offer(3, laptop12, 335.0)
1389744273587:::s_3:::buyer_1:::
offer(2, laptop12, 640.5)
1389744273603:::s_3:::buyer_2:::
offer(3, laptop12, 640.5)
1389744273649:::s_0:::buyer_1:::
offer(12, laptop12, 495.0)
1389744273649:::s_0:::buyer_2:::
offer(13, laptop12, 776.5)
1389744273743:::buyer_1:::s_1:::
offer(0, laptop12, 462.0)
1389744273743:::buyer_1:::s_3:::
 offer(2, laptop12, 430.0)
```

Listing A.3: Narrative from a simulation run.

Appendix B

CONAN Implementation in Prolog

In this appendix we show some further details on how we implement the strategy CONAN. We first show all the fluents, actions, initial setting which we gave some examples in Section 4.3. Then we show how the fluent will be effected by the action, as in Section refrevise. Finally we explore the offer generation and action selection implementation (Section 4.4.2 and Section 4.4.1) in detail.

B.1 Initial Setting

We will use the predicates Initially to initialize some of the fluents in CONAN. The following are the fluents:

```
1 initially(number_competitors = 0).
2 initially(total_status = negotiating).
3 initially(all_opp_offer(ThreadId, Item) = []).
4 initially(my_last_offer(ThreadId, Item, 0) = 0).
5 initially(opp_last_offer(ThreadId, Item, 0) = 0).
6 initially(eagerness = 0.5).
7 initially(reserved_threshold = 1).
8 initially(max_number_active_negotiations = 100).
9 initially(max_number_competitors = 40).
```

```
10 initially(demandSupplyRatio = 1).
11 initially(min_demand_supply_ratio = 0.1).
12 initially(max_demand_supply_ratio = 10).
13 initially(window_size = 3).
14 initially_t(1, self_deadline = Td):-
15         holds_at(deadline = Deadline), 1),
16         holds_at(startTime = StartTime, 1),
17         SDeadline is StartTime+Deadline.
18 initially(turn_of(ThreadId) = buyer)).
```

Listing B.1: Initialize some of the fluents in CONAN

B.2 Effect Of The Actions

```
2  % Notification of a new seller from the market controller
3  initiates(perceived(notify_about_new_seller(ThreadId)), active =
       ThreadId, T):-
4          number_sellers(NoActiveSellers, T),
5          holds_at(max_number_active_negotiations = MAN, T),
6          NoActiveSellers < MAN.

8  initiates(perceived(notify_about_new_seller(ThreadId)), result(
       ThreadId) = onnegotiation, T):-
9          number_sellers(NoActiveSellers, T),
10         holds_at(max_number_active_negotiations = MAN, T),
11         NoActiveSellers < MAN.

13 initiates(perceived(notify_about_new_seller(ThreadId)), neg_price(
       ThreadId, Item) =  Price, T):-
14         holds_at(price(Item) = Price, T),
15         number_sellers(NoActiveSellers,  T),
16         holds_at(max_number_active_negotiations = MAN, T),
17         NoActiveSellers < MAN.
```

```
19 % Notification of the number of competitors from  the market
       controller
20 initiates(perceived(notify_about_change_in_number_of_competitors(
       NoCompetitors)), number_competitors = NoCompetitors, T).

22 % Notification of a demand/supply ratio from  the market controller
23 initiates(perceived(notify_about_change_in_demand_supply_ratio(
       DemandSupplyRatio)), demandSupplyRatio = DemandSupplyRatio, T).

25 % Negotiation acts are : offer, accept, exit, request-to-reserve,
       cancel
26 % accept(Item)
27 initiates(attempted(accept(ThreadId, Item)), result(ThreadId) =
       success, T).

29 initiates(perceived(accept(ThreadId, Item)), result(ThreadId) =
       seller_accept, T).

31 % exit
32 initiates(perceived(exit(ThreadId, Item)), result(ThreadId) = failed
       , T).

34 initiates(attempted(exit(ThreadId, Item)), result(ThreadId) = failed
       , T).

36 % Offer and counter offer
37 initiates(perceived(offer(ThreadId, Item, New_price)), turn_of(
       ThreadId) = buyer, T).

39 initiates(perceived(offer(ThreadId, Item, New_price)), neg_price(
       ThreadId, Item) =  New_price , T).

41 initiates(perceived(offer(ThreadId, Item, New_price)),
       opp_last_offer(ThreadId, Item, New_price) = T, T).
```

```
43 initiates(perceived(offer(ThreadId, Item, New_price)), all_opp_offer
       (ThreadId, Item) = All_price, T):-
44         holds_at(all_opp_offer(ThreadId, Item) = Old_All_price, T),
45         append([New_price], Old_All_price, All_price).

47 initiates(attempted(offer(ThreadId, Item, New_price)), turn_of(
       ThreadId) = seller, T).

49 initiates(attempted(offer(ThreadId, Item, New_price)), neg_price(
       ThreadId, Item) = New_price, T).

51 initiates(attempted(offer(ThreadId, Item, New_price)), my_last_offer
       (ThreadId, Item, New_price) = T, T).

53 % Exit-all
54 initiates(attempted(exit_all(Item)), total_status = finished, T).

56 % request_to_reserve
57 initiates(attempte(cancel(ThreadId, Item)),result(ThreadId) = hold,
       T):-
58         holds_at(active = ThreadId, T).

60 % cancel
61 initiates(perceived(cancel(ThreadId, Item)), result(ThreadId) =
       failed, T).

63 initiates(attempted(cancel(ThreadId, Item)), result(ThreadId) =
       failed, T).

65 initiates(perceived(cancel(ThreadId, Item)), result(get_penalty(
       ThreadId) = Penalty, T):-
66         holds_at(neg_price(ThreadId, Item) = Price, T),
67         Penalty is 20.

69 initiates(attempted(cancel(ThreadId Item)), result(pay_penalty(
       ThreadId) = Penalty, T):-
```

```
70          holds_at(neg_price(ThreadId, Item) = Price, T),
71          Penalty is 20.
```

Listing B.2: Effect Of The Actions.

B.3 Offer Generation

```
2  % To find the number of sellers at any given time
3  number_sellers(NoSellers, T):-
4          findall(ThreadId, active = ThreadId, ThreadIds),
5          !,
6          length(ThreadIds, NoSellers).
7  number_sellers(0,_).

9  % To find the number of sellers that are currently negotiating at
       any given time
10 number_onnegotiation_sellers(NoOnnegotiationSellers, T):-
11         findall(ThreadId, holds_at( result(ThreadId) = onnegotiation
               , T, ThreadIds),
12         !,
13         length(ThreadIds, NoOnnegotiationSellers).
14 number_onnegotiation_sellers(0,_).

16 %To find the number of reserved offers
17 reserved_offer(N, T):-
18         findall(ThreadId, holds_at(result(ThreadId) = hold, T),
               ThreadIds),
19         !,
20         length(ThreadIds, N).
21 reserved_offer(0,_).

23 %To find the number of accepted offers at a given time.
24 accepted_offer(N, T):-
25         findall(ThreadId, holds_at(result(ThreadId) = seller_accept,
               T), ThreadIds),
26         !,
```

```
27          length(ThreadIds, N).
28  accepted_offer(0,_).

30  %To find the id of minimum price of accepted offers
31  min_accepted(Min, T):-
32          findall(Accepted_price, (holds_at(neg_price(ThreadId, Item)
                =  Accepted_price, T), holds_at(result(ThreadId) =
                seller_accept, T)), List),
33          list_min(List, Min).

35  % To find the total penalty for all reserved offers
36  total_penalty(T_penalty, T):-
37          findall(Penalty, (holds_at(result(ThreadId) = hold, T),
                thread_penalty(T, Penalty)), List),
38          list_sum(List, T_penalty).

40  % To calculate each reserved offer penalty
41  thread_penalty(T, Penalty):-
42          holds_at(neg_price(ThreadId, Item) = P, T),
43          holds_at(result(ThreadId) = hold, T),
44          Penalty is P.

46  %To find the minimum price of reserved offers
47  min_reserved(Min, T):-
48          findall(Comited_price, (holds_at(neg_price(ThreadId, Item) =
                 Comited_price, T), holds_at(result(ThreadId) = hold, T))
                , List),
49          list_min(List, Min).

51  min_reserved_id(Min_id, T):-
52          findall(Comited_price, (holds_at(neg_price(ThreadId, Item) =
                 Comited_price, T), holds_at(result(ThreadId) = hold, T))
                , List),
53          list_min(List, Min),
54          holds_at(neg_price(ThreadId, Item) = Min, T),
55          holds_at(result(ThreadId) = hold, T).
```

```
57 % return the id of the dialogue with the maximum reserved offer
58 max_reserved_id(Max_id, T):-
59         findall(Comited_price, (holds_at(neg_price(ThreadId, Item) =
               Comited_price, T), holds_at(result(ThreadId) = hold, T))
              , List),
60         list_max(List, Max),
61         holds_at(neg_price(ThreadId, Item) = Max, T).

63 max_reserved(Max, T):-
64         findall(Comited_price, (holds_at(neg_price(ThreadId, Item) =
               Comited_price, T), holds_at(result(ThreadId) = hold, T))
              , List),
65         list_max(List, Max).

67 %To find the id of minimum price of accepted offers
68 min_accepted_id(Min_id, T):-
69         findall(Accepted_price, (holds_at(neg_price(ThreadId, Item)
              = Accepted_price, T), holds_at(result(ThreadId) =
              seller_accept, T)), List),
70         list_min(List, Min),
71         findall(Min_ids, holds_at(neg_price(Min_ids, Item) = Min, T)
              , List1),
72         element(1, List1, Min_id).

74 % if value in[N,0.66] then incompatible opponent
75 map_value_range(Value,Range,  RangePosition):-
76         Value >= 0.66,
77         Range = 'incompatible',
78         RangePosition is 3.

80 % if value in[0.66,0.33] then moderately compatible opponent
81 map_value_range(Value,Range, RangePosition):-
82         Value < 0.66,
83         Value >= 0.33,
84         Range = 'moderate compatible',
```

```
85          RangePosition is 2.

87  % if value in[0.33,0] then compatible opponent
88  map_value_range(Value,Range, RangePosition):-
89          Value < 0.33,
90          Range = 'compatible',
91          RangePosition is 1.

93  % negotiation situation in each thread
94  % C1- is the opponent response time, the faster the response time
        the better.
95  opponent_response_time(ThreadId, C1, RangePosition, T):-
96          holds_at(opp_last_offer(ThreadId, _,_) = OppLastT, T),
97          holds_at(deadline = Deadline, T),
98          holds_at(my_last_offer(ThreadId,_,_) = MyLastT, T),
99          C1Value is float(OppLastT-MyLastT)/float(Deadline),
100         map_value_range(C1Value, C1, RangePosition).

102 % C2 - is the opponent concession rate- if the opponent is
        irrational it will return incompatible opponent
103 opponent_concession_rate(ThreadId, C2, RangePosition, T):-
104         holds_at(window_size = WindowSize, T),
105         holds_at(all_opp_offer(ThreadId, Item) = All_price, T),
106         holds_at(ip = Min, T),
107         holds_at(rp = Max, T),
108         (
109                 NoOppOffers >= WindowSize ->
110                 element(1, All_price, First),
111                 element(2, All_price, Second),
112                 element(3, All_price, Third),
113                 X is (Second-First)+(Third-Second),
114                 C2Value is float(X)/float(Max-Min),
115                 map_value_range(1-0.25-C2Value, C2, RangePosition)
116                 ;
117                 NoOppOffers == 2 ->
118                 element(1, All_price, First),
```

```
                element(2, All_price, Second),
                X is (Second-First),
                C2Value is float(X)/float(Max-Min),
                map_value_range(1-0.25-C2Value, C2, RangePosition)
                ;
                NoOppOffers < 2 ->
                C2Value is 1,
                map_value_range(1-C2Value, C2, RangePosition),
        ).

% compatible  [2- 3]
map_threadValue_threadStatus(ThreadValue, ThreadStatus):-
        ThreadValue =< 3,
        ThreadValue >= 2,
        ThreadStatus ='compatible'.

% moderately compatible [4- 5]
map_threadValue_threadStatus(ThreadValue, ThreadStatus):-
        ThreadValue =< 5,
        ThreadValue >= 4,
        ThreadStatus ='moderate compatible'.

% incompatible [6- 6]
map_threadValue_threadStatus(ThreadValue, ThreadStatus):-
        ThreadValue =< 6,
        ThreadValue >= 6,
        ThreadStatus ='incompatible'.

thread_situation(ThreadId, ThreadValue, ThreadStatus, ThreadId, T):-
        opponent_response_time(ThreadId, C1, C1RangePosition, T),
        opponent_concession_rate(ThreadId, C2, C2RangePosition, T),
        ThreadValue is C1RangePosition + C2RangePosition,
        map_threadValue_threadStatus(ThreadValue, ThreadStatus).

% retrieve all the threads situation and calculate the concession
    rat for the negotiation situation
```

```
154 negotiationStatus(AllThreadsSituation, T):-
155         findall(ThreadSituation, (holds_at(result(ThreadId) =
                onnegotiation, T), thread_situation(ThreadId,
                ThreadSituation, ThreadStatus, T)), List),
156         list_sum(List, AllThreadsSituation).

158 negotiationStatusCR(NegotiationStatusCR, T):-
159         number_sellers(NoOnnegotiationSellers, T),
160         negotiationStatus(AllThreadsSituation, T),
161         NegotiationStatusCR is float(AllThreadsSituation - 2 *
                NoOnnegotiationSellers)/float((6 * NoOnnegotiationSellers
                ) - (2 * NoOnnegotiationSellers)).

163 selfFactors(ThreadId, Self ,  T):-
164    holds_at(self_deadline = Td, T),
165    holds_at(startTime = Ts, T),
166    DeadlineF is float(T-Ts)/float(Td-Ts),
167    reserved_offer(N, T),
168    ReservedF is (float(1)/float(N+1)),
169    holds_at(eagerness = EagernessF, T),
170    generalNegotiationStatus(NegotiationStatusF, T),
171    Self is 0.25*(DeadlineF + ReservedF + EagernessF +
           NegotiationStatusF).

173 envFactors(ThreadId, Env, T):-
174        number_sellers(NoSellers, T),
175        NoSellersF is float(1)/float(NoSellers),
176        holds_at(number_competitors = NoCompetitors, T),
177        holds_at(max_number_competitors = MaxNoCompetitors, T),
178        NoCompetitorsF is float(NoCompetitors)/float(MaxNoCompetitors
               ),
179        holds_at(demandSupplyRatio = DemandSupplyRatio, T),
180        holds_at(min_demand_supply_ratio = MinDemandSupplyRatio, T),
181        holds_at(max_demand_supply_ratio = MaxDemandSupplyRatio, T),
182        DemandSupplyRatioF is float(DemandSupplyRatio-
               MinDemandSupplyRatio)/
```

```
183         float(MaxDemandSupplyRatio-MinDemandSupplyRatio),
184         Env is 0.33 *(NoSellersF + NoCompetitorsF +
                DemandSupplyRatioF).

186 dM(DM, Y, T):-
187          holds_at(rp = Max, T),
188          holds_at(all_opp_offer(ThreadId, Item) = All_price, T),
189          length( All_price, NoOppOffers),
190          element(NoOppOffers, All_price, First),
191          holds_at(self_deadline = Td, T),
192          holds_at(startTime = Ts, T),
193          DeadlineF is float(CurrentTime-Ts)/float(Td-Ts),
194          Y is float(First)/float(Max),
195          X is Y * DeadlineF,
196          (
197                  X > 1 ->
198                  DM is 1
199                  ;
200                  DM is X
201          ).

203 % Self factor weight, if Self= Low (0.33-0] then
204 w_Self(ThreadId, W_Self, Self, DM, T):-
205          Self < 0.33,
206          Self >= 0,
207          W_Self is DM * 0.75.

209 % Self factor weight, if Self= Medium (0.66-0.33] then
210 w_Self(ThreadId, W_Self, Self, DM, T):-
211          Self < 0.66,
212          Self >= 0.33,
213          W_Self is DM * 0.5.

215 % Self factor weight, if Self= High [1-0.66] then
216 w_Self(ThreadId, W_Self, Self, DM, T):-
217          Self =< 1,
```

```
218         Self >= 0.66,
219         W_Self is DM * 0.25.

221 % Environment factor weight, in any case then
222 w_Env(ThreadId, W_Env, W_Self, W_Opp, T):-
223         W_Env is  1-W_Self.

225 % concession rate
226 concession(CA, ThreadId, T):-
227         findall(Cycle_Times, neardeadline(Cycle_Times), List),
228         dM(DM, Dis, T),
229         list_max(List, Near_Deadline),
230         holds_at(self_deadline = Td, T),
231         (
232                 T >= (Td - (Near_Deadline)) ->
233                 CA is 0.99
234                 ;
235                 selfFactors(ThreadId, Self, T),
236                 w_Self(ThreadId, W_Self, Self, T),
237                 envFactors(ThreadId, Env, T),
238                 w_Env(ThreadId, W_Env, W_Self, T),
239                 CA is (W_Env * Env + W_Self * Self)
240         ).

242 select_offer(MyLastOffer, IntendedOffer, Offer):-
243        MyLastOffer >= IntendedOffer,
244         Offer is MyLastOffer.
245 select_offer(MyLastOffer, IntendedOffer, Offer):-
246         MyLastOffer < IntendedOffer,
247         Offer is IntendedOffer.

249 holds_at(intended_offer(ThreadId) = Offer, T):-
250         concession(CA, ThreadId, T),
251         holds_at(ip = Min, T),
252         holds_at(rp = Max, T),
253         Offer is Min + (Max - Min) * CA.
```

```
255 % calculate an offer
256 calc_next_offer(ThreadId, Offer, T):-
257         holds_at(intended_offer(ThreadId) =  IntendedOffer, T),
258         holds_at(my_last_offer(ThreadId, _, MyLastOffer) = T1, T),
259         select_offer(MyLastOffer, IntendedOffer, Offer).

261 next_offer(ThreadId, Offer, T):-
262         computed_next_offer(ThreadId, Offer, T),
263         !.
264 next_offer(ThreadId, Offer, T):-
265         calc_next_offer(ThreadId, Offer, T),
266         assert(computed_next_offer(ThreadId, Offer, T)).
```

Listing B.3: Offer Generation.

B.4 Action Selection

```
1  select(exit(ThreadId, Item), T):-
2         holds_at(neg_price(ThreadId, Item) = Opponent_offer, T),
3         holds_at(self_deadline = Td, T),
4         T >= Td,
5         holds_at(result(ThreadId) = onnegotiation, T),
6         holds_at(item = Item, T).

8  select(exit(ThreadId, Item), T):-
9          holds_at(neg_price(ThreadId, Item) = Opponent_offer, T),
10         Opponent_offer =\= 0,
11         holds_at(self_deadline = Td, T),
12         T < Td,
13         holds_at(item = Item, T),
14         number_sellers(NoActiveSellers, T),
15         holds_at(max_number_active_negotiations = MAN, T),
16         NoActiveSellers >= MAN,
17         thread_situation(ThreadValue, ThreadStatus, T),
18         ThreadStatus == 'incompatible'.
```

```
20 select(exit(ThreadId, Item), T):-
21         holds_at(result(ThreadId) = seller_accept, T),
22         holds_at(self_deadline = Td, T),
23         T < Td,
24         holds_at(item = Item, T),
25         Opponent_offer =\= 0,
26         reserved_offer(NoReservedOffers, T),
27         NoReservedOffers =\= 0,
28         holds_at(reserved_threshold = ReservedThreshold, T),
29         NoReservedOffers =< ReservedThreshold,
30         total_penalty(T_panalty, T),
31         min_reserved(Min, T),
32         (Opponent_offer + T_panalty) > Min.

34 select(exit(ThreadId, Item), T):-
35         holds_at(result(ThreadId) = onnegotiation, T),
36         findall(ThreadId, holds_at(result(ThreadId) = success, T),
               ThreadIds),
37         length(ThreadIds, NoSuccess),
38         NoSuccess == 1,
39         holds_at(self_deadline = Td, T),
40         T < Td,
41         holds_at(item = Item, T).

43 select(request_to_reserve(ThreadId, Item), T):-
44         holds_at(result(ThreadId) = onnegotiation, T),
45         holds_at(self_deadline = Td, T),
46         holds_at(near_deadline = NearDeadline, T),
47         T < Td,
48         holds_at(turn_of(ThreadId) = buyer, T),
49         holds_at(neg_price(ThreadId, Item) = Opponent_offer, T),
50         holds_at(item = Item, T),
51         Opponent_offer =\= 0,
52         next_offer(T, Offer, CounterIntendedOffers),
53         reserved_offer(NoReservedOffers, T),
54         NoReservedOffers == 0,
```

```
55          Opponent_offer =< Offer,
56          findall(Offerd_price, (holds_at(opp_last_offer(ThreadIds, _,
                Offerd_price) = _,  T), holds_at(result(ThreadIds) =
               onnegotiation,  T)), List),
57          list_min(List, Min),
58          findall(Min_ids, holds_at(neg_price(Min_ids, Item) = Min, T)
               , List1),
59          element(1, List1, Min_id),
60          Min_id == ThreadId.

62  select(request_to_reserve(ThreadId, Item), T):-
63          holds_at(result(ThreadId) = onnegotiation, T),
64          holds_at(self_deadline = Td, T),
65          T < Td,
66          holds_at(turn_of(ThreadId) = buyer, T),
67          holds_at(neg_price(ThreadId, Item) = Opponent_offer, T),
68          holds_at(item = Item, T),
69          Opponent_offer =\= 0,
70          next_offer(T, Offer, CounterIntendedOffers),
71          reserved_offer(NoReservedOffers, T),
72          NoReservedOffers =\= 0,
73          holds_at(reserved_threshold = ReservedThreshold, T),
74          NoReservedOffers =< ReservedThreshold,
75          total_penalty(T_panalty, T),
76          min_reserved(Min, T),
77          (Opponent_offer + T_panalty) =< Offer,
78          (Opponent_offer + T_panalty) < Min,
79          findall(Offerd_price, (holds_at(opp_last_offer(ThreadId, _,
               Offerd_price) = _,  T), holds_at(result(ThreadId) =
               onnegotiation, T)), List),
80          list_min(List, Min),
81          findall(Min_ids, holds_at(neg_price(Min_ids, Item) =  Min, T
               ), List1),
82          element(1, List1, Min_id),
83          Min_id == ThreadId.
```

```
85  select(request_to_reserve(ThreadId, Item), T):-
86          holds_at(result(ThreadId) = seller_accept, T),
87          holds_at(self_deadline = Td, T),
88          T < Td,
89          holds_at(neg_price(ThreadId, Item) = Opponent_offer, T),
90          holds_at(item = Item, T),
91          Opponent_offer =\= 0,
92          reserved_offer(NoReservedOffers, T),
93          NoReservedOffers == 0,
94          accepted_offer(NoAcceptedOffers, T),
95          NoAcceptedOffers == 0,
96          findall(Offerd_price, (holds_at(neg_price(ThreadId, Item) =
                Offerd_price, T), holds_at(result(ThreadId) =
                onnegotiation, T)), List),
97          list_min(List, Min),
98          findall(Min_ids, holds_at(neg_price(Min_ids, Item) =  Min, T
                ), List1),
99          element(1, List1, Min_id),
100         Min_id == ThreadId.

102 select(request_to_reserve(ThreadId, Item), T):-
103         holds_at(result(ThreadId) = seller_accept, T),
104         holds_at(self_deadline = Td, T),
105         T < Td,
106         holds_at(neg_price(ThreadId, Item) = Opponent_offer, T),
107         holds_at(item = Item, T),
108         Opponent_offer =\= 0,
109         reserved_offer(NoReservedOffers, T),
110         NoReservedOffers == 0,
111         accepted_offer(NoAcceptedOffers, T),
112         NoAcceptedOffers =\= 0,
113         min_accepted_id(Min_id, T),
114         Min_id == ThreadId.

116 select(request_to_reserve(ThreadId, Item), T):-
117         holds_at(self_deadline = Td, T),
```

```
118         T < Td,
119         holds_at(result(ThreadId) = seller_accept, T),
120         get_time(TimeStart2),
121         holds_at(neg_price(ThreadId, Item) = Opponent_offer, T),
122         holds_at(item = Item, T),
123         Opponent_offer =\= 0,
124         reserved_offer(NoReservedOffers, T),
125         NoReservedOffers =\= 0,
126         holds_at(reserved_threshold = ReservedThreshold, T),
127         NoReservedOffers =< ReservedThreshold,
128         total_penalty(T_panalty, T),
129         min_reserved(Min, T),
130         (Opponent_offer + T_panalty) < Min.

132 select(cancel(ThreadId, Item), T):-
133         holds_at(self_deadline = Td, T),
134         T < Td,
135         holds_at(result(ThreadId) = hold, T),
136         holds_at(item = Item, T),
137         reserved_offer(NoReservedOffers, T),
138         NoReservedOffers =\= 0,
139         holds_at(reserved_threshold = ReservedThreshold, T),
140         NoReservedOffers > ReservedThreshold,
141         max_reserved_id(Max_id, T),
142         Max_id == ThreadId.

144 select(accept(ThreadId, Item), T):-
145         holds_at(self_deadline = Td, T),
146         T > Td,
147         holds_at(result(ThreadId) = hold, T),
148         holds_at(item = Item, T),
149         reserved_offer(NoReservedOffers, T),
150         NoReservedOffers =\= 0,
151         min_reserved_id(Min_id, T),
152         Min_id == ThreadId.
```

```
154 select(accept(ThreadId, Item), T):-
155         holds_at(self_deadline = Td, T),
156         T > Td,
157         holds_at(item = Item, T),
158         reserved_offer(NoReservedOffers, T),
159         NoReservedOffers == 0,
160         findall(Offerd_price, (holds_at(opp_last_offer(ThreadId, _,
                Offerd_price) = _, T), holds_at(result(ThreadId) =
                onnegotiation, T)), List),
161         list_min(List, Min),
162         findall(Min_ids, holds_at(opp_last_offer(ThreadId, _, Min) =
                 _,  T), List1),
163         element(1, List1, Min_id),
164         Min_id == ThreadId,
165         holds_at(rp = Max, T),
166         Min < Max.

168 select(accept(ThreadId, Item), T):-
169         holds_at(result(ThreadId) = seller_accept, T),
170         holds_at(self_deadline = Td, T),
171         T > Td,
172         holds_at(neg_price(ThreadId, Item) = Opponent_offer, T),
173         holds_at(item = Item, T),
174         Opponent_offer =\= 0,
175         reserved_offer(NoReservedOffers, T),
176         NoReservedOffers == 0,
177         accepted_offer(NoAcceptedOffers, T),
178         NoAcceptedOffers == 0,
179         findall(Offerd_price,(holds_at(neg_price(ThreadIds, Item) =
                Offerd_price, T), holds_at(result(ThreadIds) =
                onnegotiation, T)), List),
180         list_min(List, Min),
181         findall(Min_ids, holds_at(neg_price(Min_ids, Item) =  Min, T
                ), List1),
182         element(1, List1, Min_id),
183         Min_id == ThreadId.
```

```
185 select(accept(ThreadId, Item), T):-
186         holds_at(result(ThreadId) = seller_accept, T),
187         holds_at(self_deadline = Td, T),
188         T > Td,
189         holds_at(neg_price(ThreadId, Item) = Opponent_offer, T),
190         holds_at(item = Item, T),
191         Opponent_offer =\= 0,
192         reserved_offer(NoReservedOffers, T),
193         NoReservedOffers == 0,
194         accepted_offer(NoAcceptedOffers, T),
195         NoAcceptedOffers =\= 0,
196         min_accepted_id(Min_id, T),
197         Min_id == ThreadId.

199 select(accept(ThreadId, Item), T):-
200         holds_at(self_deadline = Td, T),
201         T > Td,
202         holds_at(result(ThreadId) = seller_accept, T),
203         holds_at(neg_price(ThreadId, Item) = Opponent_offer, T),
204         holds_at(item = Item, T),
205         Opponent_offer =\= 0,
206         reserved_offer(NoReservedOffers, T),
207         NoReservedOffers =\= 0,
208         holds_at(reserved_threshold = ReservedThreshold, T),
209         NoReservedOffers =< ReservedThreshold,
210         total_penalty(T_panalty, T),
211         min_reserved(Min, T),
212         (Opponent_offer + T_panalty) < Min.

214 select(cancel(ThreadId, Item), T):-
215         holds_at(self_deadline = Td, T),
216         T > Td,
217         holds_at(result(ThreadId) = hold, T),
218         holds_at(item = Item, T),
219         reserved_offer(NoReservedOffers, T),
```

```
220         NoReservedOffers =\= 0,
221         min_reserved_id(Min_id, T),
222         Min_id \= ThreadId.

224 select(exit_all(Item), T):-
225         holds_at(total_status = negotiating, T),
226         holds_at(self_deadline = Td, T),
227         T > Td,
228         holds_at(item = Item, T).

230 select(offer(ThreadId, Item, Offer),T):-
231         holds_at(result(ThreadId) = onnegotiation, T),
232         holds_at(self_deadline = Td, T),
233         T < Td,
234         holds_at(neg_price(ThreadId, Item) = Opponent_offer, T),
235         Opponent_offer == 0,
236         holds_at(turn_of(ThreadId) = buyer, T),
237         holds_at(item = Item, T),
238         holds_at(ip = Offer, T).

240 select(offer(ThreadId, Item, Offer),T):-
241         holds_at(result(ThreadId) = onnegotiation, T),
242         holds_at(self_deadline = Td, T),
243         T < Td,
244         holds_at(turn_of(ThreadId) = buyer, T),
245         holds_at(neg_price(ThreadId, Item) = Opponent_offer, T),
246         holds_at(item = Item, T),
247         Opponent_offer =\= 0,
248         next_offer(T, Offer, CounterIntendedOffers),
249         reserved_offer(NoReservedOffers, T),
250         NoReservedOffers =\= 0,
251         total_penalty(T_penalty, T),
252         min_reserved(Min, T),
253         Offer < Opponent_offer,
254         (Offer + T_penalty) < Min,
255         get_time(TimeEnd).
```

```
select(offer(ThreadId, Item, Offer),T):-
        holds_at(result(ThreadId) = onnegotiation, T),
        holds_at(self_deadline = Td, T),
        T < Td,
        holds_at(turn_of(ThreadId) = buyer, T),
        holds_at(neg_price(ThreadId, Item) =  Opponent_offer, T),
        holds_at(item = Item, T),
        Opponent_offer =\= 0,
        next_offer(T, Offer, CounterIntendedOffers),
        reserved_offer(NoReservedOffers, T),
        NoReservedOffers =\= 0,
        total_penalty(T_panalty, T),
        min_reserved(Min, T),
        Offer < Opponent_offer,
        (Offer + T_panalty) >= Min,
        holds_at(startTime = Ts, T),
        DeadlineF is float(CurrentTime-Ts)/float(Td-Ts),
        Less_Offer is Min-(T_panalty+(Min* 0.1*(1-DeadlineF))).

select(offer(ThreadId, Item, Offer),T):-
        holds_at(result(ThreadId) = onnegotiation, T),
        holds_at(self_deadline = Td, T),
        T < Td,
        holds_at(turn_of(ThreadId) = buyer, T),
        holds_at(neg_price(ThreadId, Item) =  Opponent_offer, T),
        Opponent_offer =\= 0,
        holds_at(item = Item, T),
        next_offer(T, Offer, CounterIntendedOffers),
        reserved_offer(NoReservedOffers, T),
        NoReservedOffers == 0,
        Offer < Opponent_offer.

calculate_negotiationStatusCR(T):-
        holds_at(startTime = ST, T),
        T =\= ST,
```

```
292         negotiationStatusCR(NegotiationStatusF, T),
293         assert(generalNegotiationStatus(NegotiationStatusF, T)).
```

Listing B.4: Action Selection.

Bibliography

[1] B. An, N. Gatti, and V. Lesser. Extending alternating-offers bargaining in one-to-many and many-to-many settings. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '09, pages 423–426, Washington, DC, USA, 2009. IEEE Computer Society.

[2] B. An, N. Gatti, and V. Lesser. Bilateral bargaining with one-sided uncertain reserve prices. *Autonomous Agents and Multi-Agent Systems*, 26(3):420–455, 2013.

[3] B. An, V. Lesser, D. Irwin, and M. Zink. Automated negotiation with decommitment for dynamic resource allocation in cloud computing. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '10, pages 981–988, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[4] B. An, V. Lesser, and K. M. Sim. Decommitment in multi-resource negotiation. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 3*, AAMAS '08, pages 1553–1556, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[5] B. An, V. Lesser, and K. M. Sim. Strategic agents for multi-resource negotiation. *Autonomous Agents and Multi-Agent Systems*, 23(1):114–153, July 2011.

[6] B. An, K. M. Sim, L. G. Tang, S. Q. Li, and D. J. Cheng. Continuous-time negotiation mechanism for software agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 36(6):1261 –1272, 2006.

[7] A. Artikis, F. Guerin, and J. Pitt. Integrating interaction protocols and internet protocols for agent-mediated e-commerce. In F. Dignum and U. Cortes, editors, *Agent-Mediated Electronic Commerce III*, volume 2003 of *Lecture Notes in Computer Science*, pages 47–69. Springer Berlin Heidelberg, 2001.

[8] A. Artikis and M. Sergot. Executable specification of open multi-agent systems. *Logic Journal of the IGPL*, 18:31–65, 2010.

[9] A. Artikis, M. Sergot, and J. Pitt. Specifying norm-governed computational societies. *ACM Transactions on Computational Logic*, 10(1):1–42, 2009.

[10] R. Ashri, I. Rahwan, and M. Luck. Architectures for negotiating agents. In *Proceedings of the 3rd Central and Eastern European conference on Multi-agent systems*, CEEMAS'03, pages 136–146, Berlin, Heidelberg, 2003. Springer-Verlag.

[11] T. Baarslag, A. Dirkzwager, K. Hindriks, and C. M. Jonker. The significance of bidding, accepting and opponent modeling in automated negotiation. In *21st European Conference on Artificial Intelligence*, Frontiers in Artificial Intelligence and Applications, pages 27–32. ECAI2014: 21st European Conference on Artificial Intelligence, 2014.

[12] T. Baarslag, M. Hendrikx, K. V. Hindriks, and C. M. Jonker. Measuring the performance of online opponent models in automated bilateral negotiation. In M. Thielscher and D. Zhang, editors, *AI 2012: Advances in Artificial Intelligence*, volume 7691 of *Lecture Notes in Computer Science*, page 1–14. Springer, Springer, 2012.

[13] J. Y. Bakos. A strategic analysis of electronic marketplaces. *MIS Quarterly*, 15(3):pp. 295–310, 1991.

[14] A. Bikakis and G. Antoniou. Defeasible contextual reasoning with arguments in ambient intelligence. *Knowledge and Data Engineering, IEEE Transactions on*, 22(11):1492–1506, Nov 2010.

[15] L. Bodenstaff. Formalisation of argumentation protocols in event calculus. Master's thesis, Utrecht University, 2005.

[16] D. Bouyssou, T. Marchant, and P. Perny. *Social Choice Theory and Multicriteria Decision Aiding*, pages 779–810. ISTE, 2010.

[17] D. Bouyssou, T. Marchant, M. Pirlot, A. Tsoukiàs, and P. Vincke. *Evaluation And Decision Models With Multiple Criteria. Stepping Stones For The Analyst*. International Series in Operations Research & Management Science, Vol. 86. Springer, 2006.

[18] S. Bromuri and K. Stathis. Situating cognitive agents in GOLEM. In D. Weyns, S. Brueckner, and Y. Demazeau, editors, *Engineering Environment-Mediated Multi-Agent Systems*, volume 5049 of *Lecture Notes in Computer Science*, pages 115–134. Springer Berlin Heidelberg, 2008.

[19] S. Bromuri and K. Stathis. Distributed agent environments in the ambient event calculus. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 12:1–12:12, New York, NY, USA, 2009. ACM.

[20] S. Bromuri, V. Urovi, M. Morge, K. Stathis, and F. Toni. A multi-agent system for service discovery, selection and negotiation. In *AAMAS*, pages 1395–1396, 2009.

[21] S. Bromuri, V. Urovi, and K. Stathis. iCampus: A Connected Campus in the Ambient Event Calculus. *International Journal of Ambient Computing and Intelligence (IJACI)*, 2(1):59–65, 2010.

[22] J. Brzostowski and R. Kowalczyk. Predicting partner's behaviour in agent negotiation. In *Proceedings of the fifth international joint conference on*

Autonomous agents and multiagent systems, AAMAS '06, pages 355–361, New York, NY, USA, 2006. ACM.

[23] M. Cao, X. Luo, X. R. Luo, and X. Dai. Automated negotiation for e-commerce decision making: A goal deliberated agent architecture for multi-strategy selection. *Decision Support Systems*, 73(0):1–14, 2015.

[24] A. Chopra and M. Singh. Producing compliant interactions: Conformance, coverage, and interoperability. In M. Baldoni and U. Endriss, editors, *Declarative Agent Languages and Technologies IV*, volume 4327 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2006.

[25] N. Dipsis and K. Stathis. Ubiquitous Agents for Ambient Ecologies. *Pervasive and Mobile Computing*, 8(4):562–574, 2012.

[26] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, New York, NY, USA, 2010.

[27] A. Fabregues and C. Sierra. An agent architecture for simultaneous bilateral negotiations. In *Proceedings of the 13th International Conference of the Catalan Association for Artificial Intelligence (CCIA 2010)*, pages 29–38, Espluga de Francolí, Tarragona, 2010.

[28] P. Faratin. *Automated service negotiation between autonomous computational agents*. PhD thesis, Queen Mary & Westfield College, London, 2000.

[29] P. Faratin, C. Sierra, and N. R. Jennings. Negotiation decision functions for autonomous agents. *International Journal of Robotics and Autonomous Systems*, 24(3 - 4):159–182, 1998.

[30] M. Fasli. *Agent Technology For E-Commerce*. John Wiley & Sons, 2007.

[31] S. Fatima, S. Kraus, and M. Wooldridge. *Principles of Automated Negotiation*. Cambridge University Press, 2014.

[32] S. S. Fatima and M. Wooldridge. Multilateral bargaining for resource division. In *21st European Conference on Artificial Intelligence ECAI 2014*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 309–314. IOS Press, 2014.

[33] S. S. Fatima, M. Wooldridge, and N. R. Jennings. Multi-issue negotiation with deadlines. *Journal of Artificial Intelligence Research*, 27:381–417, November 2006.

[34] M. Fisher, R. H. Bordini, B. Hirsch, and P. Torroni. Computational logics and agents: A road map of current technologies and future trends. *Computational Intelligence*, 23(1):61–91, 2007.

[35] R. Fisher and W. L. Ury. *Getting to Yes: Negotiating Agreement Without Giving In.* Penguin (Non-Classics), 2nd edition, 1991.

[36] J. Forth, K. Stathis, and F. Toni. Decision making with a KGP agent systems. *Journal of Decision Systems*, 15(2-3):241–266, 2006.

[37] K. Fujita, T. Ito, T. Baarslag, K. Hindriks, C. Jonker, S. Kraus, and R. Lin. The second automated negotiating agents competition (anac2011). In T. Ito, M. Zhang, V. Robu, and T. Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 183–197. Springer Berlin Heidelberg, 2013.

[38] J. García-Lapresta, M. Martínez-Panero, and L. Meneses. Defining the Borda count in a linguistic decision making context. *Information Sciences*, 179(14):2309–2316, 2009. Including Special Section Linguistic Decision Making Tools and Applications.

[39] F. Guerin and J. Pitt. Guaranteeing properties for e-commerce systems. In J. Padget, O. Shehory, D. Parkes, N. Sadeh, and W. Walsh, editors,

Agent-Mediated Electronic Commerce IV. Designing Mechanisms and Systems, volume 2531 of *Lecture Notes in Computer Science*, pages 253–272. Springer Berlin Heidelberg, 2002.

[40] F. Guerin and J. Pitt. Verification and compliance testing. In M.-P. Huget, editor, *Communication in Multiagent Systems*, volume 2650 of *Lecture Notes in Computer Science*, pages 98–112. Springer Berlin Heidelberg, 2003.

[41] R. H. Guttman and P. Maes. Cooperative vs. competitive multi-agent negotiations in retail electronic commerce. In *Proceedings of the Second International Workshop on Cooperative Information Agents II, Learning, Mobility and Electronic Commerce for Information Discovery on the Internet*, pages 135–147, London, UK, 1998. Springer-Verlag.

[42] R. Hadfi and T. Ito. Addressing complexity in multi-issue negotiation via utility hypergraphs. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[43] M. He, H. Leung, and N. R. Jennings. A fuzzy logic based bidding strategy for autonomous agents in continuous double auctions. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1345–1363, 2003.

[44] J. Hernández, J. Mula, R. Poler, and A. Lyons. Collaborative planning in multi-tier supply chains supported by a negotiation-based mechanism and multi-agent system. *Group Decision and Negotiation*, 23(2):235–269, 2014.

[45] K. Hindriks, C. M. Jonker, and D. Tykhonov. The benefits of opponent models in negotiation. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '09, pages 439–444, Washington, DC, USA, 2009. IEEE Computer Society.

[46] K. Hindriks and D. Tykhonov. Opponent modelling in automated multi-issue negotiation using bayesian learning. In *Proceedings of the 7th international*

joint conference on Autonomous agents and multiagent systems - Volume 1, AAMAS '08, pages 331–338, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[47] G. Jagabathuni. Analytical techniques for dynamic negotiation runs in recon. Master's thesis, Department of Computer Science, Royal Holloway, Univ. of London, 2015.

[48] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: Prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 10(2):199–215, 2001.

[49] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1:7–38, 1998.

[50] Ö. Kafalı, S. Bromuri, E. Aguilar-Pelaez, M. Sindlar, T. van der Weide, M. Schumacher, E. Rodriguez-Villegas, and K. Stathis. A Smart e-Health Environment for Diabetes Management. *Journal of Ambient Intelligence and Smart Environments*, 5(5):479–502, 2013.

[51] A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. Declarative agent control. In J. Leite and P. Torroni, editors, *Computational Logic in Multi-Agent Systems*, volume 3487 of *Lecture Notes in Computer Science*, pages 96–110. Springer Berlin Heidelberg, 2005.

[52] K. Kolomvatsos and S. Hadjieftymiades. On the use of particle swarm optimization and kernel density estimator in concurrent negotiations. *Information Sciences*, 262(0):99–116, 2014.

[53] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, January 1986.

[54] S. Kraus. Automated negotiation and decision making in multiagent environments. In *In: MultiAgent Systems and Applications. ACAI-EASSS 2001 Proceedings, Luck M., Marik V., Stepankova O., Trappl R. (eds). Springer-Verlag*, pages 150–172. Springer-Verlag, 2001.

[55] S. Kraus. *Strategic Negotiation in Multiagent Environments*. MIT Press, Cambridge, MA, USA, 2001.

[56] P. Kruchten. The 4+1 view model of architecture. *Software, IEEE*, 12(6):42–50, Nov. 1995.

[57] G. Lai, C. Li, K. Sycara, and J. Giampapa. Literature review on multi-attribute negotiations. Technical report, Robotics Institute, Carnegie Mellon University, 2004.

[58] R. Y. K. Lau, M. Tang, O. Wong, S. W. Milliner, and Y.-P. P. Chen. An evolutionary learning approach for adaptive negotiation agents: Research articles. *International Journal of Intelligent Systems*, 21:41–72, January 2006.

[59] C. Li, J. A. Giampapa, and K. Sycara. Bilateral negotiation decisions with uncertain dynamic outside options. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Special Issue on Game-theoretic Analysis and Stochastic Simulation of Negotiation Agents*, 36(1):31–44, January 2006.

[60] R. Lin and S. Kraus. Can automated agents proficiently negotiate with humans? *Communications of the ACM*, 53(1):78–88, Jan. 2010.

[61] R. Lin, S. Kraus, T. Baarslag, D. Tykhonov, K. V. Hindriks, and C. M. Jonker. GENIUS: An integrated environment for supporting the design of generic automated negotiators. *Computational Intelligence*, 30(1):48–70, 2014.

[62] R. Lin, S. Kraus, J. Wilkenfeld, and J. Barry. Negotiating with bounded rational agents in environments with incomplete information using an automated agent. *Artificial Intelligence*, 172:823–851, April 2008.

[63] A. Lomuscio, M. Wooldridge, and N. Jennings. A classification scheme for negotiation in electronic commerce. *Group Decision and Negotiation*, 12(1):31–56, 2003.

[64] F. Lopes and H. Coelho. Strategic and tactical behaviour in automated negotiation. *International Journal of Artificial Intelligence*, 4 (Spring):35–63, 2010.

[65] F. Lopes, N. Mamede, A. Q. Novais, and H. Coelho. Negotiation strategies for autonomous computational agents. In *16th European conference on artificial intelligence (ECAI-04)*, pages 38–42. IOS Press, 2004.

[66] F. Lopes, M. Wooldridge, and A. Q. Novais. Negotiation among autonomous computational agents: principles, analysis and challenges. *Artificial Intelligence Review*, 29(1):1–44, 2008.

[67] X. Luo, C. Miao, N. R. Jennings, M. He, Z. Shen, and M. Zhang. KEMNAD: A knowledge engineering methodology for negotiating agent development. *Computational Intelligence*, 28(1):51–105, 2012.

[68] K. Mansour and R. Kowalczyk. An approach to one-to-many concurrent negotiation. *Group Decision and Negotiation*, 24(1):45–66, 2015.

[69] N. Matos, C. Sierra, and N. Jennings. Determining successful negotiation strategies: An evolutionary approach. In *Proceedings of the 3rd International Conference on Multi Agent Systems*, ICMAS '98, pages 182–189, Washington, DC, USA, 1998. IEEE Computer Society.

[70] P. McBurney, R. M. Van Eijk, S. Parsons, and L. Amgoud. A dialogue game protocol for agent purchase negotiations. *Autonomous Agents and Multi-Agent Systems*, 7:235–273, November 2003.

[71] J. McGinnis, K. Stathis, and F. Toni. A formal framework of virtual organisations as agent societies. In *Formal Aspects of Virtual Organisations*, pages 1–14, 2009.

[72] M. Morge, J. McGinnis, S. Bromuri, F. Toni, P. Mancarella, and K. Stathis. Toward a modular architecture of argumentative agents to compose services. In *Proceedings of EUMAS*, 2007.

[73] A. Munim. GOLEMLite: a framework for the development of agent-based applications. Master's thesis, Department of Computer Science, Royal Holloway, Univ. of London, September 2013.

[74] S. Munroe and M. Luck. Balancing conflict and cost in the selection of negotiation opponents. In *Rational, Robust, and Secure Negotiation Mechanisms in Multi-Agent Systems, 2005*, pages 39–53, 2005.

[75] V. Narayanan and N. R. Jennings. An adaptive bilateral negotiation model for e-commerce settings. In *Proceedings of the Seventh IEEE International Conference on E-Commerce Technology*, pages 34–41, Washington, DC, USA, 2005. IEEE Computer Society.

[76] B. Neville and J. Pitt. A computational framework for social agents in agent mediated e-commerce. In A. Omicini, P. Petta, and J. Pitt, editors, *Engineering Societies in the Agents World IV*, volume 3071 of *Lecture Notes in Computer Science*, pages 376–391. Springer Berlin Heidelberg, 2004.

[77] B. Neville and J. Pitt. Presage: A programming environment for the simulation of agent societies. In K. Hindriks, A. Pokahr, and S. Sardina, editors, *Programming Multi-Agent Systems*, volume 5442 of *Lecture Notes in Computer Science*, pages 88–103. Springer Berlin Heidelberg, 2009.

[78] T. Nguyen and N. R. Jennings. A heuristic model of concurrent bi-lateral negotiations in incomplete information settings. In *International Joint Conferences on Artificial Intelligence*, pages 1467–1469, 2003.

[79] T. Nguyen and N. R. Jennings. Coordinating multiple concurrent negotiations. In *3rd International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1064–1071, 2004.

[80] T. Nguyen and N. R. Jennings. Managing commitments in multiple concurrent negotiations. *International Journal Electronic Commerce Research and Applications*, 4:362–376, 2005.

[81] M. J. Osborne and A. Rubinstein. *A Course in Game Theory.* MIT Press, 1994.

[82] J. Pitt. *Electronic Agents in Future E-Commerce Scenarios: Some Computational, Social & Legal Aspects*, pages 23–49. Oslo University Press, 2003.

[83] I. Ponka. *Commitment models and concurrent bilateral negotiation strategies in dynamic service markets.* PhD thesis, University of Southampton, School of Electronics and Computer Science, 2009.

[84] I. Rahwan, R. Kowalczyk, and H. H. Pham. Intelligent agents for automated one-to-many e-commerce negotiation. In *ACSC '02: Proceedings of the twenty-fifth Australasian conference on Computer science*, pages 197–204, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.

[85] I. Rahwan, P. McBurney, and L. Sonenberg. Towards a theory of negotiation strategy (a preliminary report). In *Proceedings of the 5th Workshop on Game Theoretic and Decision Theoretic Agents (GTDT-2003)*, pages 73–80, Melbourne, Australia, 2003.

[86] I. Rahwan, S. D. Ramchurn, N. R. Jennings, P. McBurney, S. Parsons, and L. Sonenberg. Argumentation-based negotiation. *Knowledge Engineering Review*, 18(4):343–375, 2003.

[87] I. Rahwan, L. Sonenberg, N. R. Jennings, and P. McBurney. STRATUM: A methodology for designing heuristic agent negotiation strategies. *Applied Artificial Intelligence*, 21:489–527, June 2007.

[88] H. Raiffa. *The Art and Science of Negotiation*. Harvard University Press, 1982.

[89] D. M. Reeves, M. P. Wellman, and B. N. Grosof. Automated negotiation from declarative contract descriptions. In *Proceedings of the Fifth International Conference on Autonomous Agents*, AGENTS '01, pages 51–58, New York, NY, USA, 2001. ACM.

[90] F. Ren and M. Zhang. Prediction of partners' behaviors in agent negotiation under open and dynamic environments. In *Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops*, WI-IATW '07, pages 379–382. IEEE Computer Society, Washington, DC, USA, 2007.

[91] F. Ren, M. Zhang, and K. M. Sim. Adaptive conceding strategies for automated trading agents in dynamic, open markets. *Decision Support Systems*, 46(3):704–716, 2009.

[92] M. Resinas, P. Fernández, and R. Corchuelo. A bargaining-specific architecture for supporting automated service agreement negotiation systems. *Science of Computer Programming*, 77(1):4–28, Jan. 2012.

[93] V. Robu, D. J. A. Somefun, and J. A. L. Poutre. Modeling complex multi-issue negotiations using utility graphs. In *Proceedings of AAMAS'05*, pages 280–287, 2005.

[94] A. Rosenfeld, I. Zuckerman, E. Segal-Halevi, O. Drein, and S. Kraus. NegoChat-A: a chat-based negotiation agent with bounded rationality. *Autonomous Agents and Multi-Agent Systems*, pages 1–22, 2015.

[95] A. Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):pp. 97–109, 1982.

[96] F. Sadri, F. Toni, and P. Torroni. Logic agents, dialogues and negotiation: an abductive approach. In *Symposium on information agents for electronic commerce, York*, pages 71–78. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour, 2001.

[97] T. Sandholm and V. Lesser. Leveled Commitment Contracts and Strategic Breach. *Games and Economic Behavior*, 35:212–270, January 2001.

[98] M. Shanahan. The event calculus explained. In M. J. Wooldridge and M. Veloso, editors, *Artificial intelligence today*, pages 409–430. Springer-Verlag, Berlin, Heidelberg, 1999.

[99] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2008.

[100] K. Sim and B. Shi. Adaptive commitment management strategy profiles for concurrent negotiations. In T. Ito, M. Zhang, V. Robu, S. Fatima, and T. Matsuo, editors, *Advances in Agent-Based Complex Automated Negotiations*, volume 233 of *Studies in Computational Intelligence*, pages 177–195. Springer Berlin / Heidelberg, 2009.

[101] K. M. Sim. A market-driven model for designing negotiation agents. *Computational Intelligence*, 18(4):618–637, 2002.

[102] K. M. Sim and B. An. Evolving best-response strategies for market-driven agents using aggregative fitness GA. *IEEE Transactions on Systems, Man and Cybernetics*, 39(3):284–298, 2009.

[103] K. M. Sim and C. Y. Choi. Agents that react to changing market situations,. *IEEE Transaction on Systems, Man and Cybernetics, Part B: Cybernetics*, 33:188–201, April 2003.

[104] K. M. Sim and B. Shi. Concurrent negotiation and coordination for grid resource coallocation. *IEEE transactions on systems, man and cybernetics. Part B*, 40:753–766, June 2010.

[105] T. Skylogiannis, G. Antoniou, N. Bassiliades, G. Governatori, and A. Bikakis. DR-NEGOTIATE - a system for automated agent negotiation with defeasible logic-based strategies. *Data & Knowledge Engineering*, 63:362–380, November 2007.

[106] R. G. Smith. The Contract Net Protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113, Dec. 1980.

[107] K. Stathis and F. Toni. The KGP model of agency for decision making in e-negotiation. In *Joint-Workshop on Decision Support Systems, Experimental Economics E-Participation*, June 2005.

[108] K. Sycara and T. Dai. Agent reasoning in negotiation. In D. M. Kilgour and C. Eden, editors, *Handbook of Group Decision and Negotiation*, volume 4 of *Advances in Group Decision and Negotiation*, pages 437–451. Springer Netherlands, 2010.

[109] E. Tsang, R. Olsen, and S. Masry. Event calculus on high frequency finance. Technical report, Working Paper WP038-10, Centre for Computational Finance and Economic Agents (CCFEA), University of Essex, 2010.

[110] Y. Tsuruhashi and N. Fukuta. A framework for analyzing simultaneous negotiations. In G. Boella, E. Elkind, B. Savarimuthu, F. Dignum, and M. Purvis, editors, *PRIMA 2013: Principles and Practice of Multi-Agent Systems*, volume 8291 of *Lecture Notes in Computer Science*, pages 526–533. Springer Berlin Heidelberg, 2013.

[111] M. P. Wellman, A. Greenwald, and P. Stone. *Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2007.

[112] P. West, D. Ariely, S. Bellman, E. Bradlow, J. Huber, E. Johnson, B. Kahn, J. Little, and D. Schkade. Agents to the rescue? *Marketing Letters*, 10(3):285–300, 1999.

[113] D. Weyns. *Architecture-Based Design of Multi-Agent Systems*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[114] C. R. Williams. *Practical Strategies for Agent-Based Negotiation in Complex Environments*. PhD thesis, University of Southampton, December 2012.

[115] C. R. Williams, V. Robu, E. H. Gerding, and N. R. Jennings. Negotiating concurrently with unknown opponents in complex, real-time domains. In *20th European Conference on Artificial Intelligence*, volume 242, pages 834–839, August 2012.

[116] M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition, 2009.

[117] M. Wooldridge and P. Ciancarini. Agent-oriented software engineering: The state of the art. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in Computer Science*, pages 1–28. Springer Berlin Heidelberg, 2001.

[118] D. Zeng and K. Sycara. Bayesian learning in negotiation. *International Journal of Human-Computer Studies*, 48(1):125–141, 1998.