UNIVERSITÀ DEGLI STUDI DI TORINO
Research Doctorate in Science and High Technology
Specialization in Computer Science

Ph.D. Thesis
(Revised version dated February 8, 2013)

2CL Protocols: Interaction Patterns Specification in Commitment Protocols

Elisa Marengo

Advisor: Dr. Matteo Baldoni
Co-Advisor: Dr. Cristina Baroglio

This version has been registered as Technical Report RT 147/13,
Università degli Studi di Torino, Dip. Informatica, Torino, Italy, 2013.

ABSTRACT

Commitment protocols have been deeply investigated in the context of multiagent systems (MAS) as a valuable way for specifying interaction and communication protocols. Commitments capture social relations among the interacting parties, expressing the engagement of a *debtor* towards a *creditor* agent, to do something or to achieve some condition of interest. By specifying only *what* agents are expected to do, rather then how they are expected to satisfy their commitments, this kind of specification is highly appreciated because it leaves great freedom of behaviour to agents. However, in many practical contexts there is the need of capturing some *patters* that the interaction is desired to respect and that improves coordination. The contribution of this thesis goes in this direction, by presenting an extension of commitment protocols which allows one to express a set of desired patterns given as *constraints* among facts and commitments. Being the approach declarative, it still provide a high flexibility to agents. More in detail, the proposal is to explicitly account for a *regulative* component of the specification, capturing the set of constraints on the interaction, and a separate and decoupled *constitutive* specification defining the set of actions that can be performed. The advantages of this decoupling is a greater *modularity* in the specification which entails an easier protocol design, re-use and extension. An operational semantics is also presented. It is obtained as an extension of the commitment machine in such a way to consider the role of constraints in deciding which of the possible interactions can be considered as legal for the protocol.

The proposal is validated by modelling some real case studies and by showing the advantages of its adoption with respect to other proposals from the literature. In order to support the design and the analysis of protocol specifications, a tool offering different kinds of functionalities is presented. Among these, a prolog program allows for the generation of the *labelled* graph of the possible interactions, where labels are used to represent the violation of constraints and of commitments.

PUBLICATIONS OF THE AUTHOR

PUBLICATIONS RELATED TO THE TOPIC OF THE THESIS

In the following is reported the list of the author's publications that are related to the topics discussed in the thesis (updated to October 2012). For each paper is reported, in brackets, the list of chapters whose topic is strongly related to the content of the publication.

International Journals

- M. Baldoni, C. Baroglio, I. Brunkhorst, N. Henze, E. Marengo, and V. Patti. Constraint Modeling for Curriculum Planning and Validation. *International Journal of Interactive Learning Environments*, 19(1):83–123, 2011. (Chapters 2, 3)
- M. Baldoni, C. Baroglio, E. Marengo, and V. Patti. Constitutive and Regulative Specifications of Commitment Protocols: a Decoupled Approach. *ACM Transactions on Intelligent Systems and Technology, Special Issue on Agent Communication*, 4(2), 2013. (Chapters 3, 5)
- M. Baldoni, C. Baroglio, E. Marengo, V. Patti, and F. Capuzzimati. Engineering Commitment-based Business Protocols with 2CM. *Submitted to Journal of Autonomous Agents and Multi-Agent Systems* on December 2012. (Chapters 6, 7)

Chapters in books

- M. Baldoni, C. Baroglio, V. Patti, and E. Marengo. Supporting the Analysis of Risks of Violation in Business Protocols: the MiFID Case Study. In M. De Marco, D. Te'eni, V. Albano, and S. Za, editors, *Information Systems: Crossroads for Organization, Management, Accounting and Engineering*. Springer, 2012. Best Track Paper Award. (Chapter 7)

International Conferences and Lecture Notes

- M. Baldoni, C. Baroglio, F. Capuzzimati, E. Marengo, and V. Patti. A Generalized Commitment Machine for 2CL Protocols and its

Implementation. In M. Baldoni, L. Dennis, V. Mascardi, and W. Vasconcelos, editors, *Post-Proc. of International Workshop on Declarative Agent Languages and Technologies (DALT 2012). Revised, selected and invited papers*. vol. *LNAI*, In press. (Chapters 4, 6)

- M. Baldoni, C. Baroglio, F. Bergenti, E. Marengo, V. Mascardi, V. Patti, A. Ricci, and A. Santi. An Interaction-oriented Agent Framework for Open Environments. In R. Pirrone and F. Sorbello, editors, *Proc. Artificial Intelligence Around Man and Beyond, 12th Congress of the Italian Association for Artificial Intelligence (AI*IA 2011)*, vol. 6934 of *LNAI*. Palermo, Italy, September 2011. Springer. (Chapter 8)

- M. Baldoni, C. Baroglio, and E. Marengo. Commitment-based Protocols with Behavioral Rules and Correctness Properties of MAS. In A. Omicini, S. Sardina, and W. Vasconcelos, editors, *Post-Proc. of the 8th International Workshop on Declarative Agent Languages and Technologies VIII (DALT 2010), Revised Selected and Invited Papers*, number 6619 in LNAI. Springer, 2011. (Chapters 3, 5)

- E. Marengo, M. Baldoni, and C. Baroglio. Extend Commitment Protocols with Temporal Regulations: Why and How. C. Damasio, A. Preece and U. Straccia, editors, *Proc. of 5th International Symposium on Rules, Doctoral Consortium, RuleML 2011*, Barcellona, Spain, number CoRR abs/1107.2086, July 2011. CoRR, Cornell University Library. (Chapters 1,2,3,5)

- E. Marengo, M. Baldoni, and C. Baroglio. On Temporal Regulations and Commitment Protocols. In Toby Walsh, editors, *Proc. of the 22nd International Joint Conference on Artificial Intelligence*, Barcelona, Catalonia, Spain, July 16-22, 2011, pages 2824-2825. AAAI Press / International Joint Conferences on Artificial Intelligence. (Chapters 1,2,3)

- E. Marengo, M. Baldoni, C. Baroglio, A. K. Chopra, V. Patti, and Munindar P. Singh. Commitments with Regulations: Reasoning about Safety and Control in REGULA. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2011*, vol. 2, pages 467–474. Taipei, Taiwan, May 2011. IFAAMAS. (Chapters 5, 8)

- M. Baldoni, C. Baroglio, and E. Marengo. Behavior-oriented Commitment-based Protocols. In H. Coelho, R. Studer, and M. Wooldridge, editors, *Proc. of 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 137–142, Lisbon, Portugal, August 2010. IOS Press. (Chapter 3)

Workshops and National Conferences

- M. Baldoni, C. Baroglio, E. Marengo, and V. Patti. Grafting Regulations into Business Protocols: Supporting the Analysis of Risks of Violation. In A. Antón, D. Baumer, T. Breaux, and D. Karagiannis, editors, *Fourth International Workshop on Requirements Engineering and Law (RELAW 2011), in conjunction with the 19th IEEE International Requirements Engineering Conference*, Trento, Italy, August 30th 2011. IEEE. (Chapter 6, 7)

- M. Baldoni, C. Baroglio, E. Marengo, V. Patti, and F. Capuzzimati. Learn the Rules so you Know How to Break them Properly. In G. Fortino, A. Garro, L. Palopoli, W. Russo, and G. Spezzano, editors, *Proc. of WOA 2011: Dagli oggetti agli agenti, Progettazione ed analisi di sistemi complessi mediante modellazione e simulazione basate su agenti*, vol. 741, pages 28–36, Cosenza, Italy, July 2011. CEUR Workshop Proceedings. (Chapter 6, 7)

- M. Baldoni, C. Baroglio, E. Marengo, V. Patti, and A. Ricci. Back to the Future: an Interaction-oriented Framework for Social Computing. In A. K. Chopra, F. Dalpiaz, and S. O. Lim, editors, *First International Workshop on Requirements Engineering for Social Computing (RESC 2011), in conjunction with the 19th IEEE International Requirements Engineering Conference*, Trento, Italy, August 29th 2011. IEEE. (Chapters 3, 8)

- M. Baldoni, C. Baroglio, F. Bergenti, A. Boccalatte, E. Marengo, M. Martelli, V. Mascardi, L. Padovani, V. Patti, A. Ricci, G. Rossi, and A. Santi. MERCURIO: An Interaction-oriented Framework for Designing, Verifying and Programming Multi-Agent Systems. In N. Fornara and G. Vouros, editors, *Proc. of the 3rd Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW'10), 11th International Workshop on Coordination, Organization, Institutions and Norms in Multi-Agent Systems (COIN@-MALLOW 2010)*, vol. 627, Domain Valpré in Lyon, France, August 30 - September 2 2010. CEUR Workshop Proceedings. (Chapters 3, 8)

- M. Baldoni, C. Baroglio, and E. Marengo. Constraints among Commitments: Regulative Specification of Interaction Protocols. In A. Artikis, J. Bentahar, A. K. Chopra, and F. Dignum, editors, *Proc. of International Workshop on Agent Communication (AC 2010), held in conjuction with AAMAS 2010*, pages 2–18, Toronto, Canada, May 2010. (Chapter 3)

PUBLICATIONS NOT RELATED TO THE TOPIC OF THE THESIS

List of the author's publications that are not related or that are marginally related to the topics discussed in the thesis (updated to October 2012).

International Journals

- M. Baldoni, C. Baroglio, E. Marengo, V. Patti, and C. Schifanella. Flexible Choreography-driven Service Selection. *Journal of Intelligenza Artificiale*, IOS Press, volume 6, number 1, pages 97–115, 2012.

- M. Baldoni, C. Baroglio, E. Marengo, V. Patti, and C. Schifanella. Models of agent interaction based on modal logics. *International Journal of Intelligenza Artificiale*, 5(1):83-88, February 2011.

National Journals

- M. Baldoni, and E. Marengo. Validazione e Pianificazione Automatica di Percorsi Formativi. Invited paper for the AICA Journal *"Mondo Digitale"*, 36:64–74, 2010.

International Conferences and Lecture Notes

- M. Baldoni, C. Baroglio, and E. Marengo. Curricula Modeling and Checking. In R. Basili and M. T. Pazienza, editors, *Proc. of AI*IA 2007: Advances in Artificial Intelligence*, 10th Congress of the Italian Association for Artificial Intelligence, LNAI 4733, pages 471 – 482, Rome, Italy, September 2007. Springer.

- M. Baldoni and E. Marengo. Curriculum Model Checking: Declarative Representation and Verification of Properties. In E. Duval, R. Klamma, and M. Wolpers, editors, *Proc. of Second European Conference on Technology Enhanced Learning* (EC-TEL 2007), LNCS 4753, pages 432–437. Springer, 2007.

- M. Baldoni, C. Baroglio, I. Brunkhorst, E. Marengo, and V. Patti. Reasoning-based Curriculum Sequencing and Validation: Integration in a Service-Oriented Architecture. In E. Duval and R. Klamma, and M. Wolpers, editors, *Proc. of Second European Conference on Technology Enhanced Learning* (EC-TEL 2007), LNCS 4753, pages 426–431. Springer, 2007.

- M. Baldoni, C. Baroglio, I. Brunkhorst, E. Marengo, and V. Patti. A Personalization Web Service for Curricula Planning and Validation (poster). In W. May and M. Kifer, editors, *Poster Presentation at the 4th European Semantic Web Conference* (ESWC 2007), 2007.

Workshops and National Conferences

- M. Baldoni, C. Baroglio, E. Marengo, C. Schifanella, and V. Patti. Joint Achievement of Services' Personal Goals. In M. Baldoni, C. Baroglio, J. Bentahar, and V. Mascardi, editors, *Proc. of the 2nd Multi-Agent Logics, Languages, and Organisations Federated Workshops*, (MALLOW'009), volume 494, pages 5-13, Turin, September 2009. CEUR Workshop Proceedings.

- M. Baldoni, C. Baroglio, G. Berio, and E. Marengo. Declarative Representation of Curricula Models: an LTL- and UML-based approach. In M. Baldoni, A. Boccalatte, F. De Paoli, M. Martelli, and V. Mascardi, editors, *Proc. of WOA 2007: Dagli oggetti agli agenti, Agenti e Industrie: Applicazioni tecnologiche degli agenti software*, pages 34–41, Genova, Italy, September 2007. Seneca Edizioni.

- M. Baldoni, C. Baroglio, I. Brunkhorst, E. Marengo, and V. Patti. A Service-Oriented Approach for Curriculum Planning and Validation. In *Proceedings of the Multi-Agent Logics, Languages, and Organisations, Federated Workshops* (MALLOW'007), pages 108–123, Durham, GB, September 2007.

- M. Baldoni, C. Baroglio, I. Brunkhorst, N. Henze, E. Marengo, and Viviana Patti. A Personalization Service for Curriculum Planning. In E. Herder and D. Heckmann, editors, *Proc. of the 14th Workshop on Adaptivity and User Modeling in Interactive Systems* (ABIS 2006), pages 17–20, Hildesheim, Germany, October 2006.

ACKNOWLEDGMENTS

To be honest, writing this thesis has been sometimes quite hard, not only for the efforts that this kind of task requires, but also because of the importance I have placed on it. However, I can now say that my efforts have been well worthwhile. Without any doubt, this work was made possible thanks to many people who have supported me. I would like to thank all of them: those who present me of suggestions and directions on the thesis and those who present me of their presence and encouraging words. Certainly, nothing would have been possible without the help of Matteo Baldoni and Cristina Baroglio: I am grateful to them for their friendship and for all the things they taught me. I am also thankful to all the people I had the pleasure to work with, for their competence, patience, support and for their friendship. Among them a special thank to Federico Capuzzimati and Viviana Patti.

I would also like to thank the reviewers, Dr. Paolo Torroni (Università di Bologna) and Prof. Michael Winikoff (University of Otago) for their time, suggestions and helpful observations.

The life of a Ph.D. Student is sometimes complicated. During the most difficult moments I had once again the evidence to be surrounded by *Friends*. This is definitely the case of Chiara, Daniel, Laura, Lorena, Massimo, Stefano and Valentina. I am also grateful to all the nice people from the department with whom I have spent very nice and fanny moments, and with whom I have shared doubts and worries.

Finally, I would like to thank those that helped, supported and trusted me the most: my father, my mother and my sister Lara. Everyday they present me of their selfless and unreserved love, without which nothing of this would had never been possible.

Sincerely, thanks.

Elisa

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION AND MOTIVATIONS

Starting from the late 1980s, *multiagent systems* have received an increasing attention, and they have been adopted in a variety of different applications ranging from simple to complex systems with stringent requirements. One reason for this growing attention is that multiagent systems represent a natural abstraction for reasoning on many (social) mechanisms and for solving complex tasks (like planning strategies for problem solving, negotiation, knowledge representation, inference mechanisms and such like). This thesis finds its place in the area of agent coordination and interaction and, in particular, in the context of research on the representation of interaction protocols.

Interaction protocols represent a valuable mechanism for reaching agents coordination. Coordination supports different aims, like the achievement of desired objectives [vRW07, BBM+09, BRVS11] in a given way; the sharing of knowledge or of capabilities between agents for solving tasks they would not be able to solve alone; the cooperation between the interacting agents; the representation of agreements on how to carry on a contract between different parties. Besides representing a way for expressing a desired coordination, interaction protocols specification allows agents to understand in advance if the protocol provides them enough guarantees of achieving their objectives. Moreover, it allows for the verification of properties (e.g. interoperability, absence of deadlock or livelock) before the interaction starts [BBC+09, BBM11a, EMBD11].

In the literature it is possible to find different approaches for the specification of interaction protocols. Among these, *procedural approaches* assume to have control over the flow of the interaction by rigidly specifying the allowed sequences of actions. *Mentalistic approaches*, instead, assume to have access to the mental states of the agents. They specify how the agent's beliefs change as a consequence of the occurrence of a certain action. In general, for a specification to be effective, and thus to reach its aim, it is fundamental to consider the characteristics of the interacting parties and of the systems it will be settled in. Specifically, since agents are conceived as *autonomous* entities [RN03] it is not possible to assume to have control over the flow of the interaction. Agents, indeed, are free to decide how to behave and to deviate, at any time, from the protocol specification. Due to the agents' autonomy, it is not even possible to assume to have control over their mental states. It is unlikely, indeed, that agents will be willing to disclose their internal functioning.

Commitment protocols [Cas95, Sin99] propose a different mechanism for protocol specification, without assuming to have control over the flow of the interaction and moving towards a social semantics, that allows the verification by observation of the ongoing interaction [VS99]. A commitment represents the engagement of a *debtor* agent towards a *creditor* agent to achieve a certain condition. Agents are free to decide which path to run for satisfying the condition at issue. This makes commitment protocols more robust to handle *exceptions* and it allows agents to take advantage of *opportunities* that may arise: every execution such that at the end all commitments are satisfied, is allowed. Moreover, being the semantics of the protocol actions shared and agreed by the participants, the verification is performed by observing the interaction: everyone that is able to observe an action and that knows its semantics is able to infer the corresponding commitments and whether the debtor is behaving in accordance with them. All these aspects make commitment protocols fit well with the agents and the systems' characteristics.

Even though interaction protocols do not have the control over the flow of interaction, in many contexts there is the need of representing agreements, conventions, norms, orderings or causality relations on the interaction, that agents are expected to respect. These requirements are not limited to conditions to be achieved, but they include *constraints* on how the interaction should be carried on. We refer to these requirements as *patterns of interactions*. Let us consider some examples.

Example 1.1 (Democratic assembly)**.** *In the context of a democratic assembly there are some behavioural rules the participants are expected to follow [REHB00]. For instance, participants are expected to ask the floor and to obtain it, before speaking. Similarly, the chair has the assignment to open the voting session. Only after its opening a participant is allowed to vote.*

Example 1.2 (Personal Data Management)**.** *The management of personal data is a delicate issue. The data holder is required to respect different requirements [Org80]. For instance, when data are requested it is expected to check the compliance between the purposes for which they are stored and those for which they are required before sending them. Moreover, before sending the data he/she has to ask the authorization to the data owner. After data are sent, the data holder is expected to send a notification to the data owner.*

Example 1.3 (Financial markets). *After the crisis of financial markets, new regulations have been introduced so as to provide more guarantees to buyers and sellers [Mifa]. For instance, financial promoters are now required to supply all the necessary information and documentation before the client signs an order. Moreover, they have to ensure that the financial product presents a level of risks adequate to the investor's requirements. This should be obligatorily done before offering a product to the investor.*

In this thesis we argue that, even though commitments provide a flexible means for specifying the engagements among autonomous and heterogeneous agents, they are not expressive enough for capturing patterns of interaction. As a consequence they are not able to handle realistic application scenarios where patterns of interaction are central because of domain conventions or of other coordination requirements.

This thesis faces the problem of how to extend commitment protocols with the possibility of expressing patterns of interaction. This objective comes with a series of requirements aimed at avoiding the nullification of the advantages of adopting commitment protocols. In particular, the challenges we address are: (i) how to capture patterns of interaction without compromising the flexibility and verifiability of commitment protocols; (ii) how to express such patterns in a way that accounts also for the agents' autonomy; (iii) how to provide a protocol specification that can be easily adapted to changing requirements or to different scenarios, so as to maximise the reuse of a specification; (iv) how to support users and designers in the understanding of the requirements imposed by a specification.

REPRESENTATION OF INTERACTION PATTERNS

As depicted in Figure 1.1, besides the set of roles, representing the interacting parties, and a set of protocol actions, our representation explicitly accounts for patterns of interaction as part of the protocol specification. For their representation we define a new language named *Constraints among Commitments Language* (2CL). 2CL is a declarative language that allows for the specification of constraints between facts and commitments. These constraints aim at regulating the achievement of different conditions, basically defining the allowed evolutions of the social state. The language defines different operators that allow the specification of different kinds of requirements. These operators can be grouped into temporal operators, capturing a relative order on the achievement of different conditions, and relation operators, capturing

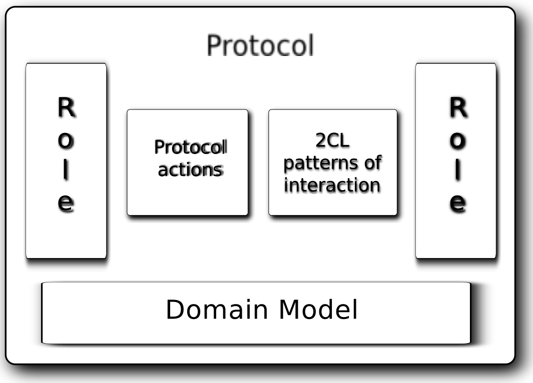


Figure 1.1: Sketch of 2CL protocol representation. The specification explicitly account for patterns of interaction.

co-occurrence or mutual exclusion between different conditions. We define the 2CL-Generalized Commitment Machine (2CL-GCM). It provides an operational semantics for 2CL protocols as an extension of Singh's Generalized Commitment Machine (GCM) [Sin07]. To this aim we associate an LTL interpretation to 2CL constraints.

FLEXIBILITY AND AGENT'S AUTONOMY. 2CL allows addressing the flexibility of the specification and the agents' autonomy thanks to its declarative nature. Specifically, constraints capture only mandatory (or forbidden) behaviours. For all the aspects that are not specified, agents are free to decide how to behave. This is in accordance with agents autonomy of deciding how to behave and to potentially take advantage from the opportunities. Additionally, since 2CL constraints rule the evolution of the social state they inherit from commitment protocols the possibility of being verified by simply observing the interaction among the interacting parties. The achievement of the condition they rule, indeed, can be inferred by the executed actions and their meaning.

ADAPTATION. With adaptation we refer to the property of a specification of being reused in different contexts (possibly with different requirements) without having to redefine it from scratch. The modularity of a 2CL protocol allows an easy extension of a specification by including new regulations. Some examples (in particular in Chapter 7), will show how 2CL constraints can be used as a means for expressing the *grafting* points between the new regulations (often involving new activities to be performed) on the existing specification. Moreover, the choice of defining constraints in terms of commitments and facts allows for a greater decoupling with the protocol actions. As a consequence the specification tends to be more easily adaptable and customizable. Protocol actions and protocol constraints, indeed, can be modified independently from one another (for instance by adding new constraints or modifying the existing ones).

SPECIFICATION UNDERSTANDABILITY. Understandability of the specification is a central aspect both for a designer and for an interacting agent. Given a specification it is important to understand which behaviours are allowed and which are forbidden. This is useful for an interacting party, for instance, to avoid incurring in sanctions or penalties. It is also important for a designer, who needs to be sure that the specification he/she has provided do not allow for behaviours that should be forbidden (and the other way around).

In order to support understandability, we associate a graphical representation to each operator in the language. This helps understanding the flow of the interaction captured by constraints, although they do not impose any rigid flow on the interaction. Additionally, we define a graphical tool for building 2CL protocol specifications and for their visualisation. The tool allows for the computation and the visualisation of the labelled graph of the possible interactions. Each path of the graph is labelled as *legal* if by undertaking such interaction all constraints are satisfied. Otherwise it is labelled, following some graphical conventions, as an illegal path. The graph supplies an overall view of the interactions that an agent can undertake, giving an immediate perception of the behaviour an agent has to adopt if it wants to run on a legal path.

OUTLINE OF THE THESIS

The Thesis is structured as follow:

CHAPTER 2 presents an overview of interaction protocols specifications. Starting from procedural and mentalistic approaches, we then focus on commitment protocols. The chapter also describes some approaches that are not based on commitments, but from which we have drawn inspiration.

CHAPTER 3 presents the 2CL protocol specification. In particular, it presents the Constraints among Commitments Language for patterns specification and the graphical representation associated to each operator.

CHAPTER 4 presents an LTL interpretation of 2CL constraints and provides the operational semantics of 2CL protocols. The operations semantics is given as extension of the Generalised Commitment Machines defined by Singh in [Sin07].

CHAPTER 5 compares the 2CL approach for patterns representation with other proposals in the literature, highlighting differences and advantages.

CHAPTER 6 presents the implementation of a tool that supplies a graphical and overall view of the possible interactions given a protocol specification. Possible interactions are labelled as legal or illegal, depending on whether they satisfy the constraints or violate some of them.

CHAPTER 7 shows how a 2CL protocol specification can be easily extended so as to take in new regulations that arise at a certain point. To this aim two real case studies are presented and analysed.

CHAPTER 8 summarises the contributions of the thesis and presents some future directions.

2 INTERACTION PROTOCOL SPECIFICATION: OVERVIEW

CONTENTS

In this chapter we present an overview of interaction protocols specifications in multiagent systems. After describing the main characteristics of procedural and mentalistic approaches we summarise the motivations that led to the definition of social commitments. We describe the main characteristics of commitments and of commitment protocols. At the end we consider some proposals from neighbouring fields from which we draw inspiration.

2.1 INTERACTION PROTOCOLS IN MULTIAGENT SYSTEMS

Multiagent systems involve autonomous and heterogeneous agents. Autonomy entails that it is not possible to tell agents what to do (they can accept or refuse to do tasks) and that is not possible to access their internal functioning. Heterogeneity means that they are implemented in different ways. Therefore, they may have different architectures and they may have different interests on how to carry on the interaction.

In this setting, in order for a set of agents to become a multiagent system it is necessary to define proper coordination mechanisms. Coordination can be used for different aims, such as for the representation of dependences between actions, or as a way for meeting global constraints that agents alone are not able to reach (due to the lack of resources, information or competences) [Wei99, Chapter 2]. However, due to autonomy and heterogeneity of the agents, it is not possible to assume to have control over the *execution flow*. For this reason methods such as *Distributed Programming*, *Object-oriented* programming or *methods invocation* are not applicable to MAS [Wei99, HK03, CAB+11].

Interaction protocols represent a coordination mechanism that guides the interaction between agents, by defining a set of rules to be respected along the interaction. The problem of how to represent interaction protocols has been widely studied in multiagent systems, thus resulting in different approaches. Some of these are based on Petri Nets [KFD98, Lee99, CCF+99, EFSHM01] and finite state machines [KIO95, MBF95]. Some others are UML-based, like [OVDPB00], Agent UML (AUML) [BMO01, Fou03] or propositional state charts [DDCP05]. Basically, these approaches define what messages are possible at any point of the interaction. These messages are at the basis of the communication and of the information exchanged between the interacting parties.

Agent Communication Languages

In order for the interaction to be profitable it is necessary not only to define the order of exchanged messages, but also to define their meaning so as to make them comprehensible for the agents. In the context of open and heterogeneous systems a natural way to proceed is by defining an agent communication language and to associate with it a standard semantics. Among the others, two well known standards are KQML [Fin94, FFMM94] and FIPA ACL [Fou02a]. In particular, FIPA ACL defines a set of message performatives and associates with them a semantics given in terms of the agent's mental states [BIP88, RG95, Woo02]. This is known as the *mentalistic approach* to the communication. Roughly, to each performative is associated a formula that expresses a condition to be satisfied on the agent's mental state (usually on its beliefs) in order for the message to be sent by the agent or as a consequence of the receipt of the message.

Example 2.1. *Consider the semantics of the* inform *performative, according to FIPA standard [Fou02b]*

$\langle i, \textit{inform}(j, \varphi)\rangle$

  feasibility precondition: $B_i\varphi \wedge \neg B_i(Bif_j\varphi \vee Uif_j\varphi)$

  rational effect: $B_j\varphi$

This definition states that an agent i sending an inform to agent j about φ is respecting the inform semantics if it believes φ and it is not the case that it believes either that j already believes whether φ is true or false, or that j is uncertain of the truth value of φ. The receipt of the message makes j to believe φ.

Pros of interaction protocols

The adoption of interaction protocols brings some interesting advantages on the definition and the analysis of the interaction: *(i)* it allows to specify the interaction in terms of *roles*, thus abstracting from the actual interacting agents; *(ii)* it allows for *a-priori properties verification*: some properties of interest (such as absence of deadlock, fairness, safety) can be verified on the protocol, before the interaction starts [EMBD10, EMBD11]; *(iii)* the correctness of the agents' interaction w.r.t. the protocol specification can be verified at *run-time*; *(iv)* agents can check, before the interaction takes place, if they would be able to behave in accordance with the specification (*conformance testing*). If the interacting agents conform to the specification of the role they play, the interaction inherits the properties verified on the protocol. For instance, if in a protocol specification roles are proved to be interoperable, any agent that accepts to conform to the protocol is ideally guaranteed that its interaction with any other agents, playing the other roles and compliant with their specification, will succeed [AHKV98, RR02, BZ09].

Procedural specifications

A well known approach to protocol specification is to *procedurally* specify the allowed interactions. This basically means that the specification defines which actions are executable at each point of the specification. Every interaction that is not explicitly foreseen is forbidden. These approaches are known as *procedural specifications* [Mon09, YS02a]. Finite state machines (FSM) are a well known formal description technique of this kind. They are basically oriented graphs, where transitions from a state to another represent the messages that can be exchanged at each point. The allowed interactions are (only) those that correspond to a path in the graph. FSMs are at the basis of some proposals such as COOL [MBF95] and AgentTalk [KIO95]. The advantages of FSM for protocol representation are that they are intuitive and the semantics is formally defined. Moreover, they have been deeply studied and widely adopted, leading to the definition of several tools that support their specification and analysis. However, FSMs present some shortcomings. Specifically, they do not allow for the representation of concurrency and they do not differentiate between outgoing and incoming messages. Moreover, by rigidly specifying the allowed interactions they limit the agents' autonomy of deciding how to behave. For instance, agents cannot take advantage from the opportunities that may arise during the interaction or to handle exceptions [YS02a]. Opportunities are intended as the possibility for a participant to choose an interaction that fits with its interests, or to simplify the interaction thanks to some additional knowledge it has (that, for instance, allows it to jump some steps like the request of some information) [YS02a].

Similar considerations hold also for other procedural approaches, like those based on Petri Nets [KFD98, Lee99, CCF^{+}99, EFSHM01]. As well as FSM, indeed, Petri Nets provide a formal semantics and tools for the design and validation of the net (like GreatSPN [AMBC^{+}95]).

Additionally, they allow to represent concurrency. However, also Petri Nets specifications are too rigid to suit with the agent's autonomy.

Another well known formalism for protocol specification is Agent UML [BMO01, Fou03]. This approach basically extends UML sequence diagrams, in order to account for some characteristics of agents and not of objects (such as the multiplicity of messages, type of delivery, messages sent in parallel). Being based on UML, this representation takes advantage from different tools that have been developed for UML diagrams. Moreover, as well as FSM, also in this approach a specification is easy to understand. However, this approach lacks of proposal for validation or properties verification and it is another example of procedural representation.

Summarising, the main shortcoming of procedural approaches is that, by rigidly specifying the allowed interactions, they do not leave agents the possibility of engaging in flexible behaviours. Moreover, these approaches mainly focus on message ordering. Either they do not account for a semantics of the exchanged messages, or they rely on standard communication languages, like FIPA ACL, based on a mentalistic semantics. In the first case, the communication lacks of a clear meaning. Messages are simply tokens that are exchanged between agents [Sin00]. The adoption of a mentalistics semantics, instead, has been accused of being not adequate for open and heterogeneous systems for two main reasons [Sin03a, TCY$^+$09]: *(i)* it imposes a limitation on the agent's architecture, that is assumed being based on mental states. This aspect impacts on the heterogeneity of the system; *(ii)* it is not possible to verify the correctness of agents behaviours with respect to the specification [Sin03a]. Mentalistics approaches, indeed, are verifiable only by performing introspection in the agent's mental states. This assumption, however, conflicts with the agent's autonomy.

Example 2.2. *Considering Example 2.1, for instance, it is not possible to verify that the receipt of the message makes the receiver agent to believe its content. It is not even possible to verify that the sender complies with the* feasibility precondition.

An example of language that is based on the mentalistic semantics is Dynamics in LOGic [BMPG04, DYn]. It is a logic programming language based on modal logics, allowing for the specification of intelligent agents reasoning and behaviours. Specifically, agents programmed in Dynamics in LOGic can choose a course of actions by considering the beliefs they have about the world and the other agents, and by considering the actions that they can perform. By means of the *communication kit* [BBMP03b, BBMP03a] which encompasses communicative acts, agents can interact according to a conversation protocol. By reasoning on it, agents can determine whether, by conforming to the protocol's rules, they will be able to achieve some desired goals. If this is not the case, an agent can decide to refuse the protocol and to look for another one that better suits its needs. Being based on a mentalistic semantics, this approach suffers of the shortcomings already described (i.e. agents' architecture is expected to be based on mental states, and that verification of agents' behaviour cannot be performed).

Commitments and the social approach

Commitments have been introduced as an alternative to mentalistic approaches for providing semantics to messages exchanged during an interaction. The idea on which they rely is to provide a social meaning to the messages based on visible effects on the agent's social environment [VS99, Sin03a, TCY$^+$09]. This semantics, known as the *social* approach to the communication, allows to abstract from any internal representation of the agents and it can be verified without performing introspection in the agents' mental states: «*social semantics have proven to be a viable approach to accommodate truly open agent societies, since they do not pose restrictions of any sort on the nature and architecture of interacting parties.*» ([TCY$^+$09]) Of course, it is still possible to think of agents as being represented in terms of mental states, but commitments allow also for other representations. Fornara and Colombetti, in their works [For03, FC04a, FC04b], propose a new semantics for FIPA ACL, where the semantics of the different performatives is given in terms of commitments. As a difference with the mentalistic semantics given by FIPA, in this way communicative messages acquire a social and shared meaning. When messages are exchanged between agents, it is possible to objectively determine their effects, and on this basis to judge the agent's compliance.

Commitment for specifying interaction protocols

After the introduction of commitments, different proposals for commitment-based interaction protocols started to appear in the agents literature. A great advantage of relying on commitments is that the protocol inherits the possibility of verifying the agents' compliance. The requirements on the interaction, indeed, are expressed in terms of observable behaviours [VS99, TCY$^+$09]. Moreover, since commitments only capture conditions to be achieved, without specifying how, or which action must be performed [Sin03a], commitment protocols allows for *flexible* interactions. Agents, therefore, are not prevented from taking advantage of opportunities that may arise and to handle exceptions.

2.2 COMMITMENT APPROACHES

Commitments have been introduced by Castelfranchi [Cas95] and Singh [Sin99]. Both understand a commitment as a social relation be-

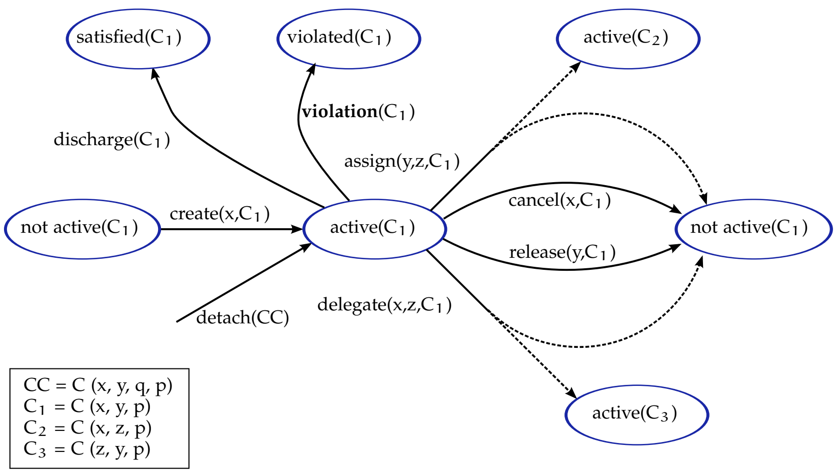


Figure 2.1: Representation of the life cycle of an unconditional commitment.

tween two agents: the debtor and the creditor of the commitment. The relational aspect that ties the two agents is a fundamental aspect that characterises a commitment as being more than just shared knowledge or intentions [Sing91, Sing92]: « *Social Commitment is not an individual commitment shared by many agents; it is the Commitment of one agent to another.*» ([Cas95])

In this thesis we rely on Singh's definition, where a commitment is a relation of the form:

$$\mathsf{C}(debtor, creditor, ant\_cond, cons\_cond)$$

It represents the fact that the *debtor* commits to the *creditor* to bring about the consequent condition *cons_cond*, when the antecedent condition *ant_cond* holds. In the special case when the antecedent condition is true, the commitment is represented with the short notation:

$$\mathsf{C}(debtor, creditor, cons\_cond)$$

Example 2.1. *The commitment* $\mathsf{C}(seller, client, pay, ship)$ *represents the engagement of the* seller *towards the* client *that if the client* pays, *then the seller will* ship *the goods.*

Commitments life cycle

Commitments can be manipulated by means of a set of operations: CREATE, DISCHARGE, DETACH, CANCEL, RELEASE, ASSIGN and DELEGATE (see Table 2.1 for more details). Depending on the operations that are performed on it, a commitment can be in different *states*. Figure 2.1 reports the *Commitment's Life Cycle* [Sin99, For03, MYS03, MS05, Sin08, MBB+11c] of a generic commitment $\mathsf{C}(x, y, p)$: *arcs* represent the operation that can be performed on commitments and *nodes* represent the states of the life cycle a commitment can be in. As depicted, the activa-

Operation	*Effects*	*Description*
CREATE	CREATE(x, y, r, u) is performed by x and it causes $C(x, y, r, u)$ to hold in the state.	It creates a new commitment. It can be the consequence of the adoption of a role, or the effect of an action performed by an agent.
DELEGATE	DELEGATE(x, y, z, r, u) is performed by x, it revokes the commitment $C(x, y, r, u)$ and creates the commitment $C(z, y, r, u)$.	It attributes the role of debtor to another agent. Can be performed by the debtor of the commitment.
ASSIGN	ASSIGN(x, y, z, r, u) is performed by y, it revokes the commitment $C(x, y, r, u)$ and creates the commitment $C(x, z, r, u)$	It can be performed by the creditor of the commitment. Its effect is to assign the commitment to another creditor.
DISCHARGE	It revokes the commitment $C(x, y, r, u)$ and makes u hold.	It is performed when the consequent condition of a commitment is achieved.
DETACH	It revokes the commitment $C(x, y, r, u)$ and makes r and $C(x, y, \top, u)$ (represented as $C(x, y, u)$) to hold.	It is performed on conditional commitments when the antecedent condition becomes true. The conditional commitment stops to hold and a new commitment is created.
CANCEL	CANCEL(x, y, r, u) is performed by x and it revokes the commitment $C(x, y, r, u)$.	It is performed by the debtor and it revokes the commitment. The debtor of the commitment is not engaged to achieve the condition anymore.
RELEASE	RELEASE(x, y, r, u) is performed by y and it revokes the commitment $C(x, y, r, u)$.	It revokes the commitment. The operation is performed by the creditor of the commitment.

Table 2.1: Operations for commitment manipulation [Sin99, VS99, MS05, Cho09]. All operations, but detach, can be performed on conditional commitments or on base-level commitments (i.e. $C(x, y, \top, u)$).

tion of a commitment of the kind $C(x, y, p)$ can be the result of a CREATE or of a DETACH of a conditional commitment (i.e. the achievement of its antecedent condition). The commitment persists in that state until an operation is performed on it. Before the creation a commitment is "not active". When a CANCEL or a RELEASE is performed on an active commitment, this latter becomes, as before it was created, "not active". When the operation performed on an active commitment is the DISCHARGE, then the commitment is said to be "satisfied". More complex are the DELEGATE and the ASSIGN operations where the original commitment becomes "not active", and a new commitment is created. In case of a DELEGATE the new commitment has a new debtor, while in case of ASSIGN the commitment has a new creditor. Finally, when there is no way to satisfy an active commitment (for instance because the interaction is considered as finished and the commitment has not been satisfied) it is said to be "violated".

In the literature there are different proposals for commitment formalization. Some of these are based on an operational semantics [YS02a, CS03, WLH05, Sin07]. Other approaches rely on temporal logics. Among these the approach by Mallya *et al.* [MYS03] associate a temporal representation to commitments, allowing to detect commitment violation, satisfaction and so on. The approach by Giordano *et al.*[GMS07] uses DLTL, while Bentahar *et al.* [BMW09] and El-Menshawy *et al.*[EMBD10, EMBD11] adopt extensions of CTL* (named CTLC and CTL^{*sc}). In these approaches commitments are basically seen as modal operators.

2.2.1 Commitment Protocols

Commitments regulative nature

Commitments have been deeply investigated as a valuable way for interaction protocols specification (e.g. [YS04, WLH05, KY09]). The functioning of commitment-based specifications relies on the *regulative nature* of commitments. Commitments, indeed, capture conditions to be achieved. They affect the agents' behaviour because the underlying assumption is that once an agent takes a commitment then it will behave in such a way to fulfil it. The debtor of a commitment, indeed, is responsible for the engagement it has taken at least towards the creditor agent [CCD98]. If at the end of the interaction the commitment is not satisfied, then it is said to be violated. This aspect is known as the *regulative* nature of commitments. It can be used as a powerful means for realizing the coordination between the agents. From the debtor agent point of view, indeed, the possibility to violate a commitment and to be discovered provides it incentives to do the best it can to satisfy its commitments. The prediction that an agent will drive on a path that allows for the discharge of its commitments, may have some effects also on other agents (not only on the debtor) [Wei99, Chapter 2]. Consider, for instance, a customer that takes a commitment to pay for some goods. The expectation that it generates is that at the end it will pay or it will be punished. This expectation can be a sufficient guarantee for the merchant to send the goods (or to take a commitment to send them), even if it has not received the payment yet.

Commitment protocols define a set of axioms asserting which *physical events* count as which social events [MBB$^+$11c]. Physical events can be of a different nature, not necessarily corresponding to messages exchanged between the agents. Their execution allow agents to indirectly manipulate commitments (for instance to create a commitment, to satisfy or to cancel an existing one).

Example 2.2. *Considering a sale protocol. The actions* order, pay, ship_ goods *and such like are* physical events. *To each of these actions is associated a corresponding social event. By relying on the notation adopted in [Cho09] (that will be used in the rest of this thesis) it can be expressed in the following way (for simplicity we omit who can perform the action):*

(a) order ***means***
$\textsc{create}(\mathsf{C}(\mathsf{customer}, \mathsf{merchant}, \mathsf{C}(\mathsf{merchant}, \mathsf{customer}, \mathsf{ship}), \mathsf{pay}))$

(b) cancel ***means*** $\textsc{cancel}(\mathsf{C}(\mathsf{merchant}, \mathsf{customer}, \mathsf{ship})) \wedge$
$\textsc{release}(\mathsf{C}(\mathsf{customer}, \mathsf{merchant}, \mathsf{pay}))$

The above axioms state that the action order, *performed by a* customer, *has the meaning of creating the commitment from the customer to the merchant to pay for a certain item if the merchant has a commitment to send it. The meaning of the action* cancel, *performed by a* merchant, *is that the merchant cancels its commitment to ship the ordered item and it releases the customer from its commitment to pay.*

The possibility to manipulate commitments, though only indirectly (i.e. through physical events), facilitates the coordination between agents, allowing the exploration of different ways for achieving it. For instance, once an agent realizes that it is not able to fulfil a commitment it has taken, instead of interrupting the interaction the agent can explore other alternatives. For instance, it can create another commitment, delegate or cancel the commitment it cannot achieve or even release other agents of their engagements toward it (as in Example 2.2). Of course, since commitments can be cancelled, released, and so forth, actions should be defined in such a way to not result in arbitrary behaviours [Sin99]. The possibility of manipulating the commitments is one of the main differences between commitments protocols and deontic logics [TT98]. These latter, indeed, have not been conceived for interaction protocols specification, so they do not allow for such a manipulation at interaction time.

Commitment-based specifications have been widely investigated in the context of interaction and business protocol representation [YS01, Sin03b, CS06, MS06, CS08, Cho09, DCS09, Des09]. However, these approaches mainly focus on the flexibility of the agents' behaviour, disregarding the need of capturing *patterns* that the interaction is expected to respect. These patterns may express requirements such as an ordering on the achievement of different conditions, or as a mutual exclusion of some conditions. Their representation, however, requires a degree of expressiveness that commitment alone are not able to capture. This topic will be deepen in Chapter 3.

2.3 OTHER APPROACHES

Let us now briefly describe other approaches that are not based on commitments and that do not necessarily concern interaction protocols specification. These approaches are interesting because, even concerning different settings (like business process or e-learning), they address requirements on the specification that are similar to those on interaction protocols (like flexibility). They can be valid sources of inspiration. In particular, we consider Declare (Section 2.3.1) and DCML (Section 2.3.2), two graphical languages settled respectively in the context of business processes specification and pedagogical constraints specification. Both approaches face the problem of how to provide flexible specifications, while being easy to understand. In Section 2.3.3, we describe a proposal concerning an expectation-based approach for interaction protocol specification.

2.3.1 Declare

Declare is a workflow management system for business processes (declarative) modelling and execution [PSvdA07, Pes08, vdAPS09]. In this approach, which does not build on commitments nor is set in the agents framework, business processes are represented in a declarative way. The language that is proposed for business process representation (previously know as "ConDec" [PvdA06] and then renamed "Declare") is a graphical language grounded on Linear-time temporal logic. By means of this language it is possible to graphically specify a set of relations among activities, where an activity is seen as an atomic unit of work. An execution trace is said to be *supported* by a Declare model if and only if it complies with all the specified constraints.

Strengths of the proposal

This proposal can be of inspiration for interaction protocols specification, because it is possible to find some analogies between the two contexts. In particular, the aim of the language is to *support flexibility* in the business processes representation [vdAPS09]. To answer to this requirement the language is declarative, allowing to specify only the desired constraints, without imposing rigid sequences. Moreover, the choice of relying on a graphical representation allows to ease the specification of business processes and makes the language usable and understandable also to people that are not familiar with logics [PSvdA07].

2.3.2 Declarative Curricula Model Language

Declarative Curricula Model Language (DCML) [BBB+11b] has been defined for the e-learning context, to supply a way for expressing pedagogical requirements on curricula of study. It is a declarative language grounded on LTL, which allows to graphically specify relations between competences that are acquired along a curricula of study.

DCML is inspired, in its graphical representation, by ConDec (Later renamed "Declare"). Indeed, one of the aim that DCML shares with ConDec is the possibility of making the language actually accessible, usable and comprehensible by a great variety of people. This need derives from the aim of e-learning of reaching as many people as possible, thus without making assumptions on their capabilities, knowledge, level or kind of instruction. This is even more important with logic-based languages, where the support of a graphical representation of constraints makes the language more usable and understandable, thus not requiring the presence of an expert of logic. Being declarative, DCML allows for a flexible specification of the requirements. Moreover, it can be easily modified and adapted to different requirements. Basically, it is sufficient to specify only what is strictly necessary or forbidden. All that remains is left free. It makes also easier the composition of different requirements (by simply merging them) coming, for instance, from different institutions.

2.3.3 Expectations-Based Approaches

Social expectations represent an alternative to commitments for providing a social semantics to the interaction. The main difference between the two is that the former provides a *rule* oriented perspective of the interaction, while the latter adopts a *state* oriented perspective [TCY+09].

An expectation is a representation of a desired behaviour [ADT+04], given in terms of an event description and the time at which the event is expected to happen. An expectation may be positive or negative. Positive expectations have the form $\mathbf{E}(p,t)$ whose meaning is that the event p is expected to happen at time t. Negative expectations, instead, have the form $\mathbf{EN}(p,t)$ capturing that event p cannot happen at time t. Notice that, as a difference with commitments, expectations are not directed from a debtor to a creditor. Therefore, even if they have a normative nature, expectations are not associated to a notion of responsibility.

The expectation based approach is at the basis of the SCIFF logical framework [ACG+08], a framework that allows for interaction specification and verification. A SCIFF specification consists of an abductive logic program, where *Social Integrity Constraints* (SICs) are used to model relations among events and expectations about events [MTC+10]. By means of the integrity constraints it is possible to define *patterns of interactions* [TCY+09] by relating the time at which different events (may) happen. An example (taken from [TCY+09]) of such constraint is:

$$\begin{aligned}\mathbf{H}(p,\mathsf{T}) \rightarrow\ & \mathbf{E}(r,\mathsf{T}_1) \wedge \mathsf{T}_1 \leqslant \mathsf{T}+5 \\ & \vee\ \mathbf{EN}(s,\mathsf{T}_2)\end{aligned}$$

The meaning of this constraint is that if p happens at time T, then the event r is expected to happen at a certain time T_1 such that $\mathsf{T}_1 \leqslant \mathsf{T}+5$. If this is not the case, then the event s is expected not to happen at any time. The SCIFF framework provides a proof procedure able to monitor the evolution of the interaction and, given the happened events, compute the sets of expectations. The generative extension of the SCIFF proof procedure, named g-SCIFF [MTC+10] allows to statically verify properties on the specification and to generate counterexamples of properties that do not hold.

The expectation based semantics of SCIFF has been used, as an alternative to LTL, for providing a formal semantics to the graphical and declarative language ConDec [PvdA06]. Some proposals can be found in [CMMT09b, MPvdA+10]. Montali [Mon09] adopts a similar use of ConDec and he proposes an automatic translation of ConDec into the *CLIMB Language* (which is a subset of SCIFF).

3 THE CONSTRAINT AMONG COMMITMENT LANGUAGE

CONTENTS

In this chapter we present our proposal for interaction protocol specification [BBM10b, BBM11a, BBM10a, BBMP13]. Roughly, it is a commitment-based representation, enriched with the possibility of expressing patterns of interaction in a declarative and flexible way. In particular, we define a new language named 2CL (*Constraints among Commitments Language*) that allows the definition of constraints among facts and commitments. These constraints basically rule the achievement of different conditions, thus capturing a set of patterns the interaction is desired to respect. In order to facilitate constraints definition and their understanding, we also provide a graphical representation of the operators of the language.

The use of the language is explained by means of the Robert's Rules of Order example, concerning the regulation of democratic assemblies. At the end of the chapter we describe the specification of two other examples: the Contract Net and NetBill protocols. Discussions on the nature of constraints end the chapter.

3.1 EXTENDING THE REGULATIVE SPECIFICATION OF COMMITMENT PROTOCOLS

Commitment protocols represent a valuable specification of interactions protocols in MASs for some interesting characteristics that we have discussed in the previous chapter. In particular, they *do not overconstrain* the specification by imposing unnecessary orderings on the execution of the shared actions, and by giving a shared (i.e. public and agreed) meaning to the social actions, they allow working on actual knowledge rather than on beliefs about each others' mental state, thus preserving the agents' autonomy. Nonetheless, commitment protocols *do not yet suit well* those situations where the evolution of the interaction is constrained by conventions, laws, preferences, agreements or habits, because they do not allow the specification of *legal patterns* of interaction [BBM10a, BBMP13], although this kind of constraints makes sense in many practical situations, as noticed also in [SC09].

Regulative characterisation of commitments

The specification of patterns of interaction requires a degree of expressiveness that commitments alone do not have. In commitment protocols, indeed, the regulative aspect is captured only by commitments, allowing to capture only that some conditions must be achieved. Notice that also conditional commitments express only conditions to be achieved. They, indeed, do not impose a temporal ordering between the acquisition of the antecedent and consequent condition. Moreover, conditional commitments do not require the antecedent condition to become true sooner or later. This entails that there are no guarantee that the commitment will become active along the interaction.

Example 3.1. *Consider, for instance, the NetBill protocol as modelled in [YS02a]. Commitments allow for a flexible enactment of it, since they allow to start the interaction by an offer of the merchant without a request of the customer (as happens for advertising), or by starting with the merchant sending some goods for trial to a customer. This specification allows also the customer to accept a quote before the merchant proposes one, mimicking the trust of the customer on the fact that the merchant will make an offer. However, it is not always possible to assume that the interaction can be based on the trust the agents have one on the other. Agents indeed are heterogeneous and therefore there are no guarantees on how they are implemented (how to recognize malicious agents or misbehaviours?). So, on which basis the customer trusts the merchant? Even assuming that the customer has some reasons for trusting the merchant, what happens if this latter is willing but is practically unable of making the offer? In this case it is not clear what the customer has accepted.*

This example shows the need of expressing a pattern of interaction *(e.g. a temporal ordering) ruling the* quote *of the merchant and the* accept *of the* *customer. Commitments, however, are not expressive enough to capture these kinds of regulative requirements.*

Extending the regulative specification

Our proposal [BBM10a, BBMP13, MBB11b, MBB11a] is to extend the regulative specification of commitment protocols, so as to account for patterns of interactions and so as to maintain commitment protocols flexibility. The way we propose to express patterns is in terms of constraints among facts and commitments. We define 2CL: *Constraints among Commitments Language*, a declarative language that includes a set of operators capturing patterns frequently used in interaction protocols. The constraints that the language allows to represent can be of different kinds, some expressing temporal requirements on the achievement of different conditions, some other just capturing a relation among them. Intuitively, by considering the possible interactions given by the protocol actions, those that are compliant with the 2CL constraints can be said to be "legal" executions of the protocol.

Even capturing requirements on how the interaction should evolve, constraints allow for flexible behaviours of the interacting parties, along the intent of commitment protocols: everything that is not affected by constraints is left free to the agents. Temporal relations, for instance, express a relative order on the achievement of different conditions. Between their achievement other conditions can be achieved in between (if not differently specified by other constraints). Moreover, 2CL does not specify which actions should be executed to satisfy a constraint. *Any* action, whose effects are compatible with the schema of evolution captured by the constraints, is applicable. This respects the characteristic of commitment protocols, which do not specify which action to perform in order to satisfy a commitment.

THE ROBERT'S RULES OF ORDER PROTOCOL. The running example that we use in this Chapter to describe the effectiveness of the proposed specification and the need for a means for capturing patterns of interaction, is the *Robert's Rules of Order* (RONR for short [REHB00]), an interaction protocol actually used (see [PKSA06]). RONR is a well-known regulation of the behaviour to be followed by a democratic deliberative assembly, like the Parliament, to discuss and decide about issues called *motions*. This protocol is interesting because not only it specifies the social actions but it also encompasses rules that govern the behaviour of the assembly [BBM10a].

The RONR includes two roles, the chair ('ch' in the following) and a generic participant ('p' in the following). Briefly, the protocol states that before voting a motion, everybody who wishes to speak must have the possibility of speaking; in order to speak it is necessary to have the floor and it is up to the chair to decide to whom assigning the floor.

At the end, all the participants must have the possibility to vote the motion.

The RONR case study shows how in certain contexts the regulative specification of an interaction protocol is not just a guideline because it is fundamental in order to give guarantees to the participants. On the other hand, the protocol is flexible because, for instance, it does not specify a predefined number of motions to be discussed or the precise moment in which the participant must decide whether to discuss the motion or refuse to do it. In the following we show how the proposed specification allows to represent these requirements on the interaction, while maintaining the interaction as much flexible as possible.

3.2 2CL PROTOCOL SYNTAX

A 2CL protocol explicitly accounts for the patterns of interactions in the specification. They are represented as a set of 2CL constraints. Therefore, the resulting definition for an interaction protocol is the following:

Definition 3.1 (2CL Interaction protocol)**.** *An interaction protocol* $\mathcal{P}$ *is a tuple* $\langle \mathsf{Ro}, \mathsf{F}, \mathsf{s}_0, \mathsf{A}, \mathsf{Cst} \rangle$*, where* Ro *is a set of roles, identifying the interacting parties,* F *is a set of literals (including commitments) that can occur in the social state,* s_0 *is a set of literals that represents the content of the initial state,* A *is a set of actions, and* Cst *is a set of constraints.*

In the following we will use the term 2CL *protocol* (or 2CL protocol specification) referring to a protocol specification given according to the above definition. The set of *roles* Ro in the above definition captures a set of abstract entities that will be instantiated by agents participating to the interaction. The *initial state* s_0 expresses conditions holding at the beginning of the interaction (facts or commitments). The set of *literals* represents the domain vocabulary. It can be seen as the set of conditions on which the agents agree on the meaning and on which actions and constraints are defined. The set of social actions A, defined on F and on Ro, forms the *constitutive specification* of the protocol, while the set of constraints Cst, defined on F and on Ro too, forms the *regulative specification* of the protocol, that extends the regulative characterization of commitments. For the sake of simplicity, if not differently stated, from now on we will refer to regulative specification as the one given by constraints, but we still account for the one given by commitments.

The distinction on what has to be considered as constitutive and what is instead regulative derives from Searle's definition [Sea69, Sea95].

Constitutive rules identify certain behaviours as foundational of a certain type of activity (they create that activity). They do so by specifying the semantics of actions. Regulative rules, in contrast, contingently constrain a previously constituted activity. In other words, they rule the "flow of activities", by capturing some important characteristics of how things should be carried on in *specific contexts* of interaction [Che73].

Example 3.1. *To clarify the distinction between constitutive and regulative rules let us consider, as example, the discussion of a motion in a democratic assembly.* Constitutive rules *are to be used to define the social meaning of the basic actions, like the open of the debate, the discussion of the motion, the request of the floor, the expression of a vote. On top of these, it is possible to specify some desired behaviour as* regulative rules. *For instance, it is possible to state that participants are allowed to discuss a motion only if they have the floor; or that a participant cannot vote until all the other participants have discussed the motion.*

Constitutive Specification

The constitutive specification of the protocol is given as a set of actions representing the physical events that can be performed by the agents. To each action is associated at least one definition, basically capturing a *count as* relation [Sea95, AÁNDVS10] between physical and social events: the performance of a physical event affects the social state by performing the specified social events. Social events are meant to occur concurrently to the physical event.

In our specification the count-as relation is expressed by a *means* construct, capturing the meaning of performing a certain action in a certain context. The actions' definitions have the following form:

$$X\ \boldsymbol{means}\ Y\ \boldsymbol{if}\ Cond$$

Where X is a physical event, Y is a social event and $Cond$ represents the condition under which X acquires the specified meaning (see Definition 3.2).

Actions do not necessarily need to have the form of a communication or a message but they need to be observable actions [VS99]. Moreover, by playing a role in a protocol the agents agree on the meaning of the protocol actions. In this way, an action that is performed during the interaction can be understood by an observing agent. Agents are still free to perform actions which are not part of the specification but, in this case, their execution cannot be interpreted as part of the interaction. Therefore, actions that are out of the protocol do not have any effect on the social state. For instance, actions that are not part of the specification cannot be interpreted as the discharge of a commitment or as the creation of a new one. Roughly, since they do not correspond to any count as relation, agents are not able to *understand* them and to interpret them in terms of social events.

The social events that are used to give meaning to the physical events are the assertion of a fact in the social state, or the performance of an operation on a commitment. The operations that can be performed on commitments are CREATE, RELEASE, CANCEL, DISCHARGE, ASSIGN and DELEGATE [Sin99, YS01] (see Section 2.2). More formally, the Backus Naur form for the constitutive specification is the following:

Definition 3.2 (Constitutive Specification)**.**

$\mathsf{A} \rightarrow (\mathsf{Action}\ \textbf{means}\ \mathsf{Operation}\ \textbf{if}\ \mathsf{Cond})^{+}$

$\mathsf{Action} \rightarrow \mathsf{protocol\_action}([\mathsf{param\_list}])$

$\mathsf{Operation} \rightarrow \mathsf{Op}(\mathsf{commitment}) \mid \mathsf{fact} \mid \mathsf{Operation} \wedge \mathsf{Operation}$

$\mathsf{Op} \rightarrow \textsc{create} \mid \textsc{cancel} \mid \textsc{release} \mid \textsc{delegate} \mid \textsc{assign}$

$\mathsf{Cond} \rightarrow \mathsf{literal} \mid \neg\mathsf{literal} \mid \mathsf{Cond} \wedge \mathsf{Cond} \mid \mathsf{Cond} \vee \mathsf{Cond} \mid$
$\qquad \mathsf{Cond}\ \textsc{xor}\ \mathsf{Cond}$

"protocol_action" is the name of an interactive action of the protocol; "param_list" denotes the (potentially empty) parameter list of the action; "Cond" specifies the condition under which the effects of the action are applied to the state; "commitment" is a shortcut to represent $C(x, y, r, p)$ as specified in Section 2.2 (see also [Cho09, page 49]) where x and y are roles in Ro and r and p are formulas in disjunctive normal form or propositional literal in F; "fact" is a proposition that does not concern commitments and which contributes to the social state (they are the conditions that are brought about.); "Op" is one of the operations on commitments; and literal can be either a commitment or a proposition (where negation means that a certain literal does not hold in the social state).

Example 3.2 (RONR Constitutive Specification)**.** *For simplicity, in the specification of the RONR we will consider the interaction between the chair and one participant only. We now describe the constitutive specification of RONR that is reported in Table 3.1.*

In a democratic assembly a participant must have the possibility of voting on the presented motion. To this aim, by the action (a) 'motion'*, the chair commits to the participant to let it vote the motion. Notice that, the specification of the other actions of the protocol do not allow for this commitment to be cancelled or released. Therefore, the only possibility the chair has to not incur a violation of this engagement is to satisfy it, by means of the action (i)* 'cfv'*. The participant can exercise its right to vote by performing the action (l)* 'vote'*. However, the preconditions to both actions* 'cfv' *and* 'vote' *are that the motion should have been introduced before. The intuitive reason is that, in order to vote, the participant should have something to vote. The action* 'mo-

(a) motion **means** motion_introduced $\wedge$ CREATE(C(ch, p, cfv))
    if $\neg$motion_introduced
(b) open_debate **means** debate_opened$\wedge$
    CREATE(C(ch, p, C(p, ch, discussed), assign_floor))
    if motion_introduced
(c) refuse_floor **means** refused_floor$\wedge$
    RELEASE(C(ch, p, C(p, ch, discussed), assign_floor))$\wedge$
    RELEASE(C(ch, p, assign_floor))$\wedge$
    CANCEL(C(p, ch, discussed))
    if motion_introduced
(d) ask_floor **means** CREATE(C(p, ch, discussed)) $\wedge$ asked_floor
    if debate_opened
(e) recognition **means** assign_floor
    if debate_opened
(f) start_talk **means** talk_started
(g) stop_talk **means** talk_ended $\wedge$ discussed
    if talk_started
(h) time_out **means** talk_ended $\wedge$ discussed
    if talk_started $\wedge$ $\neg$talk_ended
(i) cfv **means** cfv
    if motion_introduced
(l) vote **means** voted
    if motion_introduced

Table 3.1: RONR Constitutive Specification.

tion', *instead, makes sense only if the motion has not been already introduced. In this way it is possible to model that different motions are discussed one at a time.*

For what concerns the discussion phase, the action (b) 'open_debate' *is performed by the chair in order to open the discussion. The fact that the discussion is referred to a certain motion is captured by the precondition* motion_introduced. *By means of this action the chair commits to assign the floor to the participant if the participant commits to discuss the motion (by means of the conditional commitment). The action (e)* 'recognition' *allows the chair to discharge its commitment by recognizing a participant and assigning it the floor. This meaning is achieved only if the debate is open, otherwise it is not clear for which purpose the chair is recognizing the participant (what it is expecting from the participant to do).*

The participant can ask for the floor by means of the action (d) 'ask_floor'. *This action allows it to declare its interest in discussing the motion and therefore creates the commitment from the participant to the chair to discuss it. This meaning is conditioned by the fact that the debate should have been already opened. The action (c)* 'refuse_floor' *has, intuitively, the opposite meaning, that is to say that the participant is not interested in discussing the current motion. It can be performed by the participant in order to release the chair from its commitments (both conditional and base) to assign it the floor for the discussion. In order to acquire this meaning, however, the motion should* *have been introduced before. Intuitively, indeed, the participant cannot refuse to discuss something that has not been presented yet. By not requiring for the debate to be open we allow a participant to state immediately (without waiting for the debate to be opened) that it is not willing to discuss the motion.*

A commitment of the participant to discuss the motion is discharged when the action (g) 'stop_talk' *is performed. It captures that the participant has finished to discuss the motion and it makes sense only if the participant has started to discuss it. This latter condition is achieved when performing the action (f)* 'start_talk'. *In case the amount of time that a participant has for discussing a motion runs out, then the chair can interrupt a participant that is speaking, by means of the action (h)* 'time_out'. *Notice that this action has exactly the same meaning of* 'stop_talk', *but it is performed by the chair.*

Regulative Specification

The language that we propose for patterns specification is named 2CL, acronym that stands for ***C****onstraints among* ***C****ommitments* ***L****anguage*. As the name suggests, the conditions involved in the constraints are given in terms of literals (facts and commitments) rather than directly on actions.

The language allows the designer to express many kinds of constraints describing the legal evolutions of the social state. As underlined in [Mon09, BBB+11b], constraint-based declarative representations provide abstractions which allow to explicitly capture what is mandatory and what is forbidden, without the need of expressing the set of possible executions extensionally. 2CL constraints can be defined according to the following Backus Naur form:

Definition 3.3 (2CL Regulative Specification)**.**
Cst → (Disj op Disj)*
Disj → Conj OR Disj | Conj XOR Disj | Conj
Conj → literal AND Conj | literal

Constraints are defined in terms of disjunctive normal formulas. A disjunctive normal formula *dnf* is a disjunction of conjunctive formulas of literals (see [BK08]). A *literal* can be either a positive or negative commitment or a positive or negative fact. Negation means that a certain literal does not hold in the social state. op stands for a 2CL *operator*. The set of 2CL operators, together with their intuitive meanings, is reported in Table 3.2 and it is discussed in the details in the reminder of this section. Basically, operators express relations between two conditions, saying what should become true in the social state and, in case of a temporal operator, when. Each operator has a positive and a negative form. The operators that are part of the language have been defined by considering the kind of constraints frequently used in the examples from the literature (like the Contract Net Protocol or the Net-Bill protocol, Section 3.4) and in real case studies (as those described in Chapter 7). However, the modularity of the language allows for the introduction of new operators in case new needs, not tackled by the current language, arise.

3.2.1 2CL Operators

In order to better give the intuition behind the operators of the language, we have divided them into three groups according to the (temporal) requirement they capture on the interaction. The groups that we have identified are: *(i) Relation operators*, that capture occurrence and co-occurrence between two conditions, without capturing any temporal relation among them; *(ii) Temporal operators*, which capture a relative temporal ordering between two conditions; *(iii) Strong sequence*, that includes the premise operator that imposes the strongest temporal ordering that we foresee in the language.

For simplicity, in the following we represent constraints as "dnf_1 op dnf_2", where dnf_1 and dnf_2 are conditions expressed as disjunctive normal form formulas of literals (see Definition 3.3) and op is a 2CL operator.

Relation Operators

Intuitively, the aim of this kind of operators is to express conditions of the kind "*if-then*" but without any temporal ordering between the conditions: if a certain condition holds then, no matter when, another condition must be achieved. The operators that are part of this group are the *correlation* and the *coexistence*.

CORRELATION

$$\mathrm{dnf}_1 \textit{ correlate } \mathrm{dnf}_2. \tag{3.1}$$

Positive Correlation

The correlation operator is used to express that if the condition dnf_1 is achieved in an execution, then also the condition dnf_2 must become true sooner or later. In a certain sense we can say that the constraint is triggered when the condition dnf_1 is achieved (if it never holds the truth value of dnf_2 does not matter). Therefore, in order for the execution to satisfy the constraint, condition dnf_2 can become true after, in the same state or before the state in which condition dnf_1 becomes true.

Example 3.3. *For instance, in the RONR we can define the constraint*

	Operator	*Meaning*
Relation Operators	$\mathtt{dnf}_1$ *correlate* $\mathtt{dnf}_2$	In an execution where $\mathtt{dnf}_1$ occurs, also $\mathtt{dnf}_2$ must occur but there is no temporal relation between the two.
	$\mathtt{dnf}_1$ *not correlate* $\mathtt{dnf}_2$	If $\mathtt{dnf}_1$ occurs in some execution, $\mathtt{dnf}_2$ must not occur.
	$\mathtt{dnf}_1$ *co-exist* $\mathtt{dnf}_2$	It captures the mutual correlation $\mathtt{dnf}_1$ *correlate* $\mathtt{dnf}_2$ and $\mathtt{dnf}_2$ *correlate* $\mathtt{dnf}_1$.
	$\mathtt{dnf}_1$ *not co-exist* $\mathtt{dnf}_2$	This captures the mutual exclusion of $\mathtt{dnf}_1$ and $\mathtt{dnf}_2$: both $\mathtt{dnf}_1$ *not correlate* $\mathtt{dnf}_2$ and $\mathtt{dnf}_2$ *not correlate* $\mathtt{dnf}_1$ hold.
Temporal Operators	$\mathtt{dnf}_1$ *response* $\mathtt{dnf}_2$	If $\mathtt{dnf}_1$ occurs, $\mathtt{dnf}_2$ must hold at least once afterwards (or in the same state). It does not matter if $\mathtt{dnf}_2$ already held before $\mathtt{dnf}_1$.
	$\mathtt{dnf}_1$ *not response* $\mathtt{dnf}_2$	If $\mathtt{dnf}_1$ holds, $\mathtt{dnf}_2$ cannot hold in the same state or after.
	$\mathtt{dnf}_1$ *before* $\mathtt{dnf}_2$	$\mathtt{dnf}_2$ cannot hold until $\mathtt{dnf}_1$ becomes true. Afterwards, it is not necessary that $\mathtt{dnf}_2$ becomes true.
	$\mathtt{dnf}_1$ *not before* $\mathtt{dnf}_2$	In case $\mathtt{dnf}_2$ becomes true, $\mathtt{dnf}_1$ cannot hold beforehand.
	$\mathtt{dnf}_1$ *cause* $\mathtt{dnf}_2$	It is the conjunction of *response* and *before* relations: $\mathtt{dnf}_1$ *response* $\mathtt{dnf}_2$ and $\mathtt{dnf}_1$ *before* $\mathtt{dnf}_2$.
	$\mathtt{dnf}_1$ *not cause* $\mathtt{dnf}_2$	It is the conjunction of *response* and *before* negative relations: $\mathtt{dnf}_1$ *not response* $\mathtt{dnf}_2$ and $\mathtt{dnf}_1$ *not before* $\mathtt{dnf}_2$.
Strong Sequence	$\mathtt{dnf}_1$ *premise* $\mathtt{dnf}_2$	$\mathtt{dnf}_1$ must hold in the state immediately preceding one state in which $\mathtt{dnf}_2$ holds.
	$\mathtt{dnf}_1$ *not premise* $\mathtt{dnf}_2$	$\mathtt{dnf}_1$ must never hold in a state that immediately precedes one where $\mathtt{dnf}_2$ holds.

Table 3.2: 2CL operators and their meaning.

motion_introduced *correlate* debate_opened

so as to require that in every execution in which the motion is introduced then also the debate must be opened. However, by this constraint, we are not requiring a temporal ordering between the achievement of these two conditions. Therefore, the only way to violate this constraint is to introduce the motion but do not open the debate (neither before nor after).

Negative Correlation Constraints using the negative form of correlation can be expressed in the following way:

$$\mathtt{dnf}_1 \textit{ not correlate } \mathtt{dnf}_2 \qquad (3.2)$$

Also the negated form of correlation captures a relation without temporal requirements. In particular, it requires that when the condition $\mathtt{dnf}_1$ holds in some state of the execution, the condition $\mathtt{dnf}_2$ should never hold in the whole execution. Also in this case we can say that the triggering condition is $\mathtt{dnf}_1$. If it never holds then it does not matter whether $\mathtt{dnf}_2$ becomes true or not.

Example 3.4. *In the RONR example the not correlate constraint could be used to require that in every execution in which a participant refuses the floor for discussing the motion he cannot speak. The corresponding constraint is:*

refused_floor *not correlate* talk_started

CO-EXISTENCE

$$\mathtt{dnf}_1 \textit{ co-exist } \mathtt{dnf}_2 \tag{3.3}$$

Positive Co-existence

The co-existence operator is derived from the correlation operator. In particular, "$\mathtt{dnf}_1$ *co-exist* $\mathtt{dnf}_2$" equals to "$\mathtt{dnf}_1$ *correlate* $\mathtt{dnf}_2$" and "$\mathtt{dnf}_2$ *correlate* $\mathtt{dnf}_1$". The aim of the constraint is to capture the mutual correlation between the two conditions in the constraint: if one of the two holds at a certain point of the execution, also the other must be achieved in the execution. An execution in which none of the two is achieved, however, is allowed.

Example 3.5. *Suppose that in a debate of a motion the vote of a participant should be supported by its discussion. No matter which of the two occurs first but once one of the two conditions is met, also the other must become true. This condition is captured by the constraint*

voted *co-exist* discussed

Notice that the constraint allows the situation in which none of the two conditions is met. Therefore, the participant is allowed to neither vote nor discuss the motion.

Negative Co-existence

Constraints using the negative form of co-existence can be expressed in the following way:

$$\mathtt{dnf}_1 \textit{ not co-exist } \mathtt{dnf}_2 \tag{3.4}$$

It expresses that two conditions cannot both hold in the same execution, no matter which is achieved first. This operator is derived from the negative correlation: "$\mathtt{dnf}_1$ *not co-exist* $\mathtt{dnf}_2$" equals to "$\mathtt{dnf}_1$ *not correlate* $\mathtt{dnf}_2$" and "$\mathtt{dnf}_2$ *not correlate* $\mathtt{dnf}_1$".

Example 3.6. *In order to require that a participant cannot refuse the floor and discuss the motion in the same execution we can specify the following constraint:*

discussed *not co-exist* refused_floor

Temporal Operators

The aim of these operators is to capture a relative ordering on the achievement of different conditions. Notice that we are not referring to time explicitly (for instance we are not facing the problem of expressing conditions that must be achieved within a certain amount of time from another condition).

The operators that are part of this group are the *response*, *before* and *cause*.

RESPONSE

$$\mathtt{dnf}_1 \textit{ response } \mathtt{dnf}_2 \tag{3.5}$$

Positive Response

The response operator aims at capturing that if a certain condition is achieved in an execution, then another condition must be achieved later. The triggering condition is $\mathtt{dnf}_1$: when it becomes true, then, sometime in the future, $\mathtt{dnf}_2$ must be achieved. If $\mathtt{dnf}_2$ has already been true in the past, the constraint requires that it becomes true again in the future. This temporal requirement is the main difference with the correlation operator. If $\mathtt{dnf}_1$ never holds in an execution the truth value of $\mathtt{dnf}_2$ is not influenced by this constraint.

Example 3.7. *If in the RONR example we want to express that once the motion has been introduced then the chair must open the debate later, the correlation operator that we have used in Example 3.3 is not sufficient. We can rewrite it by means of the response operator in the following way:*

motion_introduced *response* debate_opened

Therefore, when the fact `motion_introduced` *becomes true in the execution, there must exists a subsequent point in which* `debate_opened` *is true.*

Negative Response

Constraints using the negative form of response can be expressed in the following way:

$$\mathtt{dnf}_1 \textit{ not response } \mathtt{dnf}_2 \tag{3.6}$$

By this operator when a certain condition is met another one must be invariably false from that point on: if $\mathtt{dnf}_1$ becomes true, then $\mathtt{dnf}_2$ cannot hold after. As well as response, also in the negative form the triggering condition is $\mathtt{dnf}_1$. Therefore, if it never holds then the truth value of $\mathtt{dnf}_2$ is not affected by the constraint. Otherwise $\mathtt{dnf}_2$ can never hold after $\mathtt{dnf}_1$.

Example 3.8. *The negative response can be used to require that once a participant has refused the floor he/she cannot change his/her mind after and decide to commit to discuss the motion. In a certain sense, it has already taken the decision to do not take part to the debate. The constraint that expresses this condition is*

$$\text{refused\_floor } \textit{not response } \mathsf{C}(\mathsf{p}, \mathsf{ch}, \mathsf{discussed})$$

BEFORE

$$\mathsf{dnf}_1 \textit{ before } \mathsf{dnf}_2 \tag{3.7}$$

Positive Before

The *before* operator captures that if a certain condition is achieved, then another one should have been true, at least once, before. Specifically, if dnf_2 is met in order for the constraint not to be violated then dnf_1 should have been true before. However, if dnf_1 becomes true there are no constraints on the truth value of dnf_2 that can become true after or still be invariably false.

The main difference between *before* and *response* is the triggering condition that is one the opposite of the other: in *before* if dnf_2 becomes true, it requires dnf_1 to have been true in the *past*; in *response*, if dnf_1 becomes true it requires that dnf_2 becomes true *later*.

Example 3.9. *The before operator is one of the most frequently used in the RONR example. Let us comment some constraints.*
(1) *Before the participant can vote it must wait for the call for vote.*

$$\text{cfv } \textit{before } \text{voted}$$

Notice that, when cfv *becomes true the participant is not obliged to vote, it is just allowed to do it.*
(2) *Before the call for vote can be done, the participant must have discussed the motion or refused the floor. The aim is to guarantee to the participant the possibility to speak. As well as before, by this constraint the chair is not obliged to make the call for vote.*

$$\text{refused\_floor or discussed } \textit{before } \text{cfv}$$

Negative Before

Constraints using the negative form of before can be expressed in the following way:

$$\mathsf{dnf}_1 \textit{ not before } \mathsf{dnf}_2 \tag{3.8}$$

This operator expresses that if dnf_2 is met, then dnf_1 cannot be achieved before. However, it is possible for dnf_1 to become true after.

Example 3.10. *For instance, in the RONR example suppose that in order for a participant to state that it is not interested in discussing a motion, it* *must wait for the debate to be opened. This requirement can be captured by the constraint:*

refused_floor *not before* debate_opened

CAUSE

$$\mathrm{dnf}_1 \ \textit{cause}\ \mathrm{dnf}_2 \tag{3.9}$$

Positive Cause

Cause constraint captures a form of *causality* between conditions dnf_1 and condition dnf_2 of the kind foreseen in [Lam78]. It is derived from the conjunction of a *response* and a *before* constraint. In particular, the requirement that it expresses is that if condition dnf_1 is met, then condition dnf_2 should be achieved at least once later (*response* operator), and it is not allowed for dnf_2 to become true before dnf_1 (*before* operator).

Example 3.11. *Before a participant can start talking it should have obtained the floor and once it is assigned of it, the participant must talk (i.e. discuss the motion).*

assign_floor *cause* talk_started

To better grasp the difference with respect to 'response' and 'before' let us comment the meaning of similar constraints where 'cause' is substituted with a 'before' and a 'response'. If we substitute the 'cause' operator with a 'before', the requirement is that a participant cannot start talking if it has not previously obtained the floor. However, once it has the floor it is not obliged to talk. By using a response, instead, the participant can start talking even before it is assigned of the floor. However, when the initiator assigns the floor, the participant is obliged to talk at least once after.

Negative Cause

Constraints using the negative form of cause can be expressed in the following way:

$$\mathrm{dnf}_1 \ \textit{not cause}\ \mathrm{dnf}_2 \tag{3.10}$$

It is a derived operator that is obtained from the conjunction of the negated forms of *before* and *response*. "dnf_1 *not cause* dnf_2" equals to "dnf_1 *not response* dnf_2" and "dnf_1 *not before* dnf_2". Therefore, the effect it has on the interaction can be deduced by applying both constraints on the same execution: if dnf_1 is achieved in the execution, then dnf_2 cannot be achieved after (*not response*) and if dnf_2 is met then dnf_1 cannot hold before (*not before*). Intuitively, the only ordering it allows is that dnf_2 is achieved before dnf_1.

Example 3.12. *Consider the following constraint in the context of RONR example.*

vote *not cause* cfv

*To better understand it let us decompose the constraint in its before and response operators: "*vote *not cause* cfv*" equals to "*vote *not before* cfv*" and "*vote *not response* cfv*". The not before component requires that if the call for vote is made, then the participant should not have already voted. The not response component, instead, requires that if the participant has voted then the chair cannot make the call for votes. By combining the two we have that the only execution allowed is the one in which the chair opens the session for votes (*cfv*) and the participant can decide whether to vote or not, but in case he/she does, the chair cannot make another call for votes.*

Strong Sequence Operator

This group is composed by only one operator: the *premise*. It is a "strong sequence operator" in the sense that it requires that a certain condition is true in the state before the one in which another condition is met. This is a stronger requirement with respect to the other temporal operators. They, indeed, only capture a relative order on the acquisition of some conditions but do not specify the "distance" (in terms of states, transitions, or steps) in which they must be achieved.

Positive Premise

A *premise* constraint has the form:

$$\mathrm{dnf}_1 \textit{ premise } \mathrm{dnf}_2 \qquad (3.11)$$

Its meaning is that if dnf_2 holds at a certain point in the execution, then dnf_1 must be true in the state before.

Example 3.13. *In order for the constraint*

assign_floor *premise* talk_started

to be satisfied, the participant can start talking only if it has the floor in the precedent state. This is to be sure that, in the time that passes between the assignment of the floor and the beginning of the discussion, the chair does not revoke the right to speak to the participant.

Negative Premise

Constraints using the negative form of premise can be expressed in the following way:

$$\mathrm{dnf}_1 \textit{ not premise } \mathrm{dnf}_2 \qquad (3.12)$$

This operator captures the condition that dnf_1 can never hold in a state that immediately precedes one where dnf_2 holds.

Example 3.14. *In order to capture the requirement that the participant cannot commit to discuss the motion and immediately change its mind by re-*

(c1)	motion_introduced *response* (debate_opened OR refused_floor)
(c2)	refused_floor *not response* C(p, ch, discussed)
(c3)	discussed *not co-exist* refused_floor
(c4)	C(p, ch, discussed) *before* assign_floor
(c5)	assign_floor *cause* talk_started
(c6)	talk_started *correlate* talk_ended
(c7)	refused_floor OR discussed *before* cfv
(c8)	cfv *before* voted

Table 3.3: RONR Regulative Specification.

fusing to talk it is possible to use the not premise operator in the following way:

$$\mathsf{C(p, ch, discussed)}\ \textit{not premise}\ \text{refused\_floor}$$

To be satisfied, it should not be the case that in the current state the participant has a commitment to discuss the motion and in the subsequent state the participant has refused to discuss.

Example 3.15 (RONR Regulative Specification). *While describing the operators of the language we have used the RONR to give the intuition on how the operators can be used to capture different requirements on the interactions. We now present the set of constraints that we have chosen as representative of the RONR protocol. They are reported in Table 3.3. Let us now enter into the details of them.*

(c1) *is an example of use of the response operator: it states that if the motion is introduced, then at least once after, one condition between* debate_opened *and* refused_floor *must be achieved. Intuitively, it requires that if the initiator presents a motion then it should also give the participant the possibility to discuss it (by opening the debate), or the participant should states its desire to not discuss the motion.*

(c2) *is a not response constraint: it states that if a participant refuses to discuss a motion, it cannot change its mind after and commit to discuss it.*

(c3) *is defined by means of a not co-exist operator, which state that interactions in which a participant has both discussed a motion and refused to do it, are not allowed (independently of which of the two conditions is achieved first).*

(c4) *is of kind before: it states that before assigning the floor to a participant, this latter should have taken a commitment to discuss the motion. In this case we could have used a cause constraint, obliging the initiator to assign the floor to a participant that wants to discuss the motion. However, this is not necessary since by performing the action* open_debate*, the chair already has a commitment to assign a floor to a participant if there is the commitment from this latter to discuss (see action (b) of Table 3.1). The aspect missing to the commitment is the temporal requirement, that we can capture by means of this constraint.*

According to the cause operator of constraint (c5)*, before speaking the participant must obtain the floor and once obtained it is expected to speak.*

(c6) *is a correlate constraint. It states that when a participant starts talking it is expected also to end its discussion. Notice that constraints do not specify by means of which action this condition must be achieved. Therefore, according to RONR constitutive specification, it could be either by a* stop_talk *or by a* time_out*.*

Constraints (c7) *and* (c8) *are both of the kind* before*.* (c7) *requires that before the call for vote, the participant must have discussed the motion or refused to do it. The aim is to guarantee to the participant the possibility of speaking. Constraint* (c8) *requires that the participant waits the call for vote before it can vote. Notice that this constraint allows the participant to abstain from voting. If this is not the desired behaviour it is sufficient to substitute the before constraint with a cause.*

3.3 THE GRAPHICAL LANGUAGE

One of the advantages in the adoption of declarative approaches is that they allow to specify what is strictly necessary, rather than having to enumerate all the allowed interactions. This makes the specification more flexible and easier to modify (e.g. by adding or modifying constraints). However, declarative approaches have been often accused of being too difficult to understand and, consequently, to be used in a profitable way [Sin07, MM08]. In particular, in declarative protocol specifications the difficulty is given by the need of an interacting party to understand which behaviours are allowed, and by the need of the designer to specify the right set of constraints that allows him/her to obtain the desired behaviour from the system. More precisely, it is difficult to imagine the flow of execution defined by constraints by simply looking at them. A 2CL constraint, indeed, does *not* give any *procedure* that allows to satisfy it. More generally, constraints that define the legal evolutions of the social state are independent from the actions that are used by the agents. Moreover, constraints do not necessarily need to be satisfied in the next state from the one in which they are activated. In fact, constraints do not specify a rigid flow of action execution but rather, following the motto *no flow in flow*, they rule the evolution of the social state by expressing only what is mandatory and what is forbidden.

In order to support the designer and the interacting parties in the use of the declarative specification of protocol constraints, we have defined a graphical representation for 2CL constraints. By means of it, the designer can express easily different kinds of requirements on the

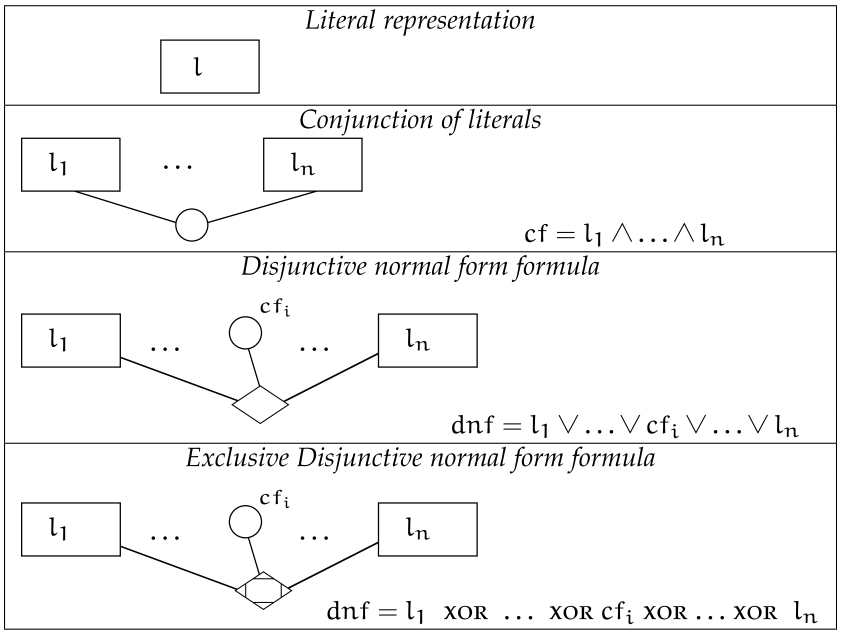

Literal representation
l
Conjunction of literals
l_1
...
l_n
$cf = l_1 \wedge \ldots \wedge l_n$
Disjunctive normal form formula
cf_i
l_1
...
...
l_n
$dnf = l_1 \vee \ldots \vee cf_i \vee \ldots \vee l_n$
Exclusive Disjunctive normal form formula
cf_i
l_1
...
...
l_n
$dnf = l_1 \text{ XOR } \ldots \text{ XOR } cf_i \text{ XOR } \ldots \text{ XOR } l_n$

Figure 3.1: Graphical elements for the representation of the graph of 2CL constraints. 2CL constraints can connect simple boxes (i.e. literals) as well as conjunctions and (exclusive) disjunctions of them, so as to represent more complex constraints.

interaction, by simply drawing them. From this graph it is possible to have a global view of constraints and to perceive the *flow of execution*. In this way a designer or a participant can rapidly have an intuition of the allowed evolutions of the interaction.

The graph of constraints associated to a regulative specification aims at supplying a global view of the requirements that constraints impose on the interaction. 2CL constraints allow the specification of relations between DNF formulas. Let us now describe how it is possible to specify them graphically.

Basic Elements Representation

Figure 3.1 summarises the elements that can be used in the graphical representation. As shown, a *literal* (commitment or fact) involved in a constraint is represented as a rectangle. A *conjunction* is represented by a circle, having as incoming arrows the literals involved in the conjunction. A *disjunction* is represented as a diamond. Since it can be composed of basic literals as well as conjuncts of literals, a diamond can have incoming arrows both from simple boxes and from circles (conjunctions). The representation of an *exclusive disjunction* is the same of the disjunction except for the fact that it is represented as a bordered diamond.

3.3.1 Graphical Representation of 2CL Operators

Let us now describe the graphical representation that we propose for 2CL operators. Each operator connects two DNF formulas. Their representation respects a few conventions: *(i)* the position of the dot intuitively denotes the occurrence of which condition activates the relation expressed by the constraint (what we have called "triggering conditions" in Section 3.2); *(ii)* arrows, when used, and their relative positions with respect to dots, denote the qualitative temporal nature of the constraint; *(iii)* negated operators are crossed by a line.

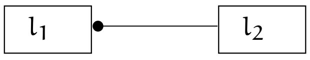
The above conventions are inspired by ConDec [PvdA06] and by DCML [BBB+11b], but the semantics of the operators is different. Moreover, also the elements involved in the constraints are different: ConDec operators relate activities and DCML concerns competences (knowledges in an e-learning context).

Let us now enter into the details of the representation by describing the symbol (reported in Table 3.4) associated to each operator. For simplicity, we report the representation for constraints involving simple literals only. The same relation can be drawn involving conjunctions or (exclusive) disjunctions.

Relation Operators

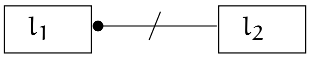
These constraints are represented by lines, without arrows. This is because these operators do not capture any kind of temporal ordering.

CORRELATION. This operator is used to express a relation on the achievement of two conditions. It is represented in the following way:



Specifically, the dot is closer to the left-hand side condition (l_1) and this represents the fact that the constraint is triggered when the condition on that side becomes true. In this case, the execution is requested to contain some state in which also the condition on the right-hand side becomes true. The absence of arrows denotes that there are no temporal requirements on the achievement of the two.

The representation of the negative correlation follows the same intuition. Additionally there is a crossed line that captures the negation:



CO-EXISTENCE. It is very similar to the correlation but, since both conditions can trigger the constraint, it is represented by two dots at the two sides of the relation.

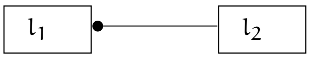
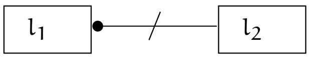
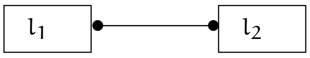
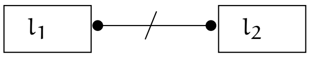
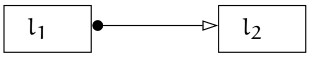
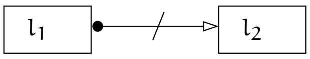
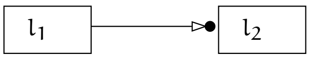
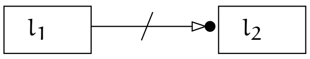
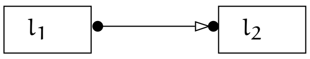
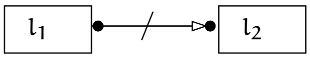
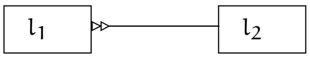
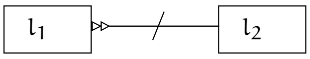
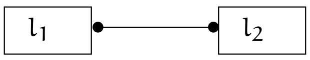
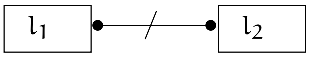
	Constraint	*Representation*
Correlation	l_1 *correlate* l_2	
	l_1 *not correlate* l_2	
Co-existence	l_1 *co-exist* l_2	
	l_1 *not co-exist* l_2	
Response	l_1 *response* l_2	
	l_1 *not response* l_2	
Before	l_1 *before* l_2	
	l_1 *not before* l_2	
Cause	l_1 *cause* l_2	
	l_1 *not cause* l_2	
Premise	l_1 *premise* l_2	
	l_1 *not premise* l_2	

Table 3.4: Graphical representation of 2CL operators.



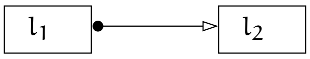
As before the negated co-existence is obtained by adding a crossing line on the relation, as shown below.



Temporal Operators

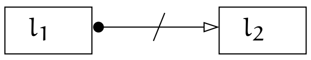
The temporal aspect of constraints is captured by an arrows that indicates the order in which conditions should be achieved.

RESPONSE. The graphical representation for the *response* is:

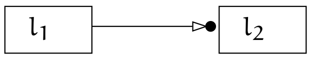


The arrow denote that the order to be followed is l_1 and then l_2. The dot specifies that the triggering condition is on l_1. Indeed, the meaning to be expressed is that if l_1 becomes true, then also l_2 must hold later.

The negative response is represented in the following way:

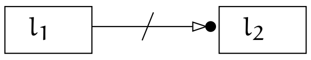


BEFORE. The graphical representation for the *before* is:

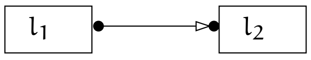


The *before* captures a temporal ordering between two conditions and this is expressed by the arrow. Similarly to *response*, also the before has a triggering condition but, in this case it is the condition at the right-hand side of the arrow. This is captured by drawing a dot at the rightmost end of the arrow.

The negative before is represented in the following way:

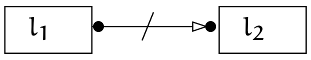


CAUSE. The *cause* operator is derived from response and before. This is confirmed also graphically. Indeed, the graphical representation of *cause* is obtained by laying the graphical representations of *response* and *before*, one upon the other. The result is the following:



The symbol for this constraints has two dots and one arrow. As usual, the arrow captures the temporal order imposed by the constraint, while the dots capture that both conditions can trigger the verification of the constraint. This is due to the fact that in *cause* constraints if the left-hand side condition is achieved also the other one must be achieved later, and if this latter is achieved, the former must be achieved before.

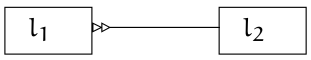
The representation for the negative form of this constraint is



Strong Sequence Operator

The Strong sequence captures a temporal constraint which expresses a condition on two subsequent states.

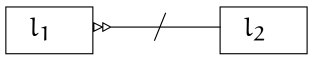
PREMISE. The direction expressed by the arrows, according to the convention adopted for the other operators, indicates the condition to be achieved after. This constraint is graphically expressed by two arrows at the leftmost hand (capturing that the temporal relation expressed by this constraint is stronger).



(c1)	motion_introduced •—▷ (debate_opened OR refused_floor)
(c2)	refused_floor •–/–▷ C(p, ch, discussed)
(c3)	discussed •/• refused_floor
(c4)	C(p, ch, discussed) —▷• assign_floor
(c5)	assign_floor •—▷• talk_started
(c6)	talk_started •— talk_ended
(c7)	refused_floor OR discussed —▷• cfv
(c8)	cfv —▷• voted

Table 3.5: RONR Graphical representation of the constraints.

The negative form of premise is represented as



Example 3.1. *Table 3.5 reports the graphical representation of the RONR constraints (given in Section 3.2.1 and summarized in Table 3.3).*

The RONR constraints can be composed so as to obtain the graph reported in Figure 3.2. Once the chair introduced the motion (node n1*), there are two possible evolutions of the interaction (due to the disjunction represented by node* n4*): the chair can open the debate (node* n2*), or the participant can declare that it is not willing to discuss the motion (node* n3*). Only after the participant has refused the floor or has discussed the motion (constraint* c7*), the initiator is allowed to open the vote session (node* n11*). The participant must wait for it before voting (node* n12*, constraint* c8*).*

*In any case, the participant cannot both refuse the floor and discuss the motion (constraint (*c3*)). If it decides to refuse it, it is not even allowed to change its mind later (constraint* c2*) by stating its intention to discuss the motion (node* n5*). The initiator cannot assign the floor (node* n7*) if the participant has not committed to discuss the motion before (constraint* c4*). In case the floor is assigned, then, and only after (constraint* c5*), the talk is expected to start (node* n8*). Every time a talk starts, sooner or later (constraint* c6*) the participant or the chair, should declare its end (node* n9*).*

3.4 EXAMPLES

In this section we present a 2CL representation for Contract Net (CNET) [Fou02c, SD78] and NetBill [ST95, CTS95] protocols. These examples are well known in the literature and they have been often used by different authors as case studies, e.g. [YS02a, WLH05, GMS07, Sin07]. We will use them as a means for describing the potentialities of our proposal.

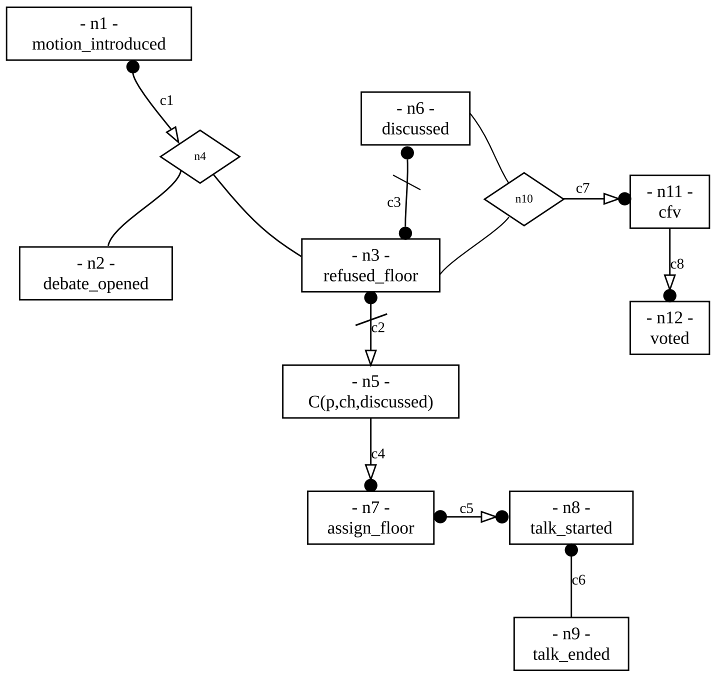


Figure 3.2: The Graph of RONR Constraints.

3.4.1 The NetBill Protocol

The aim of NetBill Protocol is to rule the purchase of *electronic information goods* (simply goods, in the following) over a network. The original version, as proposed in [CTS95, ST95] and represented in Figure 3.3, involves three parties: a customer, a merchant and a NetBill transaction server. This latter has been introduced for security reasons, in order to verify the payment of the customer. With the aim of describing the use of 2CL protocols specification, we consider a simplified version of NetBill protocol. Specifically, we consider interactions between a customer c, wishing to buy some information, and a merchant m. Intuitively, (1) the customer requests the price of certain goods to the merchant, (2) the merchant answers by quoting the goods, (3) the customer can either accept or reject the quote, (4) if the customer accepts, the merchant sends the requested information goods, (5) the customer pays the merchant, (6) the merchant sends the receipt of the payment to the customer. This simplified version is actually frequently used also by other authors (e.g. [YS02a, WLH05, YS02b]).

Constitutive Specification of NetBill

The set of actions of NetBill protocol and their definitions is reported in Table 3.6. Let us now comment on each of them.

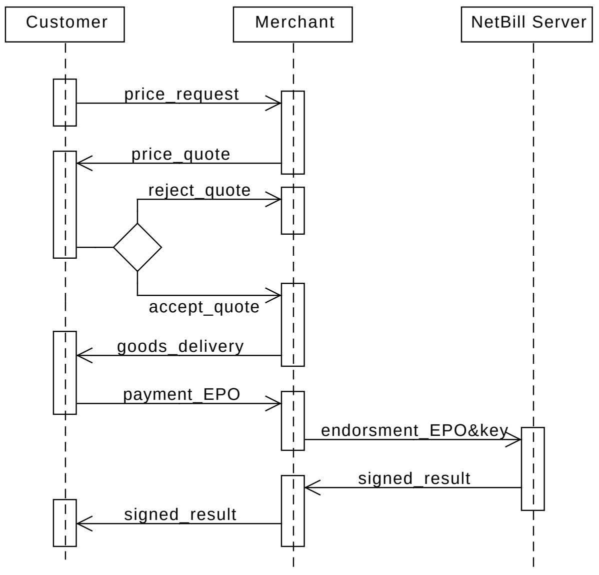


Figure 3.3: Sequence of messages exchanged in the NetBill Protocol.

SEND REQUEST *(Table 3.6(a))*. By means of this action the customer requests the price for certain goods. However, this action makes sense only if the merchant has not already provide a quote and if goods have not been sent already. In the first case, indeed, the customer already knows the price, so it is not clear why asking for it again. In the second case, instead, the customer has already received the goods.

SEND QUOTE *(Table 3.6(b))*. By means of this action the merchant communicates the quote for certain goods to the customer. It represents more than just an information addressed to the customer: it represents an engagement of the merchant to send the goods in case the customer commits to pay for them. This is captured by the creation of the commitment $C(m, c, C(c, m, goods, pay), goods)$. Additionally, the merchant commits to send the receipt to the customer in case it pays ($C(m, c, pay, receipt)$).

(a)	$send\_request$ **means** $request$ **if** $\neg quote \wedge \neg goods$
(b)	$send\_quote$ **means** $quote \wedge \text{CREATE}(C(m, c, pay, receipt)) \wedge$ $\text{CREATE}(C(m, c, C(c, m, goods, pay), goods))$
(c)	$send\_accept$ **means** $\text{CREATE}(C(c, m, goods, pay))$ **if** $\neg pay$
(d)	$send\_goods$ **means** $goods \wedge \text{CREATE}(C(m, c, pay, receipt))$
(e)	$send\_EPO$ **means** pay
(f)	$send\_receipt$ **means** $receipt$ **if** pay

Table 3.6: Constitutive Specification of NetBill Protocol.

(c1)	$\mathtt{quote}$ *before* $\mathsf{C}(\mathtt{c},\mathtt{m},\mathtt{goods},\mathtt{pay}) \vee \mathsf{C}(\mathtt{c},\mathtt{m},\mathtt{pay})$
(c2)	$\mathsf{C}(\mathtt{m},\mathtt{c},\mathtt{pay},\mathtt{receipt}) \wedge \mathtt{goods}$ *before* $\mathtt{pay}$
(c3)	$\mathtt{pay}$ *cause* $\mathtt{receipt}$
(c4)	$\mathtt{quote}$ *before* $\mathtt{pay}$

Table 3.7: Regulative Specification of NetBill Protocol.

The meaning of this action does not depend on any condition. For this reason, this action has no precondition. Intuitively, a merchant is free to make an offer at any time for some goods to someone.

SEND ACCEPT *(Table 3.6(c)).* An agent can accept a quote by means of the action *send_accept*. This entails that the customer commits to pay for the goods in case the goods are sent. The precondition to this action is that the customer should have not paid yet. In this case, indeed, the customer has already accepted the quote (eventually implicitly, that is immediately paying, without performing the acceptance of the quote explicitly).

SEND GOODS *(Table 3.6(d)).* By means of the action *send_goods* we capture the delivery of the goods to the customer. The effect this action has may depend also on the commitments holding when the action is performed. For instance, if the merchant has taken the commitment (conditional or base) of sending the goods, then the commitment is discharged. If the customer has the commitment of paying in case the merchant sends the goods, then the commitment becomes unconditional: the customer has the commitment of paying. Moreover, by means of this action, the merchant commits to send the receipt of the payment to the customer if this latter pays ($\mathsf{C}(\mathtt{m},\mathtt{c},\mathtt{pay},\mathtt{receipt})$).

SEND EPO *(Table 3.6(e)).* This action is performed by the customer and it represents the payment of the goods. This action has no preconditions, since its meaning does not depend on the truth value of some conditions. This action causes the discharge of the commitment from the customer to the merchant to pay, when it is present.

SEND RECEIPT *(Table 3.6(f)).* The meaning of the action *send_receipt* is quite intuitive from its name. This action makes sense only if the customer has paid. Otherwise, the merchant cannot emit the receipt.

Regulative Specification of NetBill

The regulative specification of NetBill protocol is given by the set of constraints reported in Table 3.7.

Constraint (c1) is of kind *before*. The condition it captures is that before the customer can commit to pay for some goods (either with a conditional commitment or with a base commitment) the merchant should propose a quote.

Also constraint (c2) is of kind *before*. It requires that before the customer pays for the goods, the merchant should commit to send the receipt in case of payment, and it should send the goods.

Constraint (c3) is of kind *cause*. It expresses that if the customer pays, then the merchant is expected to send the receipt and this latter should be sent only after. Notice that the behaviour it captures is different from the behaviour that can be expressed by the commitment $C(m, c, pay, receipt)$: the commitment does not expresses any temporal relation between the two conditions. Therefore, if allowed by the protocol actions, the merchant can send the receipt even if the customer has not paid.

Finally, constraint (c4) specifies that the customer is allowed to pay only if it has received a quote before.

Constraints' graphical representation

Figure 3.4 reports the graph of the regulative specification of NetBill protocol. The flow that it captures is divided into two parts. The first part can be followed by starting from node *(n1)*, that expresses the fact that the merchant has made a *quote*. This should happen before the commitment from the customer to pay if the merchant sends the goods ($C(c, m, goods, pay)$), or before the (base) commitment to pay. The second part concerns the delivery of the goods and the receipt. Specifically, before the customer is allowed to pay there should be the commitment from the merchant to send the receipt and the goods should have been sent. If the customer pays then, and only after, the merchant is expected to send the receipt. However, before the customer can pay it should have received a quote.

By means of this graphical representation it is possible to have an overall view of the constraints of the specification. In this way, it is

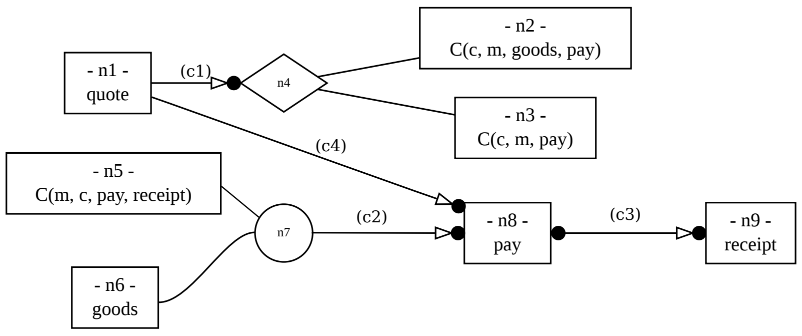


Figure 3.4: Regulative specification of NetBill protocol: boxes represent literals, the circle represents a conjunctions of boxes, the diamond represents the disjunction of boxes.

possible to infer those relations that are not explicitly captured by the constraints, but that affect the achievement of different conditions. For instance, even if there is no constraint that directly ties the commitment of the merchant to send the receipt ($C(m, c, pay, receipt)$) and the delivery of the receipt, they actually cannot but be achieved in this order. This is a consequence of constraints (c2) and (c3). The same reasoning can be performed on the constraint specification. However, a graphical representation may help, especially for complex contexts (i.e. context where many aspects need to be taken into account).

3.4.2 The Contract Net Protocol

The Contract Net (CNET) is a well known Protocol in the multi-agent system literature, where it has been deeply investigated and it has been used as a reference example in many works. In this thesis, when talking about CNET we refer to FIPA specification [Fou02c].

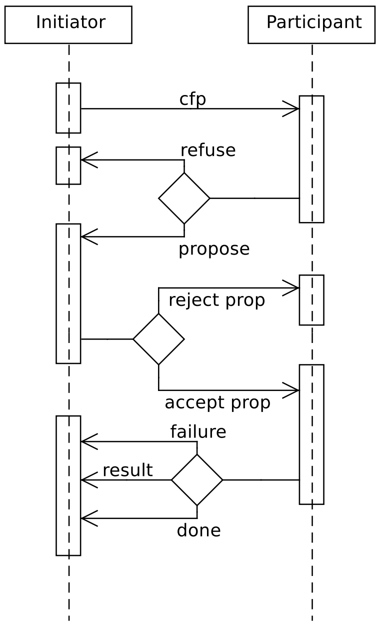


Figure 3.5: The FIPA Specification of the Contract Net Interaction Protocol [Fou02c].

The schema of the interaction of FIPA CNET is reported in Figure 3.5. Briefly, it includes two roles, the initiator (simply ‘i’ in the following) and a participant (‘p’ in the following). The initiator calls for proposals. The participant may send a proposal or a refusal. When a proposal is received, the initiator may either reject or accept it. If the proposal is accepted, the participant may answer with a failure or by sending the solution. For the sake of simplicity, we do not model the information concerning the proposal itself (e.g. costs, time, resources needed) but only the interaction concerning the task assignment and the notification of the solution.

Constitutive Specification of CNET

Let us now describe the constitutive specification of CNET actions. Actions definitions are reported in Table 3.8. Let us comment on them.

CALL FOR PROPOSALS *(Table 3.8(a)).* By means of the action *send_cfp* we meant to capture the request for help done by an initiator toward a participant, for solving a certain task. Therefore, this action does not create any commitment, but simply asserts a fact 'cfp' that records that the call for proposals has been made. The call, however, acquires this meaning if the participant has not already stated its impossibility to solve the task (or tasks of that kind) and if it has not provided the solution yet.

SEND PROPOSAL *(Table 3.8(c)).* By means of this action, the participant states its interest in solving a task. This action does not represent simply an inform but rather it is an engagement, as clearly underlined in the FIPA specification:

> The proposals are binding on the Participant, so that once the Initiator accepts the proposal, the Participant acquires a commitment to perform the task. ([Fou02c])

The participant must be aware of the role of this action because in case the initiator accepts the proposal, it will be responsible for solving it. This meaning is captured by means of the conditional commitment $C(p, i, assigned(i, p), solved(p, i))$ by which the participant 'p' commits to solve the task if the initiator 'i' assigns its solution to 'p'. In our representation we abstract from details on participant's proposal (e.g. price, time needed and so on). These details could be added as parameters, in order to let the initiator decide which proposal is more convenient.

This meaning is conditioned by the fact that the participant should have not solved the task yet. Intuitively, in this case the proposal of solving it does not actually represent a proposal. Other conditions, like the occurrence of a call for proposal, do not impact on the meaning of the action. For instance, if the participant sends a proposal before the call, it can be interpreted as advertising an offer.

SEND REFUSAL *(Table 3.8(c)).* Every participant can send a refusal by performing the action *send_refusal*. However, this action makes sense only if a call for proposal has occurred already. In this case, indeed,

(a) $\mathtt{send\_cfp}(i,p)$ **means** $\mathtt{cfp}(i,p)$ **if** $\neg\mathtt{failed}(p,i) \wedge \neg\mathtt{solved}$
(b) $\mathtt{send\_proposal}(p,i)$ **means** $\mathtt{proposal}(p,i) \wedge$
$\textsc{create}(C(p,i,\mathtt{assigned}(i,p),\mathtt{solved}(p,i)))$
if $\neg\mathtt{solved}(p,i)$
(c) $\mathtt{send\_refusal}(p,i)$ **means** $\mathtt{refused\_task}(p,i)$
if $\mathtt{cfp}(i,p) \wedge \neg\mathtt{solved}(p,i)$
(d) $\mathtt{send\_accept}(i,p)$ **means** $\mathtt{assigned}(i,p)$
(e) $\mathtt{send\_reject}(i,p)$ **means** $\mathtt{rejected\_proposal}(i,p) \wedge$
$\textsc{release}(C(p,i,\mathtt{assigned}(i,p),\mathtt{solved}(p,i)))$
if $\mathtt{proposal}(p,i)$
(f) $\mathtt{send\_done}(p,i)$ **means** $\mathtt{solved}(p,i)$
(g) $\mathtt{send\_failure}(p,i)$ **means** $\mathtt{failed}(p,i) \wedge$
$\textsc{cancel}(C(p,i,\mathtt{assigned}(i,p),\mathtt{solved}(p,i))) \wedge$
$\textsc{cancel}(C(p,i,\mathtt{solved}(p,i)))$

Table 3.8: Constitutive Specification of Contract Net Protocol.

the participant has something to refuse, otherwise it is not clear what it is refusing.

Send Accept *(Table 3.8(d)).* The action *send_accept* is performed by the initiator to assign the solution of the task to a participant. Notice that the precondition does not require the presence of an offer from the participant. This is because a task can be assigned even to someone that has not sent a proposal. Consider, for instance, an order from the head of a company to an employee. In this case the presence of an offer from the employee is not necessary since it is part of its duties. If this eventuality is to be avoided, then it should be captured by a proper constraint. This requirement, indeed, impacts on the way the action can be used, rather than contextualize its meaning. This justifies the use of a constraint rather than of a precondition.

Send Reject *(Table 3.8(e)).* With this action the initiator rejects a participant's proposal to solve the task. Therefore, its execution releases the participant from its commitment to solve the task. The precondition to this action is that there must exist a proposal from the participant to solve the task, otherwise it is not clear what the initiator is refusing.

Send Done *(Table 3.8(f)).* It is the action by which the agent informs the initiator that it has finished to compute the solution for the task. It asserts the fact $\mathtt{solved\_task}(p,i)$ and consequently discharges (if present) the corresponding commitment of the participant.

This action has no precondition. Consider, for instance, the case of a participant that sends the solution of a task without being assigned of solving it. Also in this case the meaning of the action is that the task has been solved and the solution has been sent. If this behaviour is to be avoided, then this could be captured by defining a proper constraint

(c1) cfp *cause* refused_task(p, i) OR C(p, i, assigned(i, p), solved(p, i))
(c2) C(p, i, assigned(i, p), solved(p, i)) *cause* rejected_proposal(i, p) OR assigned(i, p)
(c3) assigned(i, p) *cause* solved(p, i) OR failed(p, i)
(c4) refused_task(p, i) *not co-exist* C(p, i, assigned(i, p), solved(p, i))
(c5) rejected_proposal(i, p) *not co-exist* assigned(i, p)
(c6) solved(p, i) *not co-exist* failed(p, i)

Table 3.9: Regulative Specification of Contract Net Protocol.

(e.g. capturing that a participant is allowed to send a result only after the initiator has assigned it the task).

SEND FAILURE *(Table 3.8(g)).* If an agent starts to compute the solution of a task and it realizes that it is not able to provide a solution, it can send a failure. In this case the fact failed(p, i) is asserted and the participant is released of its commitment to solve the task.

As well as the action *send_done*, also the *send_failure* has no preconditions. Indeed, the participant can perform it at any moment in order to announce that it would not be able to solve a certain task. For instance, consider a web service. In case some internal errors arise, it can use this action to inform all the clients connected to it that it will not be able to satisfy any request. A reasonable doubt that may arise at this point, is how to avoid that the participant abuse of this action. It allows, indeed, to cancel a commitment. A malicious agent may take advantage of it by sending a failure not because it would not be able to solve the task but because it has changed its mind and it is no more willing to help the initiator. In this case, by means of this action it is released of its duties without incurring in any violation. Actually, this is a possible scenario but it cannot be avoided at the constitutive level. These behaviours must be discouraged by mechanisms such as, for instance, the payment of penalties (which is not treated in this thesis).

Regulative Specification of CNET

The set of actions defined above represents the actions that the agents can perform during the interaction. Each of them can be executed *at any point* (preconditions impacts on the meaning of the actions, not on their execution) and their aim is to capture the meaning of the actions leaving aside the *desired* use of them. The specification of the desired *patterns* on the interaction is demanded to 2CL constraints.

The set of 2CL constraints, that defines the regulative specification of CNET protocol, are reported in Table 3.9. Constraint (c1) is of the kind *cause* and the condition that it captures on the interaction is that when a call for proposal $\mathtt{cfp}$ is made, the participant is expected to either refuse the task or to take the commitment to solve it by making a proposal. It is not obliged to take a decision at the next step of its execution (other actions can be performed in between) but sooner or later it is expected to do it. Moreover, the constraint foresees that the participant cannot take the initiative of proposing to solve a task (or of refusing to do something). This could be done only after the initiator has declared that there is a task to solve.

Constraints (c2) and (c3) are similar to the previous one since they are all of kind *cause*. (c2) requires that if the participant commits to solve the task, then, and only after, the initiator must either reject the proposal or assign the task to that participant. Constraint (c3) requires that when a task is assigned to a participant then it is bound to reply (either with a solution or with a failure). However, the participant cannot send a *failure* or a *solved_task* before it is assigned to solve the task. Notice that the conditional commitment $C(p, i, assigned(i, p), solved(p, i))$ is not sufficient to capture an order between the assignment of the task and its solution. Since conditional commitments do not express any temporal relation between the antecedent and the consequent condition, the participant can solve the task and discharge the commitment, before it is activated by the initiator assigning the task.

Constraints (c4-c6) are of different nature with respect to previous constraints. They are of kind *not co-exist*, thus capturing a relation among conditions without any temporal aspect. Specifically, (c4) states that a participant cannot refuse to solve a task and send the proposal for solving it during the same execution (no matter which of the two conditions is achieved first, but only one of them can become true). (c5) requires that the initiator cannot both accept and refuse a proposal from a participant. Finally, (c6) states that a participant cannot send the solution of a task and the failure in solving it, both in the same execution.

The specification of the CNET described above, is the 2CL specification which is closest to the FIPA specification. Indeed, constraints c1-3 impose the same sequence, in terms of allowed evolutions of the social state, represented in the FIPA specification (see Figure 3.5), while constraints c4-6 capture mutual exclusion relations. The difference is that, by means of 2CL operators, the order is preserved but no rigid sequences of events are given: other actions can be performed in between the achievement of different state conditions that satisfy the constraints.

Constraints' graphical representation

The regulative specification of 2CL CNET protocol is graphically represented in Figure 3.6 (for the sake of readability parameters of literals

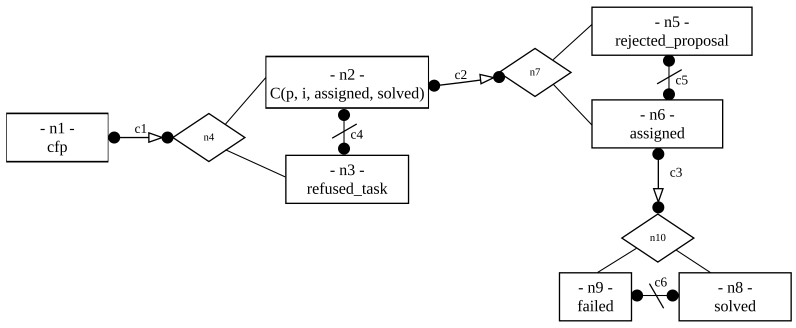


Figure 3.6: Graphical representation of the regulative specification of the Contract Net Protocol. Boxes represent facts or commitments and diamonds represent the disjunction of boxes.

are omitted in the figure). The Figure highlights the (flexible) flow imposed by the constraints. It can be perceived by starting from node *(n1)* that captures the initiator's call for proposals (cfp). If this happens, afterwards (constraint (c1): •—▷•) there must be a proposal to solve the task (node *n2*) or a refusal (node *n3*) but not both (constraint (c4): •/•). Node *(n4)* represents a disjunction. If the condition in node *(n2)* is achieved, i.e. the participant makes a proposal, then one among conditions in node *(n5)* and node *(n6)* must be achieved. As before, they cannot occur both in the same execution, as captured by constraint (c5). In case the participant is commissioned to solve the task, node *(n6)*, then one among conditions in node *(n8)* and *(n9)* must be achieved but not both (c6).

3.4.3 2CL Protocol Adaptation: CNET Variants

In this section we present some variants of CNET protocol. The aim is to practically show how an existing 2CL protocol specification can be easily adapted to different needs. When modifying a 2CL protocol specification, indeed, it is possible to act separately at the level of the actions and at the level of constraints. This aspect increases the protocol reuse and simplifies protocol modification, as shown in the following.

The variants that we present below are obtained by modifying the constraints of the specification given in Section 3.4.2, that represents the closest representation of FIPA CNET (in terms of allowed interactions). Let us see how we can allow different interactions by acting at the level of constraints.

Lazy and zealous participant

For the first examples let us consider constraint (C1) of specification in Section 3.4.2:

$$\mathtt{cfp}\ \textit{cause}\ \mathtt{refused\_task}(p,i)\ \textsc{or}\ \mathsf{C}(p,i,\mathtt{assigned}(i,p),\mathtt{solved}(p,i))$$

The *cause* operator that is used in this constraint requires that if the initiator issues a call for proposals, then (and only after) the participant is asked to reply with a proposal (a commitment to solve the task) or with a refuse.

The lazy participant variant

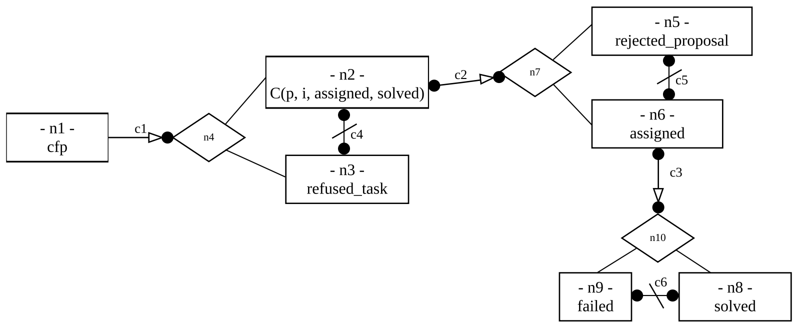
In some contexts, however, this requirement is too strong and the participant should be left free to decide whether to reply or not. Consider, for instance, a conference setting. In this case many *call for papers* are sent around in order to advertise the event. However, the organisers do not expect an answer from all those who have received the call. An interested author can submit the title and the abstract, thus committing to submit also the remaining part of the paper.

How is it possible to specify a constraint that does not impose participants to answer? This can be done by substituting the *cause* (•—▷•) operator in (C1) with a *before* (—▷•) constraint, as shown in Figure 3.7(a). A constraint of this kind requires that the answer of the participant happens after the `cfp` but does not impose it to answer (the participant is allowed to have an unresponsive behaviour). This variant is called the "lazy participant" variant of the CNET.

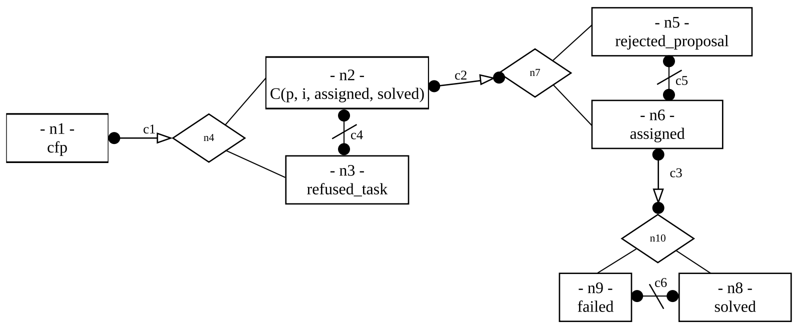
It is interesting to notice that a similar "lazy" behaviour can be easily obtained also for the initiator, by adopting a similar mechanism on (C2). This constraint, indeed, is a *cause* constraint, thus requiring the initiator to answer to a participant's proposal. By replacing it with a *before*, the initiator can reply with an acceptance or a rejection of the participant's proposal only after the proposal has been made, but it is not obliged to answer.

The zealous participant variant

Another variant that we propose suit well in an advertising context, where people propose to solve some tasks or advertise some services, before a client formulates a request. Basically, in this setting, when the initiator (e.g. a client) asks for a proposal from a participant (e.g. a company) this latter is expected to answer. However, a company may advertise its products without waiting for a request. In general, this variant of the CNET can be applied in those situations where participants are already known. The role of a *call for proposals*, indeed, is to discover who will take part to the interaction. Allowing participants to answer before the call entails that they are already known and able to interact with the initiator agent.



(a) Lazy Participant;



(b) Zealous Participant;

Figure 3.7: Variants of the CNET involving constraint c1: the *Lazy Participant* and the *Zealous Participant* specifications.

In order to obtain the described behaviour, we have to modify constraint (c1) of the CNET specification. Again, we need to substitute the *cause* operator, this time with a *response* constraint, obtaining the following result:

$$\mathtt{cfp}\ \textit{response}\ \mathtt{refused\_task}(p,i)\ \textsc{or}\ \mathsf{C}(p,i,\mathtt{assigned}(i,p),\mathtt{solved}(p,i))$$

This constraint, does not prevent the participant to answer before the call, but when this is issued by the initiator, then it is expected to answer. We called this CNET variant "zealous participant" and it is depicted in Figure 3.7(b).

CNET with Anticipated Failure

Let us now consider another context that can be identified as a *bidding* scenario. The characteristics of this scenario are: *(i)* an initiator publishes an open call, e.g. in an official gazette, that does not require the subscribers to the gazette to answer; *(ii)* a participant can notify a failure in solving the task, even if the task has not been assigned to it yet. This is useful when the initiator makes a cfp and the participant realises that it would not be able to solve the task. The model should include the possibility for a participant to communicate an *anticipated*

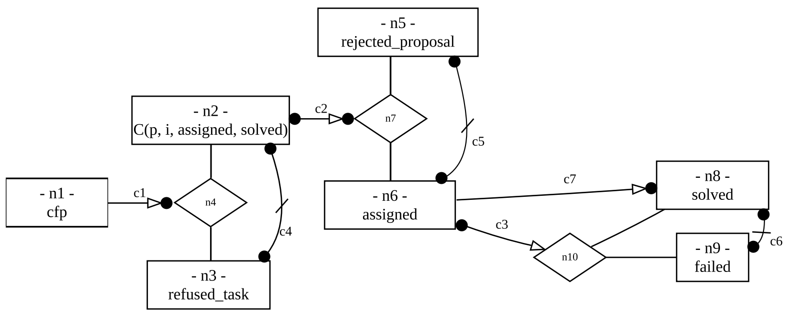


Figure 3.8: CNET variant with Anticipated Failure.

failure; *(iii)* anticipated solutions are not allowed, that is to say that in order to send the solution, a participant must wait until the task is assigned to it.

For what concerns the first requirement, the solution is to substitute the *cause* operator with a *before* constraint (as done in the "Lazy Participant" version). In this way, participants must wait for the call in order to answer but they are not obliged to answer.

The second requirement, the "anticipated failure", is forbidden in the original version by constraint (c3) ($\mathsf{assigned}(i, p)$ *cause* $\mathsf{solved}(p, i)$ OR $\mathsf{failed}(p, i)$). Also in this case the problem is the *cause* operator that requires a sequence between the antecedent and the consequent conditions. If the requirement is that the participant can declare a failure before the assignment of the task, then the *cause* constraint should be transformed into a *response*, as shown below:

$$\mathsf{assigned}(i, p) \textit{ response } \mathsf{solved}(p, i) \text{ OR } \mathsf{failed}(p, i)$$

In this way, when the participant is assigned of a task it is obliged to answer, but it does not have to wait for the assignment.

The new version of the constraint (c3) (reported above), does not forbid the participant to send the solution even if the task is not assigned to it. In the requirement of this version, however, we want to forbid "anticipated solutions". To require that the solution can be sent only after the assignment we add a *before* constraint between these two conditions, in the following way:

$$\mathsf{assigned}(i, p) \textit{ before } \mathsf{solved}(p, i)$$

Figure 3.8 shows the set of constraints of the Anticipated Failure variant of CNET.

Soft Contract Net Protocol

The last version of the Contract Net Protocol that we propose is a very soft interaction protocol that, differently from the previous ones, expresses just a few regulative constraints, leaving a much greater freedom of behaviour to the initiator and to the participant.

The only requirements that this version expresses are: *(i)* a task cannot be assigned to a participant who has not yet committed to solve it; *(ii)* the assignment of a task to a participant and the rejection of its proposal are mutually exclusive; *(iii)* a participant proposal to solve a task and refusing to solve it are mutually exclusive.

The set of constraints that capture these requirements are the following:

c1: $\mathsf{C}(p, i, \mathtt{assigned}(i, p), \mathtt{solved}(p, i))$ *before* $\mathtt{assigned}(i, p)$
c2: $\mathtt{refused\_task}(p, i)$ *not co-exist* $\mathsf{C}(p, i, \mathtt{assigned}(i, p), \mathtt{solved}(p, i))$
c3: $\mathtt{rejected\_proposal}(i, p)$ *not co-exist* $\mathtt{assigned}(i, p)$

In particular, constraint (c1) requires that before a task is assigned to a participant there must be a commitment from the participant to solve it. Constraints (c2) and (c3) capture the mutual exclusion requirements between a refusal and a proposal of solving the task (c2), and between the acceptance and the reject of a participant's proposal (c3).

This version is called "Soft CNET" since it allows for many interactions that are forbidden by the CNET specification. For instance, in the soft version a participant can propose to solve a task for which no call has been made, and it can also send a solution before any assignment of the task. The initiator is allowed to do not answer to a participant's proposal.

The soft variant of CNET is depicted in Figure 3.9.

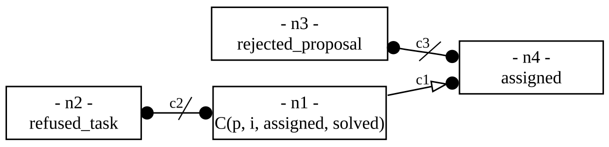


Figure 3.9: Soft CNET variant.

3.5 CONSIDERATIONS ABOUT COMMITMENTS AND CONSTRAINTS

In this section we comment and summarise some relevant issues concerning the relation between commitments and constraints [BBMP13], including some considerations on the autonomy of the agents.

Can commitment substitute constraints?

For what concerns whether constraints can be represented (and, then, substituted) by proper commitments, this is not possible because the expressive power of 2CL constraints and of commitments is different. We have already mentioned this aspect, but let us come back on this point and explain it by means of the Robert's Rules of Order example. Figure 3.2 reports graphically the constraints that define the regulative specification of this protocol. Here there is the need to express a temporal ordering between the assignment of the floor and the discussion of a motion by the participant: the former condition is to be achieved first. The only way in which it is possible to express relations between conditions by means of commitments is by using conditional commitments of the kind:

$$C(p, ch, assign\_floor, talk\_started)$$

By this commitment the participant p commits to the chair ch to discuss the motion. This is to be done conditioned to the fact that the chair ch assigned p the floor ($assign\_floor$). Different evolutions of the interaction are allowed. In particular, it is not required that these two conditions are achieved subsequently. Among the possible executions there is also the possibility for the participant to start talking without having the floor. This means that the antecedent conditions that are used inside commitments are not necessarily to be accomplished before the engagement to bring about the consequent conditions. For this reason commitments cannot be used to impose temporal orderings. Instead, constraints are specifically used to express patterns of interaction, including temporal relations. In the above case one could, for instance, specify:

$$assign\_floor \relbar\!\triangleright\!\bullet\; talk\_started$$

meaning that the condition $talk\_started$ cannot be achieved before condition $assign\_floor$.

Are there differences in the nature of engagements?

Another important issue related to the (dis)similarities between constraints and commitments concerns the *nature of the engagements* that they represent. Ever since their introduction, commitments have been given an explicit regulative nature [Cas95, Sin99]. Also constraints, which define the patterns of interaction of a protocol, have a regulative nature, in the sense that they represent what must hold in an execution for a protocol to be respected. Both constraints and commitments, due to this regulative nature, introduce a notion of *violation*. There are, however, some differences. An agent takes a commitment by autonomously deciding to execute an action, whose public and shared meaning includes that commitment. Afterwards, it is bound to make the condition in the commitment become true. As soon as the condition of the commitment is achieved, the commitment is discharged. The debtor paid its debt. On the other hand, an agent that accepts to interact according to a protocol accepts to stick to the patterns of interaction that the protocol specifies. The agent is free to execute any action, as long as it respects the rules. While a commitment requires the debtor to satisfy it sooner or later, a constraint is to be respected for the whole interaction. A consequence of the different nature of commitments and of constraints concerns the *time* at which *violations can be detected*. Violations of commitments can be detected only at the end (or what is considered as the end) of the interaction: an ending state containing a commitment means that the debtor did not comply to what expected from it. Violations to constraints, instead, can be detected during the interaction. A constraint is, in fact, like a boundary that should not be crossed. Another consequence of their different nature is that commitments can be used to capture *achievement* conditions requiring agents to do something. Constraints, instead, may also require agents not to do something.

What about profiting of opportunities?

The classical commitment-based approach is very respectful of the agents' autonomy. Autonomy implies that each agent decides what is the best for itself. For this reason, protocols do not dictate agents when to execute specific actions. In particular, an agent can decide to take an opportunity, when one arises. This spirit is respected by our proposal, even though our protocols include a regulative specification which restricts the acceptable executions. Constraints, indeed, do not specify rigid flow of interaction or which actions should be performed to satisfy the conditions they relate. Therefore, agents are free to decide how to shape the interaction in such a way to satisfy both constraints and commitments, behaving as they prefer for all the aspects that are not ruled by them. For instance, the presence of a constraint does not capture which actions are to be executed and, even in case it is a temporal constraint, it does not require the antecedent and consequent conditions to be satisfied in to subsequent states. Therefore, within the boundaries specified by constraints and commitments, agents are free to decide to undertake the interaction that better answers their needs.

Since we do not regiment the regulative rules, an agent can also decide to break the rules if a situation it can take advantage of arises, although this amounts to a violation in our framework. Let us explain the meaning of opportunity, and the possibility to incur into a violation, with the help of a simple example: suppose that at a summer school the official language everybody should speak is English. Every student at the school has a badge reporting the name and the nationality of the person. If a French attendee meets a colleague whose badge says she is from France, the first student might decide to speak in French with the other even though the official language she should speak is English. There is a clear expected advantage in doing so: a better understanding. However, the violation of the rules of the protocol introduces a certain amount of *risk*. For example, if the second student by mistake took the badge of her room-mate, she might not understand the former one because the protocol is not attended. This would not have happened if the first agent did not violate the rules.

Do one's commitments commit others to commit?

This question concerns constraints that relate commitments with different debtors, like for instance:

$$\mathsf{C}(\mathsf{c}, \mathsf{m}, \mathsf{purchase}(\mathsf{goods})) \bullet\!\!\longrightarrow\!\!\bullet\ \mathsf{C}(\mathsf{m}, \mathsf{c}, \mathsf{sold}(\mathsf{goods}, \mathsf{price}))$$

by which the commitment of the customer c to buy some goods $\mathsf{C}(\mathsf{c}, \mathsf{m}, \mathsf{purchase}\ (\mathsf{goods}))$ is to be followed by the commitment of the merchant m to sell the goods at some agreed price ($\mathsf{C}(\mathsf{m}, \mathsf{c}, \mathsf{sold}(\mathsf{goods}, \mathsf{price}))$). It may seem, in this case, that the autonomy of the merchant is reduced by the simple fact that another agent took some commitment, but it is not so. In fact, since constraints are supposed to be public and they can be inspected, an agent, willing to play a role in a protocol, has the means for understanding if that pattern of interaction meets its goals. By autonomously deciding whether entering the protocol it, however, *commits* to respect its rules along the *whole* interaction. The fact that all agents accept to respect the rules has the advantage of making the course of interaction predictable and, therefore, of giving guarantees to all of the participants. In the example, the customer has the guarantee that the merchant will take the commitment to sell the goods after it commits to buy.

4 A COMMITMENT MACHINE FOR 2CL PROTOCOL SPECIFICATION

CONTENTS

In this Chapter we provide an operational semantics for 2CL protocols. To this aim we define the 2CL-Generalized Commitment Machine (2CL-GCM). 2CL-GCM relies on Generalized Commitment Machine (GCM), defined by Singh [Sin07], for what concerns the inference of the possible evolutions of the social state. This is given by relying on protocol actions and commitment life cycle. 2CL-GCM additionally takes into account a set of 2CL constraints that further restricts the set of legal paths. Specifically, besides requiring for a path to be legal to satisfy all the active commitments, 2CL-GCM requires it to satisfy also the protocol constraints. In order to perform the verification of constraints on the 2CL-GCM paths we first provide a Linear-time Temporal Logic (LTL) interpretation of 2CL operators. By relying on this interpretation we determine whether a constraint is satisfied or it is violated.

4.1 LINEAR-TIME TEMPORAL LOGIC IN A NUTSHELL

In this section we briefly recall Linear-Time Temporal Logic [Eme90] syntax and semantics. The reader already familiar with this logic can skip this section.

LTL Syntax

The Linear-time Temporal Logic is a logic whose connectives allow to refer to the future. It models time as a sequence of states, extending infinitely into the future [HR04].

Definition 4.1 (LTL Syntax)**.** *The syntax of the logic LTL is given by the following grammar:*

$$\varphi ::= \bot \mid \top \mid p \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi)$$

$$\mid \bigcirc \varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi \mathsf{U} \varphi \mid \varphi\, U_S\, \varphi \mid \varphi \mathsf{R} \varphi$$

where p *is any propositional atom from some set* ATOMS*.*

Temporal connectives are $\bigcirc$ *next state* (the formula is satisfied in the immediately following state), $\Diamond$ *eventually* in the future (the formula is eventually satisfied), $\Box$ *globally* (the formula is true in all the subsequent states), U *weak until*, U_S *strong until*, and R *release*.

As a convention, the unary connectives (i.e. $\neg$, $\bigcirc$, $\Diamond$ and $\Box$) bind most tightly. Next there are the binary temporal connectives (i.e. U, R and U_S). Then there are the logical connectives $\wedge$ and $\vee$. Finally there is the logical implication $\rightarrow$.

LTL Semantics

The systems on which LTL formulas are verified are modelled as labelled transition systems (simply *transition system* in the following). A transition system can be seen as an oriented graph whose nodes represent *states* and arcs represent *transitions* among the states. A state contains some information about the system, while transitions describe how a system may evolve from a state to another.

Definition 4.2 (Transition system)**.** *A transition system* M *is a tuple* $M = (S, \delta, I)$ *such that (i)* S *is a set of states; (ii)* $\delta \subseteq S \times S$ *is a transition relation such that* $\forall s \in S, \exists s' \in S$ *with* $(s, s') \in \delta$*; and (iii)* $I : S \rightarrow \mathcal{P}(\text{ATOMS})$ *is an interpretation function, where* ATOMS *is the set of atomic propositions and* $\mathcal{P}(\text{ATOMS})$ *is the power set of* ATOMS*.*

The role of the interpretation function I is to assign to each state a set of atomic propositions holding in the state. Condition *(ii)* of the above definition requires the transition relation to be serial [HR04], that is: for each state s of the system, there exists at least one state $s' \in S$ such that $(s, s') \in \delta$. Notice that it is possible to introduce seriality by adding to the system an extra state s_d with a self-loop, and to add new transitions from all the states without outgoing transitions to s_d [HR04]. Other mechanisms for achieving seriality can be found in [BK08, BA08].

Let us now define a *Path* of a transition system.

Definition 4.3 (Path of a Transition System)**.** *A path in a model* $M = (S, \delta, I)$ *is an infinite sequence of states* $\tau_1, \tau_2, \tau_3, \ldots$ *in* S *such that,* $\forall i \geqslant 1, (\tau_i, \tau_{i+1}) \in \delta$*. A path is represented as* $\langle(\tau_1, \tau_2), (\tau_2, \tau_3), \ldots\rangle$

Now we have all the elements for defining the LTL satisfaction relation ($\models_{LTL}$).

Definition 4.4 (LTL semantics). *Let* $M = (S, \delta, I)$ *be a model and* $\tau = \langle(\tau_1, \tau_2), (\tau_2, \tau_3), \ldots\rangle$ *be a path in* M. *Let* $\tau(i) = \tau_i$. *Let* $\tau^i = \langle(\tau_i, \tau_{i+1}), \ldots\rangle$ *be a suffix of* τ. *Whether* τ *satisfies an LTL formula is defined by the satisfaction relation* $\models_{LTL}$ *as follows:*

1. $\tau \models_{LTL} \top$
2. $\tau \not\models_{LTL} \bot$
3. $\tau \models_{LTL} p$ *iff* $p \in I(\tau(1))$
4. $\tau \models_{LTL} \neg\varphi$ *iff* $\tau \not\models_{LTL} \varphi$
5. $\tau \models_{LTL} \varphi_1 \wedge \varphi_2$ *iff* $\tau \models_{LTL} \varphi_1$ *and* $\tau \models_{LTL} \varphi_2$
6. $\tau \models_{LTL} \varphi_1 \vee \varphi_2$ *iff* $\tau \models_{LTL} \varphi_1$ *or* $\tau \models_{LTL} \varphi_2$
7. $\tau \models_{LTL} \varphi_1 \rightarrow \varphi_2$ *iff* $\tau \models_{LTL} \varphi_2$ *whenever* $\tau \models_{LTL} \varphi_1$
8. $\tau \models_{LTL} \bigcirc\varphi$ *iff* $\tau^2 \models_{LTL} \varphi$
9. $\tau \models_{LTL} \Box\varphi$ *iff* $\forall i \geqslant 1, \tau^i \models_{LTL} \varphi$
10. $\tau \models_{LTL} \Diamond\varphi$ *iff there is some* $i \geqslant 1$ *such that* $\tau^i \models_{LTL} \varphi$
11. $\tau \models_{LTL} \varphi\ U_S\ \psi$ *iff there is some* $i \geqslant 1$ *such that* $\tau^i \models_{LTL} \psi$ *and for all* j, $1 \leqslant j \leqslant i-1$, $\tau^j \models_{LTL} \varphi$
12. $\tau \models_{LTL} \varphi\ \ U\ \ \psi$ *iff there is some* $i \geqslant 1$ *such that* $\tau^i \models_{LTL} \psi$ *and for all* j, $1 \leqslant j \leqslant i-1$, $\tau^j \models_{LTL} \varphi$*; or for all* $k \geqslant 1$, $\tau^k \models_{LTL} \varphi$
13. $\tau \models_{LTL} \varphi\ R\ \psi$ *iff there is some* $i \geqslant 1$ *such that* $\tau^i \models_{LTL} \varphi$ *and for all* j, $1 \leqslant j \leqslant i$, $\tau^j \models_{LTL} \psi$*; or for all* $k \geqslant 1$, $\tau^k \models_{LTL} \psi$

Let us comment on some of the above clauses. When a formula does not involve temporal operators, then the formula is evaluated in the first state of the path. This is what happens in clause 3. Temporal operators, instead, are used to move forward in the path for the verification of the conditions. For instance, clause 8 allows the verification of a formula starting from the second state of the path (τ^2).

Clauses 11-13 deal with binary temporal connectives. In particular, the *until* operators require that a formula φ_1 is invariably true until another condition φ_2 is achieved. The difference between *strong until* (U_S) and weak until (U) is that the former requires that φ_2 becomes true sooner or later. The weak until does not capture this requirement. Notice that for both operators, once the condition φ_2 is achieved there are no requirements on the truth value of φ_1. The *release* connective, instead, is the dual of $\mathcal{U}_S$. Specifically, $\varphi_1 \mathsf{R} \varphi_2$ equals to $\neg(\neg\varphi_1 \mathcal{U}_S \neg\varphi_2)$.

In all temporal clauses (9-13) the future includes the present. This means that the formula must be verified starting from the current state on. A consequence of this convention is that the formulas $\Box p \rightarrow p$, $p \rightarrow q \mathcal{U}_S p$, and $p \rightarrow \Diamond p$ are true in every state of every model.

The satisfaction relation given above is defined on one path. In the following Definition we provide the satisfaction relation for a model.

Definition 4.5. *Let $M = (S, \delta, I)$ be a model, $s \in S$ and φ is a LTL formula. We write $M, s \models_{LTL} \varphi$ if for every execution path τ of M starting at s we have that $\tau^s \models_{LTL} \varphi$.*

Given the above definition and the LTL satisfaction relation it is possible to define when two formulas are equivalent.

Definition 4.6 (Equivalence)**.** *Let φ and ψ be two LTL formulas. We say that φ and ψ are (semantically) equivalent if for all models M and all paths τ of the model, $\tau \models_{LTL} \varphi$ iff $\tau \models_{LTL} \psi$. We denote equivalence as $\varphi \equiv_{LTL} \psi$.*

The way a formula can be verified in a given model is by means of *model checking*. The literature about model checking is wide and different model checkers have been proposed. Some of the most famous model checkers are SPIN [Hol03] (LTL model checker based on automata verification), NuSMV [CCGR00] (CTL and LTL symbolic model checker), MCMAS [Rai06] (a symbolic model checker tailored for MAS verification).

4.2 AN LTL INTERPRETATION FOR 2CL CONSTRAINTS

A 2CL protocol defines a set of legal interactions by specifying a set of constraints that must be respected. Intuitively, a set of agents will be compliant to the protocol when all the commitments they have taken are satisfied (as usual in the commitment approach), and the overall execution respects the constraints that constitute the regulative specification. With the aim of providing a way for verifying if a certain interaction is legal according to the protocol constraints, we associate to each constraint a formal interpretation given in terms of LTL formulas. In order for the interpretation to be given on general formulas, we consider 2CL constraints expressed in terms of *Disjunctive Normal Form* (DNF) formulas. A DNF is defined in terms of conjunctions and disjunctions of facts and commitments (to which we refer as *literals* in the following).

Definition 4.7 (2CL Disjunctive Normal Form Formulas)**.** *Let* l_i *be a literal, i.e. a fact or a commitment. Let* $cf = l_1 \wedge \cdots \wedge l_i \wedge \ldots l_n$ *be a* 2CL conjunctive formula*. A* 2CL disjunctive normal form *formula* dnf *is a disjunction of* 2CL *conjunctive formulas:* $dnf = cf_1 \vee \cdots \vee cf_i \vee \ldots cf_n$*.*

The way we obtain the *exclusive disjunction* in the above formula is by means of the following equivalence: $A \text{ XOR } B \equiv (A \wedge \neg B) \vee (\neg A \wedge B)$.

Given a 2CL constraint c we define, for each constraint, its corresponding LTL interpretation and we denote it as $\varphi_{ltl}(c)$.

Definition 4.8 (LTL Constraints Interpretation)**.** *Let* $c = dnf_1 \text{ op } dnf_2$ *be a* 2CL *constraint.* $\varphi_{ltl}(c)$ *is the LTL interpretation of constraint* c *as reported in Table 4.1.*

Let us now describe the intuition behind the interpretation we provide for each operator. As a general consideration, notice that the semantics of negated operators does not always correspond to the negation of the semantics of the corresponding positive operators. This is due to the intention of capturing a different meaning for these operators. In most cases, indeed, the LTL interpretation of a positive operator is given as an implication that can be read as: "if something happens, then something else is expected to happen too". In a certain sense we can say that these constraints have a *conditional* characterization. By negating an implication the formula is transformed into a conjunction (according to classical logic):

$$\neg(a \rightarrow b) \equiv \neg(\neg a \vee b) \equiv a \wedge \neg b$$

Instead, the idea is to preserve a conditional characterization also for negative constraints (see below for the comments referred to each operator).

4.2.1 Relation operators

The set of relation operators is made of *correlation* and the *co-existence* (this latter is derived from the former). In order to provide an LTL interpretation that captures the meaning of these operators it is important to recall that they do not require any temporal ordering on the achievement of the conditions involved in the constraints.

	Relation	Constraint	φ_{ltl}
Relation Operators	*correlate*	$\mathrm{dnf}_1 \bullet\!\!- \mathrm{dnf}_2$	$\Diamond \mathrm{dnf}_1 \rightarrow \Diamond \mathrm{dnf}_2$
	not correlate	$\mathrm{dnf}_1 \bullet\!\!\not- \mathrm{dnf}_2$	$\Diamond \mathrm{dnf}_1 \rightarrow \neg\Diamond \mathrm{dnf}_2$
	co-exist	$\mathrm{dnf}_1 \bullet\!\!-\!\!\bullet \mathrm{dnf}_2$	$\varphi_{ltl}(\mathrm{dnf}_1 \bullet\!\!- \mathrm{dnf}_2) \wedge$ $\varphi_{ltl}(\mathrm{dnf}_2 \bullet\!\!- \mathrm{dnf}_1)$
	not co-exist	$\mathrm{dnf}_1 \bullet\!\!\not-\!\!\bullet \mathrm{dnf}_2$	$\varphi_{ltl}(\mathrm{dnf}_1 \bullet\!\!\not- \mathrm{dnf}_2) \wedge$ $\varphi_{ltl}(\mathrm{dnf}_2 \bullet\!\!\not- \mathrm{dnf}_1)$
Temporal Operators	*response*	$\mathrm{dnf}_1 \bullet\!\!-\!\!\triangleright \mathrm{dnf}_2$	$\Box(\mathrm{dnf}_1 \rightarrow \Diamond \mathrm{dnf}_2)$
	not response	$\mathrm{dnf}_1 \bullet\!\!\not-\!\!\triangleright \mathrm{dnf}_2$	$\Box(\mathrm{dnf}_1 \rightarrow \neg\Diamond \mathrm{dnf}_2)$
	before	$\mathrm{dnf}_1 -\!\!\triangleright\!\!\bullet \mathrm{dnf}_2$	$\neg \mathrm{dnf}_2 \; \mathsf{U} \; \mathrm{dnf}_1$
	not before	$\mathrm{dnf}_1 \not-\!\!\triangleright\!\!\bullet \mathrm{dnf}_2$	$\Box(\Diamond \mathrm{dnf}_2 \rightarrow \neg \mathrm{dnf}_1)$
	cause	$\mathrm{dnf}_1 \bullet\!\!-\!\!\triangleright\!\!\bullet \mathrm{dnf}_2$	$\varphi_{ltl}(\mathrm{dnf}_1 \bullet\!\!-\!\!\triangleright \mathrm{dnf}_2) \wedge$ $\varphi_{ltl}(\mathrm{dnf}_1 -\!\!\triangleright\!\!\bullet \mathrm{dnf}_2)$
	not cause	$\mathrm{dnf}_1 \bullet\!\!\not-\!\!\triangleright\!\!\bullet \mathrm{dnf}_2$	$\varphi_{ltl}(\mathrm{dnf}_1 \bullet\!\!\not-\!\!\triangleright \mathrm{dnf}_2) \wedge$ $\varphi_{ltl}(\mathrm{dnf}_1 \not-\!\!\triangleright\!\!\bullet \mathrm{dnf}_2)$
Strong Sequence	*premise*	$\mathrm{dnf}_1 \triangleright\!\triangleright\!\!- \mathrm{dnf}_2$	$\Box(\bigcirc \mathrm{dnf}_2 \rightarrow \mathrm{dnf}_1)$
	not premise	$\mathrm{dnf}_1 \triangleright\!\triangleright\!\!\not- \mathrm{dnf}_2$	$\Box(\bigcirc \mathrm{dnf}_2 \rightarrow \neg \mathrm{dnf}_1)$

Table 4.1: LTL interpretation of 2CL constraints.

Correlation

Requirement. Correlation requires that once a certain condition is met, then, no matter when (before, in the same state or after), another condition should be achieved.

This requirement can be captured in LTL by the temporal operator $\Diamond$, by means of which it is possible to express that there must exist a state in which a certain condition is achieved. To give the intuition on how the *correlation* can be expressed, let us first consider a constraint made of simple formulas. The interpretation is the following:

$$l_1 \bullet\!\!- l_2 \equiv \Diamond l_1 \rightarrow \Diamond l_2$$

The formula involves temporal operators but the point in time in which the two conditions are achieved is independent one from the other. Specifically, both the antecedent condition l_1 and the consequent condition l_2 are preceded by a $\Diamond$ operator. The resulting requirement that the formula captures is: starting from a certain state and considering the future, if there exist a state at which l_1 is achieved, then there must exist another state at which l_2 is achieved. However, these two states are independent one from the other, thus entailing that the formula does not express requirements on which condition should be achieved first. The important aspect is that if l_1 is achieved, then also l_2 must be achieved sooner or later.

LTL Interpretation. The formula given above can be generalized, thus allowing to express more complex conditions. Constraint "$\mathtt{dnf}_1$ *correlates* $\mathtt{dnf}_2$" ($\mathtt{dnf}_1 \bullet\!\!- \mathtt{dnf}_2$) corresponds to:

$$\varphi_{ltl}(\mathtt{dnf}_1 \textit{correlate}\ \mathtt{dnf}_2) = \Diamond\ \mathtt{dnf}_1\ \rightarrow\ \Diamond\ \mathtt{dnf}_2 \tag{4.1}$$

Negative Correlation

Requirement. The negative form of correlation captures that if a certain condition is met in an execution, then another condition must never be achieved.

The intuitive interpretation given on a simple formula is the following:

$$l_1 \bullet\!\!\not- l_2 \equiv \Diamond l_1 \rightarrow \neg\Diamond l_2$$

Notice that this formula is not obtained by simply negating the LTL formula corresponding to the positive constraint (Equation 4.1). In that case the resulting formula would be:

$$\neg(\Diamond l_1 \rightarrow \Diamond l_2) \equiv \Diamond l_1 \wedge \neg\Diamond l_2$$

This interpretation would require that in every execution l_1 must be achieved sooner or later while l_2 must be invariably false. This formula results to be too strong because we lose the "conditional" behaviour of the constraint: if l_1 is achieved then l_2 must be invariably false.

LTL Interpretation. The LTL formula associated to "$\mathtt{dnf}_1$ *not correlates* $\mathtt{dnf}_2$" ($\mathtt{dnf}_1 \bullet\!\!\not- \mathtt{dnf}_2$) is:

$$\varphi_{ltl}(\mathtt{dnf}_1 \textit{not correlate}\ \mathtt{dnf}_2) = \Diamond\ \mathtt{dnf}_1\ \rightarrow\ \neg\Diamond\ \mathtt{dnf}_2 \tag{4.2}$$

The requirement it captures is: if there exist a state in the future at which $\mathtt{dnf}_1$ is achieved, then from now on there cannot exist a state in which $\mathtt{dnf}_2$ holds.

Co-existence

Requirement. This operator is derived from the correlation. Intuitively, it captures that if any of the two conditions is achieved sooner or later, then also the other must be achieved. Therefore, the way it could be obtained is simply by a conjunction of two correlation constraints, where conditions are switched.

LTL Interpretation. The LTL formula associated to "$\mathtt{dnf}_1$ *co-exist* $\mathtt{dnf}_2$" ($\mathtt{dnf}_1 \bullet\!\!-\!\!\bullet\ \mathtt{dnf}_2$) is based on correlation constraint, as follow:

$$\begin{aligned}\varphi_{ltl}(\mathtt{dnf}_1 \textit{co-exist}\ \mathtt{dnf}_2) =\\ \varphi_{ltl}(\mathtt{dnf}_1\ \textit{correlate}\ \mathtt{dnf}_2) \wedge \varphi_{ltl}(\mathtt{dnf}_2\ \textit{correlate}\ \mathtt{dnf}_1)\end{aligned} \tag{4.3}$$

Negative Co-existence

Requirement. As well as its positive form, the negative co-existence is a derived operator. It is obtained as the conjunction of two negative correlation constraints.

LTL Interpretation. The LTL formula associated to "$\mathtt{dnf}_1$ *not co-exist* $\mathtt{dnf}_2$" ($\mathtt{dnf}_1 \bullet\!\!\not-\!\!\bullet\ \mathtt{dnf}_2$) is based on negative correlation constraint, as follow:

$$\begin{aligned}\varphi_{ltl}(\mathtt{dnf}_1 \textit{not co-exist}\ \mathtt{dnf}_2) =\\ \varphi_{ltl}(\mathtt{dnf}_1\ \textit{not correlate}\ \mathtt{dnf}_2)\ \wedge\ \varphi_{ltl}(\mathtt{dnf}_2\ \textit{not correlate}\ \mathtt{dnf}_1)\end{aligned} \tag{4.4}$$

4.2.2 Temporal operators

Relation operators involve LTL temporal operators but they do not capture temporal relations on the achievement of the specified conditions. Temporal operators, instead, are defined so as to express some relative order on the achievement of different conditions.

Response

Requirement. The *response* operator is used to express constraints of the kind "if a certain condition is met then another condition must be achieved later, no matter if it has been true previously". This behaviour can be captured by checking, in every state, if the antecedent condition holds. If this is the case, then starting from that point the consequent condition must be achieved at least once in one of the subsequent states. The LTL formula that captures this aspect, on simple conditions, is the following:

$$l_1 \bullet\!\!\!\longrightarrow l_2 \equiv \Box(l_1 \to \Diamond l_2)$$

LTL Interpretation. The LTL formula, which allows to express more complex conditions, corresponding to "dnf_1 *response* dnf_2" ($\mathrm{dnf}_1 \bullet\!\!\!\rightarrow \mathrm{dnf}_2$) is:

$$\varphi_{ltl}(\mathrm{dnf}_1 \textit{response}\ \mathrm{dnf}_2) = \Box\ (\mathrm{dnf}_1 \ \rightarrow\ \Diamond\ \mathrm{dnf}_2) \tag{4.5}$$

According to this LTL formula, every time dnf_1 is achieved (that is, one of the conjuncts that compose it is satisfied) then there must exist a subsequent state in which dnf_2 is satisfied. The fact that dnf_2 was achieved in the past (before dnf_1) is not sufficient for making the constraint satisfied.

Negative Response

Requirement. The Negative form of response captures the requirement that if a certain condition is met, then from that point on another condition must be invariably false.

$$l_1 \bullet\!\!\!\not\rightarrow l_2 \equiv \Box(l_1 \rightarrow \neg\Diamond l_2)$$

The above formula is very similar to the formula of response, with the exception that the consequent condition is negated ($\neg\Diamond l_2$).

Also for negative response, the LTL translation is not obtained by simply negating the formula of the response constraint (Equation 4.5). This is because the formula would be too strong:

$$\neg(\Box(l_1 \rightarrow \Diamond l_2)) \equiv \Diamond(l_1 \wedge \neg\Diamond l_2)$$

It requires the existence of a state in which l_1 holds and, from that point on, l_2 is invariably false. There are two reasons why this formula does not correspond to the intuitive interpretation of the constraint. The first is that it requires l_1 to become true, thus losing any conditional behaviour. Secondly, it loses the characterization of a formula that must hold in every state (requirement captured by the external $\Box$ of our translation, given in Equation 4.6).

LTL Interpretation. The LTL formula corresponding to "dnf_1 *not response* dnf_2" ($\mathrm{dnf}_1 \bullet\!\!\!\not\rightarrow \mathrm{dnf}_2$) is:

$$\varphi_{ltl}(\mathrm{dnf}_1 \textit{not response}\ \mathrm{dnf}_2) = \Box\ (\mathrm{dnf}_1 \ \rightarrow\ \neg\Diamond\ \mathrm{dnf}_2) \tag{4.6}$$

The meaning of this formula is that in every state in which dnf_1 holds then from that point on dnf_2 must be invariably false.

Before

Requirement. The aim of the *before* operator comes quite straightforwardly from the name itself. The relation that it captures is that, in order for a certain condition to be achieved, another condition must be achieved before. This temporal relation is captured by means of the *LTL weak until* operator.

$$l_1 \multimap\!\!\bullet\ l_2 \equiv \neg l_2 \ \mathsf{U}\ l_1$$

The above formula requires l_2 to be false until l_1 is met. Moreover, according to weak until semantics once l_1 is achieved there are no requirements on the achievement of l_2. The formula does not require l_1 to be achieved either, but in case it never becomes true, also l_2 must be invariably false.

LTL Interpretation. The LTL formula corresponding to "dnf_1 *before* dnf_2" ($dnf_1 \multimap\!\!\bullet\ dnf_2$) is:

$$\varphi_{ltl}(dnf_1 \textit{before}\ dnf_2) = \neg\ dnf_2\ \ \mathsf{U}\ \ dnf_1 \tag{4.7}$$

It expresses that dnf_2 cannot be achieved at least until dnf_1 is met. Once this condition is satisfied, no matter whether dnf_2 becomes true or not.

Negative Before

Requirement. The negative form of before expresses that if a certain condition is met, then another condition cannot hold before. This "conditional" flavour is what distinguishes the negative before from positive before: negative before is not simply another way to express before (l_1 *not before* $l_2 \not\equiv l_2$ *before* l_1).

The desired behaviour of negative before is obtained by the following formula:

$$l_1 \not\multimap\!\!\bullet\ l_2 \equiv \Box(\Diamond l_2 \rightarrow \neg l_1)$$

This formula is checked in every state, and it requires that if in the future l_2 becomes true, then in the current state l_1 is false.

As for the others, also for this operator the LTL interpretation is not obtained by negating the positive formula of before (Equation 4.7). In that case the result would be[1]:

$$\neg(\neg l_2\ \mathsf{U}\ l_1) \equiv (\neg l_1)\ \mathcal{U}_S\ (l_2 \wedge \neg l_1)$$

1 Recall that $\neg(p\ \mathsf{U}\ q) \equiv (\neg q)\ \mathcal{U}_S\ (\neg p \wedge \neg q)$ [Hol03].

where U and U_S stand for weak and strong until respectively. The above formula requires l_1 to be false until in the state holds l_2 and does not hold l_1. Moreover, transforming the weak until into a strong until, it requires l_2 to become true. This requirement, however, is too strong for our intuitive interpretation of negative before.

LTL Interpretation. The LTL formula corresponding to "dnf_1 *not before* dnf_2" ($\mathrm{dnf}_1 \not\!-\!\!\triangleright\!\!\bullet\ \mathrm{dnf}_2$) is:

$$\varphi_{ltl}(\mathrm{dnf}_1 \textit{not before}\ \mathrm{dnf}_2) = \Box\,(\Diamond\ \mathrm{dnf}_2\ \rightarrow\ \neg\ \mathrm{dnf}_1) \tag{4.8}$$

Cause

Requirement. The requirement that a *cause* operator captures is that if a certain condition is achieved, then another condition must be achieved and this latter must hold only after the former. This requirement is basically obtained by merging the behaviour expressed by the *before* and by the *response* operators. In this way, *cause* inherits the temporal characterization of both.

$$l_1 \bullet\!\!-\!\!\triangleright\!\!\bullet\ l_2\ \equiv\ l_1 \bullet\!\!-\!\!\triangleright\ l_2\ \wedge\ l_1 -\!\!\triangleright\!\!\bullet\ l_2$$

LTL Interpretation. The LTL formula corresponding to "dnf_1 *causes* dnf_2" ($\mathrm{dnf}_1 \bullet\!\!-\!\!\triangleright\!\!\bullet\ \mathrm{dnf}_2$) is simply the conjunction of the formulas representing response and before.

$$\begin{aligned}&\varphi_{ltl}(\mathrm{dnf}_1 \textit{cause}\ \mathrm{dnf}_2) = \\ &\qquad \varphi_{ltl}(\mathrm{dnf}_1\ \textit{response}\ \mathrm{dnf}_2)\ \wedge\ \varphi_{ltl}(\mathrm{dnf}_1\ \textit{before}\ \mathrm{dnf}_2)\end{aligned} \tag{4.9}$$

If dnf_1 becomes true then dnf_2 must be achieved at least once later (response), and dnf_2 cannot be achieved before dnf_1 (before).

Negative Cause

Requirement. The negative form of the cause operator is derived from the negative before and the negative response. Therefore, its behaviour follows from that of their components: if a certain condition l_2 is met, then another condition l_1 cannot hold before (*not before*) and if the condition l_1 is met, then condition l_2 must be invariably false from that point on.

$$l_1 \bullet\!\!\not\!-\!\!\triangleright\!\!\bullet\ l_2 \equiv l_1 \bullet\!\!\not\!-\!\!\triangleright\ l_2 \wedge l_1 \not\!-\!\!\triangleright\!\!\bullet\ l_2$$

LTL Interpretation. The LTL formula corresponding to "dnf_1 *not cause* dnf_2" ($\mathrm{dnf}_1 \bullet\!\!\not\!\!\!\rightarrow\!\!\bullet\ \mathrm{dnf}_2$) is simply the conjunction of the negative response and negative before:

$$\varphi_{ltl}(\mathrm{dnf}_1 \textit{not cause}\ \mathrm{dnf}_2) = \\ \varphi_{ltl}(\mathrm{dnf}_1\ \textit{not response}\ \mathrm{dnf}_2) \wedge \varphi_{ltl}(\mathrm{dnf}_1\ \textit{not before}\ \mathrm{dnf}_2) \tag{4.10}$$

4.2.3 Strong sequence

Premise

Requirement. The *premise* is the 2CL operator that captures the strongest temporal relation between two conditions. In particular, it is used to express that if a certain condition is met, then another one must be true in the previous state. The way we express this condition is by means of the "next state" ($\bigcirc$) LTL operator:

$$l_1 \rhd\!\!\rhd\!\!- l_2 \equiv \Box(\bigcirc l_2 \rightarrow l_1)$$

This formula is composed of two LTL operators: the $\Box$, which state that the formula must be checked in every state, and the $\bigcirc$, which allows to express conditions concerning the next state w.r.t. the current one. In particular, this latter is used to capture that if in the subsequent state l_2 holds, then in the current one l_1 must be true.

LTL Interpretation. The LTL formula corresponding to "dnf_1 *premise* dnf_2" ($\mathrm{dnf}_1 \rhd\!\!\rhd\!\!- \mathrm{dnf}_2$) is:

$$\varphi_{ltl}(\mathrm{dnf}_1 \textit{premise}\ \mathrm{dnf}_2) = \Box\ (\bigcirc\ \mathrm{dnf}_2\ \rightarrow\ \mathrm{dnf}_1) \tag{4.11}$$

Negative Premise

Requirement. The negative form of premise aims at capturing that if a certain condition holds in a state, in the previous state another condition must be false. This is expressed in the following way.

$$l_1 \rhd\!\!\rhd\!\!\not- l_2 \equiv \Box(\bigcirc l_2 \rightarrow \neg l_1)$$

Again, this formula is not obtained by negating the positive formula of premise (Equation 4.11). In that case, indeed, the resulting formula would be:

$$\neg(\Box(\bigcirc l_2 \rightarrow l_1))\ \equiv\ \Diamond(\bigcirc l_2 \wedge \neg l_1)$$

that does not capture the intended meaning associated to this operator. The above formula, indeed, requires the existence of a state such that l_1 does not hold and in the subsequent state l_2 holds. It does not capture the condition that we meant to express with premise because it expresses a condition that must be achieved sooner or later (rather then a condition to be checked in every state), and it requires the achievement of l_2, losing the "conditional" characterization of the constraint.

LTL Interpretation. The LTL formula corresponding to "$\mathtt{dnf}_1$ *not premise* $\mathtt{dnf}_2$" ($\mathtt{dnf}_1 \not\twoheadrightarrow \mathtt{dnf}_2$) is:

$$\varphi_{ltl}(\mathtt{dnf}_1 \textit{ not premise } \mathtt{dnf}_2) = \Box\,(\bigcirc\,\mathtt{dnf}_2 \rightarrow \neg\mathtt{dnf}_1) \qquad (4.12)$$

4.3 2CL GENERALIZED COMMITMENT MACHINE

Having defined the LTL interpretation for 2CL constraints, we now have all the elements for defining a Generalized Commitment Machine for 2CL protocols. In the following, we will refer to them as 2CL-Generalized Commitment Machines (2CL-GCM). These are obtained as an extension of Singh's definition of Generalized Commitment Machines (GCM) [Sin07]. The idea is to exploit the mechanism provided by GCM for the inference of the possible transitions between the states and to enrich it with the evaluation of the protocol constraints. By accounting for both aspects, we result in providing a definition of when a sequence of states is a path of a given 2CL-GCM. Intuitively, a sequence of states is a path of a 2CL-GCM if it is a path according to the GCM inference mechanism and if it satisfies the LTL interpretation of constraints. In the following we will use the NetBill protocol (defined in Section 3.4.1) as an example.

4.3.1 GCM: Generalized Commitment Machine

In order to describe our extension, let us start by describing the functioning of Generalized Commitment Machines [Sin07]. The reader already familiar with them can skip this section.

Propositions

A GCM is defined on a set of propositions, capturing conditions of interest (e.g. the fact that a payment has occurred or that a request for quote has been made) and social relationships among the parties. The former are expressed as facts, while the latter as commitments. Moreover, true and false are assumed to be part of the set of propositions, representing respectively the *true* and the *false* values of propositional logic.

States A GCM features a set S of possible states, each of which is represented by a logical expression defined on a set of propositions. S represents the possible configurations of the social state throughout the possible interactions. A single state represents a snapshot taken at a particular moment of the interaction.

Example 4.1. *Considering NetBill protocol, the expression* $\mathsf{goods} \wedge C(\mathsf{c}, \mathsf{m}, \mathsf{pay})$ *represents one possible configuration of the social state, i.e. it is a state in* S*. This expression means that the goods were shipped and that there is a commitment from* c *(customer) to* m *(merchant) to pay for them. Another example is* $\mathsf{goods} \wedge \mathsf{pay} \wedge C(\mathsf{m}, \mathsf{c}, \mathsf{receipt})$*, meaning that goods were shipped, that payment occurred and that there is an active commitment from* m *to* c *to having a receipt sent.*

Good states Particularly relevant is the subset of S, whose elements are named *good states*: they are the desired final states of the interaction. The characterization of good states depends on the particular application. For instance, they may be only those that do not contain unsatisfied active commitments or those which satisfy a condition of interest (e.g. payment done and goods shipped).

Action theory In GCM, transitions between the states are logically inferred on the basis of an action theory Δ. It contains a set of axioms of the kind $p \overset{a}{\hookrightarrow} q$, meaning that q is a consequence of performing action a in a state where p holds. When q is false the meaning is that a is impossible if p holds. In general, Δ contains all the axioms deriving from the specification of the protocol actions. Additionally, Δ also contains an axiom for each action a and for each couple of states s and s' such that the execution of a in s causes a transition to s': for instance, if the precondition p of a is satisfied in s and its effect q is satisfied in s', then $s \overset{a}{\hookrightarrow} s'$ is in Δ. The way in which these axioms are obtained is by applying the inference rules defined in [Sin07] (and reported in Tables 4.2 and 4.3) to the axioms deriving from actions definition. The resulting set of axioms allows to infer all the possible transitions among the states in S: a transition between two states s and s' can be inferred if there is an axiom $s \overset{e}{\hookrightarrow} s'$ in Δ.

Example 4.2. *According to the* 2CL *protocol syntax, the action* sendAccept*, performed by the customer to accept a quote of the merchant, is defined as:*

$$\text{sendAccept } \textbf{\textit{means}} \text{ CREATE}(C(\mathsf{c}, \mathsf{m}, \mathsf{goods}, \mathsf{pay})) \textbf{\textit{ if}} \neg \mathsf{pay}.$$

The corresponding axiom is

$$\neg \mathsf{pay} \overset{sendAccept}{\hookrightarrow} C(\mathsf{c}, \mathsf{m}, \mathsf{goods}, \mathsf{pay}).$$

Consequence:	$\dfrac{p \vdash q \qquad q \overset{e}{\hookrightarrow} r \qquad r \vdash u}{p \overset{e}{\hookrightarrow} u}$
Conjunction:	$\dfrac{r \overset{e}{\hookrightarrow} p \qquad r \overset{e}{\hookrightarrow} q}{r \overset{e}{\hookrightarrow} p \wedge q}$
Consistency:	$\dfrac{p \overset{e}{\hookrightarrow} q}{\neg(p \overset{e}{\hookrightarrow} \neg q)}$
Inertia:	$\dfrac{p \overset{e}{\hookrightarrow} q \qquad q \nvdash \neg\alpha}{p \wedge \alpha \overset{e}{\hookrightarrow} q \wedge \alpha}$

Table 4.2: Inference rules for the definition of the action theory [Sin07]. $\vdash$ and $\equiv$ represents, respectively, the logical consequence and the logical equivalence of propositional logics.

Now, if one considers a state in which $\neg pay \wedge quote$ *holds, it is also possible to infer the following axiom, by applying the* consequence *rule reported in Table 4.2.*

$$\neg pay \wedge quote \overset{sendAccept}{\hookrightarrow} C(c, m, goods, pay).$$

Given the elements we have described, the definition of a Generalized Commitment Machine results to be:

Definition 4.9 (Generalized Commitment Machine). *Let* $\vdash$ *and* $\equiv$ *be, respectively, the logical consequence and the logical equivalence of propositional logic. A* 2CL-GCM *is a tuple* $\mathbf{P} = \langle S, A, s_0, \Delta, G, \rangle$, *where:*

- S *is a finite set of states represented as logical expressions;*
- A *is a finite set of actions;*
- $s_0 \in S$ *is the initial state;*
- Δ *is an action theory;*
- $G \subseteq S$ *is a set of good states;*

(i) *Members of S are logically distinct, that is:* $\forall s, s' \in S, s \not\equiv s'$; (ii) $\text{false} \notin S$; *and* (iii) $\forall s \in G, s' \in S : (s' \vdash s) \Rightarrow (s' \in G)$, *i.e. any state that logically derives a good state is also good.*

4.3.2 2CL-GCM: 2CL-Generalized Commitment Machine

The definition of a 2CL-Generalized Commitment Machine (2CL-GCM) is based on the definition of Generalized Commitment Machine

establish:	$\mathtt{true} \xhookrightarrow{create(x, C_{x,y}p)} C_{x,y}p$
resolve and remove:	$p \wedge C_{x,y}p \xhookrightarrow{discharge(x, C_{x,y}p)} \neg C_{x,y}p$
remove (by debtor):	$C_{x,y}p \xhookrightarrow{cancel(x, C_{x,y}p)} \neg C_{x,y}p$
remove (by creditor):	$C_{x,y}p \xhookrightarrow{release(y, C_{x,y}p)} \neg C_{x,y}p$
change creditor:	$C_{x,y}p \xhookrightarrow{assign(y,z, C_{x,y}p)} C_{x,z}p \wedge \neg C_{x,y}p$
change debtor:	$C_{x,y}p \xhookrightarrow{delegate(x,z, C_{x,y}p)} C_{z,y}p \wedge \neg C_{x,y}p$
detach:	$CC_{x,y}(q,r) \xhookrightarrow{detach(z, CC_{x,y}(q,r))} \neg CC_{x,y}(q,r) \wedge C_{x,y}r$
eliminate:	$CC_{x,y}(q,r) \xhookrightarrow{eliminate(z, CC_{x,y}(q,r))} \neg CC_{x,y}(q,r)$

Table 4.3: GCM: Operations on commitments. $C_{x,y}p$ is a short notation for $C(x,y,p)$ and $CC_{x,y}(q,r)$ is a short notation for $C(x,y,q,r)$.

(Definition 4.9), but it additionally accounts for the set of 2CL constraints that are part of a protocol.

Action Theory In order to provide the definition of a 2CL-GCM corresponding to a protocol let us star defining how we obtained the set Δ, i.e. the action theory responsible for the inference of the transitions among the states of the machine. The definition of Δ depends on the definitions of the protocol actions. In Definition 4.10, we consider E_F as a logical expression (possibly true) concerning facts only, and E_{op} as a logical expression (possibly true) concerning operations on commitments only. Moreover, $\vdash$ is the logical consequence of propositional logic.

Definition 4.10 (Action Theory of a Protocol Action)**.** *Let* s *and* s' *be two states represented as logical expressions. An axiom* $s \overset{a}{\hookrightarrow} s'$ *is an axiom of the action theory* Δ *of a protocol* $\mathcal{P} = \langle \mathsf{Ro}, \mathsf{F}, s_0, \mathsf{A}, \mathsf{Cst} \rangle$ *iff there exists a definition "*$(a$ **means** $E_F \wedge E_{op}$ **if** $\mathsf{Cond})$*" in* A *s.t.* $s \vdash \mathsf{Cond}$ *and:*

(a) $\forall e_{op}$ *s.t.* $E_{op} \vdash e_{op}$ *and given* $z \overset{e_{op}}{\hookrightarrow} z'$ *according to commitment operations' axioms (see Table 4.3) then if* $s \vdash z$ *then* $s' \vdash z'$*; and*

(b) $\forall e_F$ *s.t.* $E_F \vdash e_F$ *then* $s' \vdash e_F$ *and:*

(b.1) if $s \vdash C(x,y,e,e_F)$ *(with* e *possibly true) then* $s' \vdash \neg C(x,y,e,e_F)$ *and*

(b.2) if $s \vdash C(x,y,e_F,e')$ *then:*
if $s' \nvdash e'$ *then* $s' \vdash \neg C(x,y,e_F,e') \wedge C(x,y,e')$*; otherwise* $s' \vdash \neg C(x,y,e_F,e') \wedge \neg C(x,y,e')$.

The definition of the action theory reported above considers the possible effects of a protocol action. Specifically, the effects can be positive or negative literals. If they are negative they are handled as in GCM: when an action has an effect $e = \neg p$, its effect is to overturn the value of p in the state. For what concerns operations on commitments they can be seen as events that happens concurrently to the protocol action. Operations on commitments (*(a)* in Definition 4.10) are handled as specified in Table 4.3. Additionally, when an operation on a commitment fails because the preconditions for its success are not satisfied, we interpret it as the failure of the operation but not necessarily this represents a failure of the protocol action. Let us clarify this point by means of the Example 4.3

Example 4.3. *Let us consider the* CANCEL *operation. According to Table 4.3, in order to succeed the commitment should hold in the state before the execution of the operation and should not hold after. However, consider a protocol action that includes a cancel of a commitment in its definition. The way we interpret its execution is that if the commitment is present in the source state, then it should not hold in the target state. If the commitment is not present in the source state the action can succeed by causing the other effects. The mandatory requirement, even in this case, is that the commitment does not hold in the target state.*

By relying on the definition of the action theory of a protocol, we are now able to provide the definition of a 2CL-GCM corresponding to a protocol. The definition adopt the same notation of [Sin07] (reported in Section 4.3.1) but additionally considers the set Cst of 2CL constraints.

Definition 4.11 (2CL-GCM of a protocol)**.** *A* 2CL-GCM *of a protocol* $\mathcal{P} = \langle \mathsf{Ro}, \mathsf{F}, s_0, \mathsf{A}, \mathsf{Cst}\rangle$ *is a tuple* $\mathbf{P} = \langle \mathsf{S}, \mathsf{L}_\mathsf{A}, s_0, \Delta, \mathsf{G}, \mathsf{Cst}\rangle$ *where:*

- S *is a set of states represented as logical expressions;*
- L_A *is a set of actions s.t.* $a \in \mathsf{L}_\mathsf{A}$ *iff* $\exists a$ ***means*** E ***if*** $\mathsf{Cond} \in \mathsf{A}$ *;*
- $s_0 \in \mathsf{S}$ *and represents the initial state;*
- Δ *is an action theory s.t.* $\forall s, s' \in \mathsf{S}, s \overset{a}{\hookrightarrow} s' \in \Delta$ *iff there exists* a ***means*** E ***if*** $\mathsf{Cond} \in \mathsf{A}$ *s.t.* $s \overset{a}{\hookrightarrow} s'$ *is an action axiom of 'a' according to Definition 4.10;*
- $\mathsf{G} \subseteq \mathsf{S}$ *is a set of good states;*
- Cst *is a set of* 2CL *constraints.*

Moreover: (i) $\forall s, s' \in \mathsf{S}, s \not\equiv s'$, *i.e. members of* S *are logically distinct;* (ii) *false* $\notin \mathsf{S}$; *and* (iii) $\forall s \in \mathsf{G}, s' \in \mathsf{S} : (s' \vdash s) \Rightarrow (s' \in \mathsf{G})$, *i.e. any state that logically derives a good state is also good.*

Since the starting state s_0, and the set of constraints Cst of a 2CL-GCM are the same of the protocol, the definition uses the same symbols. By varying the sets S and G one obtains different 2CL-GCMs associated to the same protocol: when S contains all the states that can be reached from s_0, applying the protocol actions, the machine can infer all the possible interactions of the protocol. When S is smaller, only a subset of the possible interactions is determined.

Path of a 2CL-GCM

Interactions between agents can be seen as paths traversing states, the transitions among which are labelled by actions (events) that caused them. We denote a path τ as the sequence $\langle(\tau_0, a_0, \tau_1), (\tau_1, a_1, \tau_2), \dots\rangle$. In order for a path to be part of a 2CL-GCM it must respect some conditions:

(1) The path must be infinite[2];
(2) All the transitions of the path must be inferable by the machine; and
(3) All constraints must be satisfied in the path.

It is not restrictive to focus on infinite paths. Indeed, all finite paths can be transformed into infinite ones by adding a transition from the last state of the finite path towards an artificial new state with a self loop [Sin07]. In 2CL-GCM we assume that the action axioms that allow inferring such transitions are part of Δ.

2CL constraints verification can be done by exploiting the LTL formula associated to each of them. In particular, a constraint is satisfied in a path when it is verified in the *transition system* corresponding to the path. Given a path, the corresponding transition system can be derived quite straightforwardly. Intuitively, the set of states and transitions of the system is the same set of states and transitions in the sequence. A requirement on transition systems is that each state has at least one outgoing transition (i.e. runs are infinite). In Definition 4.12, $\vdash$ is the logical consequence of propositional logics.

Definition 4.12 (Transition System)**.** *A transition system* $T(\tau)$ *of a path* $\tau = \langle(\tau_0, a_0, \tau_1), (\tau_1, a_1, \tau_2), \dots\rangle$ *where* τ_i *is the state at position* i *in* τ *and* a_i *is the action that causes the transition from state* τ_i *to state* τ_{i+1}*, is a tuple* $\langle S_\tau, \delta_\tau, I_\tau\rangle$ *where:*

- $S_\tau = \{\tau_i | \ \tau_i$ *is a state in* $\tau\}$;
- $\delta_\tau : S_\tau \to S_\tau$ *is a transition function s.t.* $\delta_\tau(\tau_j) = \tau_k$ *iff* (τ_j, a, τ_k) *is in* τ*;*

2 Condition inherited from [Sin07]

- $l_\tau : S_\tau \to 2^F$ *is a labelling function, s.t.* F *is a set of facts and commitments and* $l_\tau(\tau_i) = \{l | \tau_i \vdash l\}$.

To define a 2CL-GCM path, we adapted the definition of GCM path by adding the requirement that the sequence of states satisfies all the constraints of the 2CL-GCM. This condition is checked on the transition system corresponding to the path, by means of the LTL satisfaction relation[BK08]. We denote it with the symbol $\models_{LTL}$. In the following definition we adopt the same notation in [Sin07].

Definition 4.13 (2CL-GCM path). *Let* $\mathbf{P} = \langle S, A, s_0, \Delta, G, Cst \rangle$ *be a* 2CL-GCM, $\tau = \langle (\tau_0, a_0, \tau_1), \dots \rangle$ *be an infinite sequence of triples and* $T(\tau)$ *be the corresponding transition system. Let* $\mathrm{inf}(\tau)$ *be the set of states that occur infinitely often in* τ. τ *is a path generated from* $\mathbf{P}$ *when:*

(i) $\forall (\tau_i, a_i, \tau_{i+1})$ *in* τ *we have that* τ_i *and* τ_{i+1} *belong to* S, $a_i \in A$, $\tau_i \overset{a_i}{\hookrightarrow} \tau_{i+1} \in \Delta$*; and*

(ii) $\mathrm{inf}(\tau) \cap G \neq \emptyset$*; and*

(iii) $\forall c \in Cst : \; T(\tau), \tau_0 \models_{LTL} \varphi_{ltl}(c)$.

In the above definition, *(i)* and *(ii)* are the conditions for a path to be generated from a GCM [Sin07]. Condition *(i)* requires that each *state* in the sequence is a state of the 2CL-GCM, that the *action* that causes the transition from a state to the subsequent one in the sequence is an action of the 2CL-GCM, and that the transition is inferable according to the *axioms* in Δ. Condition *(ii)* requires that at least one *good state* occurs infinitely often in the sequence. Condition *(iii)* was added to account for the evaluation of the protocol constraints. Since $T(\tau)$ is a transition system made of *one* linear path only (by construction), whose starting state is the starting state of τ, the condition $T(\tau), \tau_0 \models_{LTL} \varphi_{ltl}(c)$ amounts to checking that c is satisfied in the path of the transition system corresponding to τ.

Summary

The possibility of determining whether a path is legal according to a 2CL-GCM is important for different aspects. If we consider the 2CL-GCM corresponding to a 2CL protocol, indeed, it corresponds to determining whether a given interaction is legal or not and, in this latter case, which kind of requirement is violated (is it an unsatisfied commitment or is it consequence of the violation of a constraint?). The possibility of performing this kind of evaluation may be used by the agents to avoid sanctions. Specifically, a protocol draws the boundaries within which the interacting agents are free to decide how to behave. Crossing such boundaries (for instance violating a constraint) corresponds to an illegal behaviour that may result in a sanction for the agent. It is, therefore, important to an agent to understand which are the allowed interactions.

Besides being important for an interacting party, this kind of evaluation is important also for a protocol analyst (or designer) that has to understand a protocol specification and how it can be modified so as to specify different requirements (for instance more restrictive requirements). Chapters 6 and 7 describe these kinds of reasoning and present tools that support them as well as real case studies where such analysis are performed.

For completeness, in the following we report the modelling steps for the definition of a 2CL-GCM of the NetBill protocol described in Section 3.4.1. These steps are those defined in [Sin07] adapted and extended with the additional step that allows to introduce protocol constraints.

2CL-GCM of NetBill.

Atomic propositions. The first step is the definition of the involved atomic propositions, such as `quote`, `pay`, `goods` and so on. Roughly, this set must contains all the propositions that are used during the interaction and that can be asserted in the social state. Intuitively, they represent the agreed vocabulary that the interacting parties are able to understand.

Commitments. The second step is the identification of the set of commitments relevant for the interaction to be represented. These are the engagements that will tie the agents during the interaction. For instance, the set of commitments for the NetBill protocol is made of $C(m, c, C(c, m, goods, pay), goods)$, $C(m, c, pay, receipt)$ and $C(c, m, goods, pay)$.

Identify the states in S. As well as GCM [Sin07], also 2CL-GCM are thought to infer the possible transitions between a given set of possible configurations of the social state. Therefore, they are not meant to generate all the states that it is possible to obtain, starting from an initial state, by applying protocol actions. The identification of the set S of states to include is one of the modelling steps. Which states to consider depends on the protocol and on the kind of reasoning that must be performed on it. One task could be to determine if a given configuration of the social state is reachable from the initial state (i.e. if all the states given as input are reachable following some inference path). For instance, in the NetBill protocol one could consider the states $\{pay\}$, $\{goods\}$ and $\{receipt\}$ and determine if there exists a path that connects them, starting from the empty state.

Identify the actions in L_A. The NetBill protocol actions are those we have defined in Section 3.4.1. They are *send_request*, *send_quote*, *send_accept*, *send_goods*, *send_EPO* and *send_Receipt*.

Assert action theory Δ. The action theory Δ can be obtained by applying Definition 4.10.

For instance, the action `send_request` of NetBill has the following definition:

$$\mathit{send\_request}\ \textbf{means}\ \mathit{request}\ \textbf{if}\ \neg \mathit{quote} \wedge \neg \mathit{goods}$$

The axiom that can be derived directly from this definition is:

$$\neg \mathit{quote} \wedge \neg \mathit{goods} \overset{\mathit{send\_request}}{\hookrightarrow} \mathit{request}$$

meaning that from a state where $\neg \mathit{quote} \wedge \neg \mathit{goods}$ holds and another in which $\mathit{request}$ holds it is possible to infer a transition due to action `send_request`.

Another example is the action `send_accept`:

$$\mathit{send\_accept}\ \textbf{means}\ \textsc{create}(C(c, m, \mathit{goods}, \mathit{pay}))\ \textbf{if}\ \neg \mathit{pay}$$

that corresponds to the axiom:

$$\neg \mathit{pay} \overset{\mathit{send\_accept}}{\hookrightarrow} C(c, m, \mathit{goods}, \mathit{pay}).$$

Therefore, given a state in which $\neg \mathit{pay}$ holds and another in which $C(c, m, \mathit{goods}, \mathit{pay})$ holds it is possible to infer a transition due to action `send_accept`.

Notice that the operation create, effect of the action *send_accept*, has been applied according to the rules defined for the operations on commitments in [Sin07, Section 2.2] and reported in Table 4.3.

Staring from the action axioms it is possible to apply the inference rules (reported also in Table 4.2) on the set of state of the 2CL-GCM. All the axioms that it is possible to derive in this way are added to Δ.

Identify the good states in G. A state is good if it does not contain unsatisfied commitments.

Identify the set of constraints in Cst. In addition to the usual steps for GCM definition, the 2CL-GCM requires the specification, potentially empty, of a set of 2CL constraints. This set corresponds to the regulative specification of the protocol.

5 COMPARISON WITH THE LITERATURE

CONTENTS

In this chapter we discuss and compare the use of 2CL approach for protocol specification with other proposals in the literature. Particularly, we focus on the specification of patterns of interaction and the impact on the agents' autonomy and with the reuse of the specification.

5.1 COMPARISON WITH COMMITMENT-BASED APPROACHES

Commitments have been introduced for expressing a verifiable and flexible semantics of communicative acts. Commitment protocols aim at specifying interaction between agents maintaining such characteristics. In particular, for what concerns flexibility, they represent an alternative to procedural approaches, accused of being too rigid [YS02a] in the context of multiagent systems. However, focusing on the intent of providing a flexible specification of the interaction, commitment protocols completely disregard the possibility of expressing *patterns of interaction* (referred to as regulative specification in the following). These patterns are meant to capture requirements on *how* different conditions can be satisfied. In particular, Chopra and Singh [CS08, Cho09] introduce the distinction between constitutive and regulative specifications in the definition of commitment protocols but study in depth only the constitutive part. Each agent is publicly described by the *effects* of the actions it can execute. Such specifications allow agents to agree on the meaning of their actions. A similar approach can be found also in [YS01, YS02a], where the protocol is seen as a set of actions, whose semantics is given in terms of operations on commitments. By reasoning on the actions, agents can decide to undertake the behaviour that better suits their aim. Other approaches propose logical formalizations of commitments life cycle [GMS07, EMBD10, EMBD11]. In these proposals commitments are represented as modalities, thus allowing for the formalization of different properties on the interaction, such as fairness, liveness and the satisfaction of all the commitments.

All these approaches, however, *do not account* for regulative specifications, besides the one given by commitments. Our proposal aims at extending commitment protocols with the possibility of expressing patterns of interaction. With this aim we have defined a new language named 2CL (Constraints among Commitments Language). The main characteristics of our proposal are summarized in Figure 5.1. As depicted, a 2CL protocol specification is made of two components: the constitutive and the regulative specification. The former collects the set of actions and their definitions. Actions are defined in terms of operations on commitments. With regulative specification, instead, we refer to the regulative requirements on the interaction, that it is possible to represent beside those given by the regulative nature of commitments. The regulative specification is given in terms of 2CL constraints. These constraints are expressed in terms of conditions (facts and commitments) to be achieved, rather than directly on actions. This justifies the connection between constraints and commitments in Figure 5.1, and the absence of a direct connection between constraints and actions.

Constitutive and regulative decoupling

As we will see in the following comparisons, a greater decoupling between the constitutive and the regulative specifications allows to increase the reuse of the specification. This is an important aspect as witnessed by many authors [CS06, SMR$^+$08, Mon09]. In particular, the reuse of a specification in many cases represents a significant gain of costs, resources and improves the quality of the specification [Kan04]. A specification is said to be reusable if it can be easily *customized* and *adapted*. With customization we refer to the possibility of modifying an existing specification so as to handle requirements that may frequently change along time, or to add additional requirements that may arise. By adaptation, instead, we refer to the possibility of "migrating" a specification thought for a given context, to another (similar but possibly not identical) context, without needing to replace it completely [RBW07, SMR$^+$08].

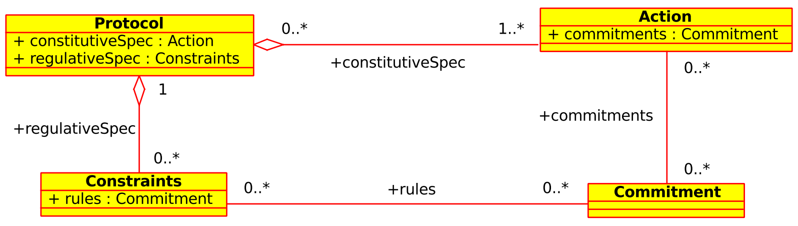


Figure 5.1: 2CL protocol specification: decoupling between constitutive (actions) and regulative (constraints) specifications.

In the 2CL approach for protocol specification, adaptation and customisation are supported by the declarative nature of the language and by the decoupling between the regulative and the constitutive components in the specification. In this way, when a requirement needs to be changed it is immediate the identification of the part of the specification that need to be modified. Moreover, the constitutive and regulative specifications can be changed independently one from the other. In this way it is possible, for instance, to add a new action (or a new constraint) without necessarily changing the set of constraints (or of actions). The declarative nature, instead, allows to specify just what is mandatory and what is forbidden: all that remains is unconstrained [PvdA06, Mon09, BBB$^+$11b]. This is a great advantage from the point of view of adaptation. If the specification is to be reused in a different context, indeed, it is sufficient to individuate the set of constraints that need to be changed in order to obtain a different behaviour. All the other parts of the specification remain unchanged. Procedural approaches, instead, would require to specify the new set of allowed interactions. This would probably means redefining the specification from scratch.

2CL *and the agent's autonomy*

2CL declarative nature is aligned with the intent of commitments to preserve the agent's autonomy of deciding how to behave in two different ways: (i) agents are always free to decide to violate a constraint (being aware of the possible consequences, like possible sanctions); (ii) declarative approaches specify only what is strictly necessary. For all that is not specified agents can behave as they wish.

The need of capturing patterns of interaction, more expressive than those that is possible to represent by means of commitments only, is witnessed also by other proposals in the literature. Despite the fact that in these proposals it is possible to recognize various attempts to capture both a constitutive and a regulative specification, they still miss the clear distinction postulated in [Sea69, Che73] and described in the 2CL approach. In particular, none of the proposals allow the specification of both parts (i) *in a way that allows flexible representations*, (ii) *by* *means of first-class languages*, (iii) *in a sufficiently decoupled way*: either one of the two specifications is disregarded or it is too strict or the two representations are to some extent mixed.

Let us, therefore, comment and compare some of these proposals.

5.1.1 Actions Preconditions

A Protocol specification defines the set of actions that agents can perform during the interaction. There are many definitions for actions in the literature. Most of the work on agents adopts a *precondition-effect* view. *Preconditions* can be of two kinds: preconditions to the action execution, and preconditions to some effect. The former are literals that must hold in the social state to make the action executable, the latter are additional conditions that, when holding, allow the production of the specific effect that they control.

Example 5.1. *In order to pay by credit card it is necessary to own a credit card (precondition to the execution of the action). If a credit card owner uses it for paying, the payment will be done only if the card is valid (conditional effect).*

By affecting the executability of the actions or the effects that they have, preconditions can be used as a way to obtain a desired ordering on the execution of the actions. Winikoff and colleagues [Win07, WLH05] propose the use of preconditions to the execution of the actions, as a means for a protocol designer to avoid reaching *undesirable* states. A similar approach is adopted in [FK04a, FK04b] and [CS06, KY09], where it is possible to specify both preconditions to the executability and to the effects.

Example 5.2. *In a protocol representing a democratic assembly it is possible to avoid the representation of the interaction in which a participant speaks without having obtained the floor by adding a proper precondition. An example is:*

start_talk ***means*** talk_started ***if*** has_floor

Limitations of the proposal

This solution has some limits. In particular, adding a precondition to an action allows to prevent its execution (or makes it to have no effect on the interaction) at design time. Different is the situation at *run-time*, where a precondition may be not sufficient to ensure the avoidance of undesired states or undesired behaviours during the interaction. The reason is simple: agents are autonomous and therefore it is not possible to check that their implementation of the actions respects the one prescribed by the protocol (in this case, that their implementation includes the specified preconditions). Without this verification, it

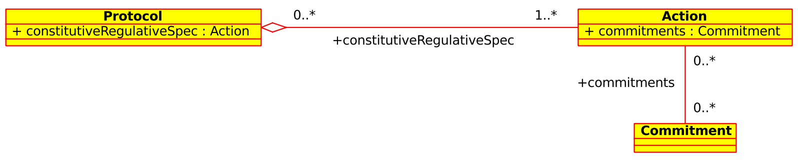

Protocol
+ constitutiveRegulativeSpec : Action
0..*
1..*
+constitutiveRegulativeSpec
Action
+ commitments : Commitment
0..*
+commitments
0..*
Commitment

Figure 5.2: Regulative specification based on actions definition.

is not possible to guarantee that actions will be executed only when the specification allows it (i.e. when preconditions are satisfied). For instance, going back to the Example 5.2, it is not possible to hush an agent that has not the floor: if it starts talking everyone can hear what it is saying, even if not allowed. The only way the functioning of preconditions can be guaranteed is through organisms as institutions [ERRAA04, ANRAS06] or organizations [ZJW03, MVB+04, BBvdT06, FHSB07]. These organisms implements the desired preconditions by means of proper artifacts [BBB+10, BBMP11, BBB+11a]. For instance, an institution can hush a not-allowed speaker by not passing on its message. Without this kind of mechanism, the designer must be very careful in representing only the actual behaviours of the agents, otherwise it could be the case that the model foresees less executions than those that are actually possible. Additional requirements on the interaction need to be represented in such a way that allows for the monitoring and the verification of the agents' behaviours [VS99, TCY+09]. In these cases, the designer is aware of the possible evaluation and can enact enforcement mechanisms.

Even assuming to be able to effectively realize a mechanism based on preconditions for expressing patterns of interaction, the specification still presents some limits. First, the use of preconditions forces the *regimentation* of the regulations [JS94]. It prevent, indeed, the execution of actions that would lead to a violation. The adoption of regimentation, rather than enforcement, is a choice that should be left up to the designer of the system and not be imposed by the specification framework [GAD07]. Second, and most importantly, it becomes more difficult to individuate the regulative aspect of the specification. More precisely, patterns of interactions are hidden inside actions definition. This aspect is graphically represented in Figure 5.2. It shows that the regulative and the constitutive specification are basically indistinguishable. This choice decreases the reuse of the specification, complicating its customization and adaptation. Suppose, indeed, that a regulation changes at a certain point. It could be hard to understand how it was implemented in terms of preconditions, and consequently, to determine where and how the specification should be changed.

Similar considerations hold for the proposal of adding ad-hoc commitments so as to mark undesirable final states [WLH05]. Also in this case, indeed, it is difficult to distinguish among commitments freely taken by the agents, and commitments added so as to label non final states.

Comparison

By means of 2CL it is possible to explicitly express those regulations that, if expressed as preconditions, are hidden inside actions definition. In this way, the designer is free to choose which among *enforcement* or *regimentation* better applies to the particular context.

Example 5.3. *The way it is possible to express that agents are not allowed to talk if they do not have the floor can be captured as a* 2CL *constraint in the following way:*

`has_floor` *before* `talk_started`

The designer may decide, if possible, to regiment this requirement in the system behaviour, or to monitor the interaction in order to verify its satisfaction.

Moreover, in a 2CL protocol the regulative component is clearly separated from the constitutive one (as depicted in Figure 5.1). In this way, regulations are easier to be understood and can be immediately identified and distinguished from the constitutive rules. As a consequence: *(i)* regulations are easier to be modified: in 2CL when new regulations are to be added it is sufficient to add a new set of constraints to the specification; and *(ii)* activities can be easily added: their definition can be simply included in the constitutive specification of the protocol; if they are to be interleaved with previously existing activities, it is sufficient to add proper constraints that capture when and how the added activities are to be used. These needs emerge clearly in many practical settings characterised by a high degree of regulation that changes along time (two practical examples are given in Chapter 7, where the specification of the Markets in Financial Instruments European Directive and the OECD Guidelines for private data protection are presented). The same easiness of adaptation and level of reuse cannot be reached in approaches based on the actions' preconditions, where regulations are not easy to be identified.

Finally, by means of 2CL it is possible to capture a wider range of regulations. When an action is executed, indeed, preconditions are able to capture conditions that must hold in the previous state. 2CL temporal constraints, instead, allow also to express relative orders on the achievement of two conditions, without specifying any number of steps within which the constraints must be satisfied. With 2CL is also possible to express relation constraints (without temporal requirements).

Imitate preconditions in 2CL

By means of 2CL it is possible to write constraints that imitate the effects of capturing regulative rules by means of preconditions. Specifically, since 2CL constraints are not expressed directly on actions, we first need to introduce for each action of the protocol a literal that is univocally associated to it (as an effect of the action). Then, we can use a constraint of kind *premise* among these literals.

Example 5.4. *Consider the actions* start_talk *and* open_debate. *The action* start_talk *is executable by the participant when the discussion has been opened (precondition). This requirement can be imitated by* 2CL *by giving the following definition of the actions:*

start_talk ***means*** talk_started
open_discussion ***means*** discussion_opened

and by adding the constraint:

discussion_opened *premise* talk_started

This constraint states that the condition talk_started *can be achieved only if in the previous state* discussion_opened *was true. In general, it is not possible to prevent a not allowed agent from speaking. In this case the constraint would be violated. However, the violation can be detected and the agent may be possibly sanctioned.*

5.1.2 Interaction Diagrams

For those who are familiar with UML, interaction diagrams are very similar to *Sequence Diagrams*. In particular, an interaction diagram «*specifies which actions can be performed by the agents at every stage of the interaction.*» ([FC03]) All the sequences that are not explicitly represented, are not allowed. For this reason, this kind of specification can be classified as *procedural*.

Interaction diagrams [BMO01, Fou03] have been used in the commitment-based approach by Fornara and Colombetti [For03, FC04a, FC04b]. In these works, the aim of the authors was the definition of a commitment-based semantics for the speech acts of agent communication languages, like FIPA ACL. Even not focusing on patterns representation, the authors propose the use of interaction diagrams for capturing the allowed sequences of messages. Figure 5.3 reports the graphical representation of this specification. In particular, the regulative specification is explicitly represented in the protocol (as a difference with approaches based on preconditions, see Figure 5.2).

Limits of the proposal

The choice of relying on interaction diagrams for patterns specification is very strong because it forces an ordering on the execution of the actions, thus losing the flexibility aimed at by the adoption of commitments and limiting the agents' autonomy of deciding how to behave.

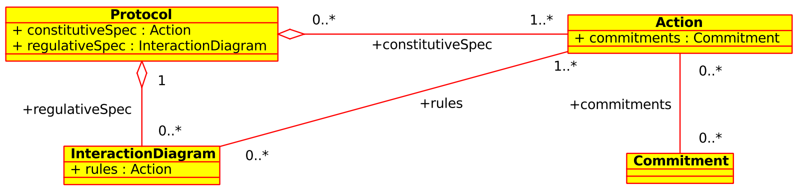


Figure 5.3: Regulative specification given by interaction diagrams. Sequences are given in terms of actions.

For what concerns reuse and understandability, as a first impression interaction diagrams seem to be quite intuitive because they allow to map the desired interactions straightforwardly on the allowed sequences of actions. This consideration, however, holds for simple contexts only: the more the number of actions is high and regulations are tangled, the more a procedural specification becomes difficult to manage. Imagine, indeed, to include in the specification an action that can be executed at any time. In interaction diagrams this can be represented by adding it as an alternative to every step allowed by the diagram. Another solution is to consider it as an "exception" with respect to the specified sequences. In this case it is handled and represented as a separate diagram. This, for instance, is the solution adopted by FIPA CNET for the *cancel* and the *not-understood* meta-protocols [Fou02c].

Similar considerations hold also for [CW05, CW09].

Comparison

2CL adopts a completely different approach to the specification of patterns. It, indeed, is a declarative language, thus allowing to specify only requirements that are strictly necessary, rather then enumerating the set of allowed sequences. This improves the reuse and the understandability of the specification. In 2CL, for all the aspects that are not captured by 2CL constraints the agents are free to decide how to behave. For instance, actions that can be executed at any time can be represented very easily, by simply adding them to the constitutive specification of the protocol (without adding constraints).

Thanks to the decoupling between the constitutive and the regulative specification, in 2CL protocols it is also possible to modify one of the two specifications without necessarily having to intervene also on the other. This entails that actions can be added without having to modify the protocol constraints in order to account for them. The greater decoupling of 2CL protocols can be seen also by comparing Figures 5.1 and 5.3. This latter presents a direct connection between the regulative specification and the actions. Interaction diagrams, indeed, are represented directly in terms of messages. This limits the protocol reuse. For instance, in order to add a new action to the protocol it is necessary to revise the specified sequences in order to include it. Similarly, when it is necessary to modify an existing action or an existing regulation, it is necessary to revise all the allowed sequences in order to determine how changed impact on them. This practically means that every time the specification is to be modified, the set of allowed sequences must be re-computed.

How to obtain the same effect with 2CL

Strict sequences of actions, similar to those captured by interaction diagrams, can be represented in 2CL protocols by using constraints of kind *premise*.

Example 5.5. *Suppose to have an interaction diagram that specifies the sequence* call_for_vote *and* vote*. In order to represent it, in* 2CL *specifications we first univocally associate a literal to the actions:*

call_for_vote ***means*** vote_open
vote ***means*** voted

Then, we add a constraint:

vote_open *premise* voted

This constraint expresses that in order for the condition voted *to be achieved, the condition* vote_open *should hold in the previous state.*

5.1.3 Precedence Logic

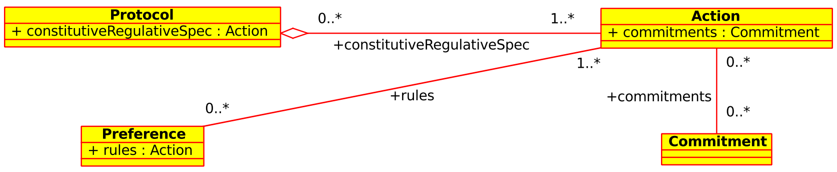
Precedence Logic has been introduced by Singh [Sin03b] and it basically corresponds to a propositional logic augmented with a temporal operator. This operator is named *before* and it is represented by the symbol '·'. It allows to capture "minimal ordering requirements" between events. By means of this operator it is possible to specify a set of sequences, named *dependencies*, in a declarative way. Every dependence captures a relative order between events. Everything that is not specified is left free to the agents.

Example 5.6. *The expression*

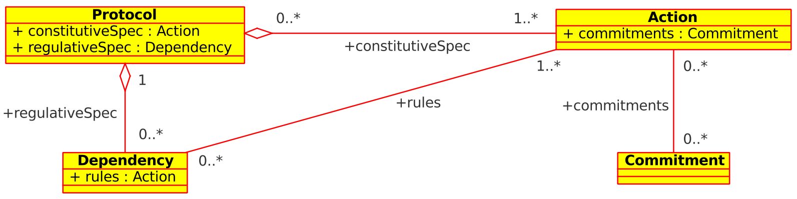
$call\_for\_vote \cdot vote$

captures that the actions call_for_vote *and* vote *must both be executed and in the specified order.*

Mallya and Singh [MS06] propose the use of the same logic for defining *preferences* on the execution of the actions. The set of preferences can be further ranked depending on how much it is desired that the specified order is respected. The main difference with dependences is that preferences do not express strict requirements. They are simply a preference criterion. For this reason, preferences can be interpreted more as guidelines than regulations.



(a) Mallya and Singh's model: adding preferences on actions.



(b) Singh's dependencies among events.

Figure 5.4: Patterns representation by means of Precedence Logic.

Figure 5.4 summarizes the approach used in these two proposals. The main difference between the two is that, since preferences do not have a regulative characterization, this proposal relies only on commitments for the regulative aspects. This characteristic can be graphically perceived from the fact that in Figure 5.4 (a) there is not a direct connection between the protocol and the preferences. In dependences approach (Figure 5.4 (b)), instead, dependences are used to capture regulative requirements. Therefore, in the protocol specification there is an explicit account for this regulative aspect, represented in the picture by a direct connection between protocol and dependences. In both approaches, precedence logics is used to capture ordering between events. This justifies the connection between dependences and preferences directly with actions.

Limits of the proposal

Since the kinds of rules expressed by the two proposals are very similar, they suffer of the same limits. In particular, giving regulations in terms of actions makes the specification less flexible and less easily adaptable or open [BBM10b, BBMP13]. For instance, when an action name changes it is also necessary to revise the set of constraints (thus introducing potential oversights). Another evidence is given in case new actions, with the same meaning of existing actions, are to be added to the specification. Let us see an example.

Example 5.7. *Suppose that payment should occur before sending the goods and that the protocol foresees the actions* pay-by-credit-card *and* send-goods. *The protocol specification would be:*

$$\text{pay-by-credit-card} \cdot \text{send-goods}$$

Now, if a client arrives who can pay cash, he/she will not be able to take part in the interaction unless the specification is changed by adding the action to the protocol and by changing the above rule to:

(pay-by-credit-card OR pay-by-cash) · send-goods

This should be done even though the actions pay-by-credit-card *and* pay-by-cash *have the same semantics in terms of commitments. In case regulations are given on literals rather than directly on actions, a new action can be added without having to modify the existing regulations.*

The need of modifying the specification even when actions have the same semantics (as in the example), gives *an undesired rigidity to the protocol*. Problems arise also in case an agent can execute a sequence of actions which *altogether* implement one of those foreseen by the protocol.

Comparison

2CL constraints relate commitments and not actions/events. This aspect emerges clearly in Figure 5.1, where the regulative specification made of constraints is not directly connected to the protocol actions. This modularity facilitates the design of interaction protocols because it allows for a separate specification of the actions and of the set of constraints. Moreover, it improves the re-use of previously defined actions or regulations as long as the domain of discourse does not change.

Example 5.8. *Let us reformulate the Example 5.7 by adopting* 2CL *protocol specification. At the beginning the protocol specification foresees the definition of the actions for paying by credit card and for sending the goods:*

pay-by-credit-card ***means*** *paid*
send-goods ***means*** *sent_goods*

and the regulative specification includes the constraint

paid *before* sent_goods

Now, if we want to introduce a new action by means of which the participant can pay by cash it is sufficient to add it to the constitutive specification.

pay-by-cash ***means*** *paid*

No changes are needed on the regulative specification since it already contains the necessary constraint for making payment occur before shipment.

The same flexibility of the specification cannot be reached with approaches that rule directly actions, like [Sin03b, MS06, Mon09].

Finally, 2CL constraints allow the specification of a wider range of constraints with respect to those that can be expressed by means of precedence logic. This latter, indeed, foresees the use of a “before” operator that is similar to the 2CL *cause* relation. The difference is that

2CL *cause* relation does not necessarily impose the achievement of the two conditions. The precedence logics operator, instead, imposes the achievement of both the conditions and in the specified order. The other 2CL operators cannot be represented (or do not find easy or immediate correspondence) in event logic.

How to obtain the same effect with 2CL

Also in this case, it is possible to imitate in 2CL the specification of constraints on actions, as already described for the approaches in Sections 5.1.1 and 5.1.2, and to imitate the *before* operator of precedence logic.

Example 5.9. *A constrain in precedence logic of the kind*

paid · sent_goods

can be represented in 2CL *with the constraint*

paid *cause* sent_goods

This constraint requires that if paid *is achieved then, and only after,* sent_goods *must be achieved. However,* 2CL *does not force the achievement of the condition* paid *or of* sent_goods*, but it checks that, if achieved, they are achieved in the right order.*

5.1.4 Regula: Commitments to Regulations

The Regula framework defined by Marengo *et al.* [MBB+11c] extends the kind of conditions that can be expressed in commitments, by allowing the definition of *commitments to regulations*. By including regulations inside commitments, the language allows to explicitly assign a responsible agent for a regulation: it coincides with the debtor of the commitment.

Example 5.10 ([MBB+11c])**.** *Consider a regulation saying that a physician's referral should precede a surgeon's procedure. In this case, it is not clear whether the physician is responsible for moving first or the surgeon is responsible for moving second. By placing the regulation inside the commitment one can explicitly represent who among the physician and the surgeon is responsible of the regulation.*

The Regula language relies on precedence logics for the representation of temporal conditions inside commitments: '$a \cdot b$' means that both events a and b must occur and event a before event b. Commitment life cycle is extended so as to include the notion of *progression* of commitments on temporal regulations. Progression basically relates the condition in the commitment and the happened events.

Since events involved in temporal conditions may depend on different agents, Marengo *et al.* [MBB+11c] introduce a notion of *safety*.

Safety allows agents to decide whether it is reasonable for an agent to adopt a certain commitment. An agent committed to a temporal regulation, indeed, is considered responsible for its satisfaction even in case it does not own the capabilities for satisfying it. The notion of safety relies on the notion of *control*: an agent controls an event (*innate* control) if it can bring it about; an agent controls a sequence of events '$e_1 \cdot \ldots \cdot e_n$' if it can bring about all the events in the sequence and in the given order. An agent is said to have *social* control over an event (or a sequence) if it has a commitment from someone which has the control over it. A commitment is *safe* for its debtor when it has established enough control over its consequent condition or it controls the negation of the antecedent condition. This latter eventuality allows the agent to make the commitment never be activated.

Example 5.11. *In the RONR example, given a regulation*

$$\mathsf{ask\_floor} \cdot \mathsf{give\_floor}$$

we can say that the participant has innate control *over the event* ask_floor, *while the chair has innate control over the event* give_floor.

A commitment of the kind $C_1 = C(p, ch, \mathsf{ask\_floor} \cdot \mathsf{give\_floor})$ *is not safe for the participant for two reasons: (i) it cannot make* give_floor *happens (it does not have control over it); (ii) it cannot be sure that the chair will wait for* ask_floor *to be true, before performing* give_floor.

Commitment C_1 *would become safe if a commitment of the kind* $C_2 = C(ch, p, \mathsf{ask\_floor} \cdot \mathsf{give\_floor})$ *holds. By means of it, indeed, the chair commits to respect the sequence. Thanks to* C_2*, the participant acquires the social control over the regulation* ask_floor · give_floor.

Comparison

A great advantage of expressing patterns of interaction inside commitments is that they acquire a deontic semantics given by commitments. As well as commitments to simple conditions, an agent is expected to satisfy the commitments it has taken, that is to achieve the specified conditions and in the given order. Moreover, by relying on commitments it is possible to explicitly individuate who to blame in case of a violation: the debtor of the commitment.

However, by relying on precedence logic, the Regula framework suffers of the same shortcomings described in Section 5.1.3. To briefly summarise them: (i) regulations are expressed in terms of events, thus decreasing the reuse of the specification; (ii) the kind of patterns that it is possible to express are limited. As already discussed, 2CL is more expressive thus allowing to capture a greater variety of patterns of interaction.

5.2 COMPARISON WITH NON COMMITMENT-BASED APPROACHES

Let us now discuss some differences between 2CL protocols and other approaches for protocol specification, not based on commitments. It is hard to be exhaustive, due to the interest that different authors have placed in investigating interaction and coordination mechanisms along the years. For this reason, in this section we focus on few proposals that, even not based on commitments, are closer to our approach. Specifically, we discuss the use of *expectations* which, as sated in [TCY+09], are orthogonal to commitments and *Declare*, from which we inspired for the definition of our language. The aim of the following discussions is to underline the differences among these approaches and our proposal.

5.2.1 Expectation-Based Approaches

The approaches based on expectations [ADT+04, CMMT09b, Mon09, MPvdA+10, MTC+10] allow to define relations between happened events and expectations on the occurrence of other events. Further requirements can be expressed on the time (absolute or relative) at which some events are expected to occur.

As noticed also in [TCY+09], commitment-based and expectation-based semantics are orthogonal since they face the interaction from different points of view: commitments adopt a state-oriented approach, while expectations adopt a rule-oriented approach. Also their regulative nature is different: expectations do not account for any responsible but they define a set of constraints (SICs) that rule the evolution of the interaction by stating which are the expectations to be satisfied. Commitments, instead, individuate a responsible of the conditions to be achieved. This is important for establishing who to blame when a certain condition is not achieved. 2CL constraints are similar to SICs in the aim of capturing a set of patterns on the interactions. However, the fact of being based on commitments allows 2CL to maintain the regulative aspect given by commitments.

Besides the differences between the two semantics, 2CL protocols and expectation-based approaches differ also in the representation of the protocol and of the patterns of interaction. Specifically, expectation based protocols include the definition of a knowledge base (expressing some knowledge about the domain) and a set of integrity constraints. These latter basically capture relations between different events that may or that are expected to happen. In this representation, therefore,

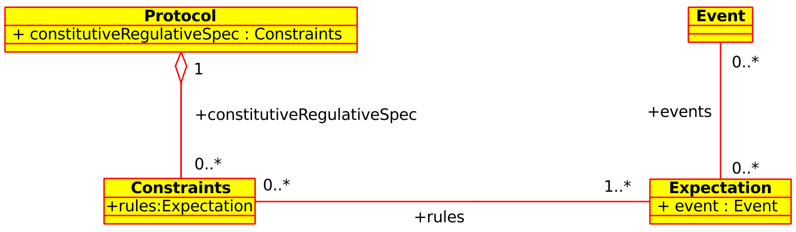


Figure 5.5: Expectation based model: regulative specification given by means of integrity constraints defined in terms of expectations.

the protocol specification does not foresees the definition of a meaning for the events that can occur during the interaction. The set of possible events can be obtained by considering those involved in the integrity constraints. These latter, indeed, are used to generate the set of possible executions, as in [MTC+10], or to verify that a given sequence of events respects the set of constraints. In other words, in this proposal there is not an explicit representation of the constitutive specification, that is hidden inside the regulative one (Figure 5.5). This aspect lowers the decoupling between the regulative and the constitutive specification, thus entailing the same shortcomings (related to flexibility and adaptability) already described for Precedence Logics approaches (see Section 5.1.3). Imagine, indeed, to account for a new event that may occur in the interaction or to change the requirements in order to adapt the specification to a different context. By analysing the integrity constraints it could be difficult to establish where to intervene to obtain the desired specification.

The lack of decoupling can be found also in Chesani *et al.* [CMMT09a] and Torroni *et al.* [TCMM09] proposals, where conditions inside commitments are expressed in terms of events. These approaches differ from 2CL also on the aim of the constraints that can be expressed. Specifically, they focus on *commitments tracking*, thus expressing conditions on the life cycle of commitments. For instance, they allow to express that a certain condition must be satisfied within a certain amount of time from its creation. The aim of 2CL constraints is different in the sense that it is meant to rule the evolution of the interaction rather than the evolution of the life cycle of a commitment. This is done by expressing (temporal) requirements on the *achievement of different conditions* (in terms of commitments and facts) rather than expressing relations between operations on commitments only.

5.2.2 Declare

Declare is a declarative language that allows to express a set of constraints between activities that can be performed in a business process.

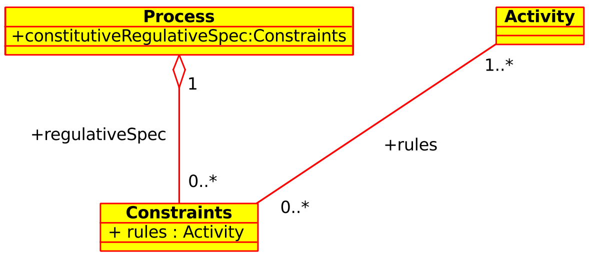


Figure 5.6: ConDec model: constitutive and regulative specification given by means of constraints on activities.

The operational semantics of a Declare process is the Büchi automaton that is obtained by translating constraints, applying well-known algorithms from the field of Model Checking [CGP01] and refined in [PBvdA10]. In this way it is possible to obtain the allowed sequences starting from a declarative representation.

Also in this approach there is not a clear distinction between the constitutive and the regulative specification. Constraints, indeed, have a constitutive nature, because the allowed behaviours are generated from them. The activities that are part of the process are only those that are part of the constraints specification (eventually specified as activities not connected to other activities in the constraint diagram). However, the allowed executions are generated taking into account the regulative requirements imposed by the constraints. And this is why they have also a regulative nature that, as depicted in Figure 5.6 is mixed with the constitutive one.

Even in the case where one uses the above model only with a regulative intent, other problems emerge, due to the fact that constraints are defined over activities (actions or events). Therefore, this proposal suffers of the same limits on the flexibility and adaptability of the specification, as the proposals described in Sections 5.1.3 and 5.2.1.

5.3 SUMMARY OF COMPARISONS

The need to represent patterns of interaction in protocol specification is an important aspect, as witnessed by the proposals we have discussed in this chapter. Figure 5.7 summarizes our analysis and the comparison with the 2CL approach. In particular, in this summary, we focus on how the different approaches represent the constitutive and the regulative specification, and whether these are decoupled.

We also consider the respect of the agent's autonomy (how much the specification left agents free to decide how to behave), and how easily the specification can be adapted and customized.

	Constitutive spec.	Patterns spec.	Decoupling	Agents' autonomy	Adaptation	Customization	Description
Action preconditions	Yes	No	No	No	No	No	Ad-hoc preconditions on actions
Interaction Diagrams	Yes	Yes	No	No	No	No	Interaction diagrams
Precedence Logic	Yes	Yes	No	Yes	½	½	Before relation between events
Expectation-based constraints	No	Yes	No	Yes	½	½	Declarative constraints expressed as expectations on events
2CL Protocol	Yes	Yes	Yes	Yes	Yes	Yes	Declarative specification of constraints among commitments and facts

Figure 5.7: Comparison between commitment-based approaches in the literature. The symbol ½ is used to express that the proposal partially faces a certain need, but not in a completely satisfactory way.

In this respect, the use of preconditions for capturing a desired ordering [WLH05, CS08], does not have a regulative characterization but, being part of the actions specification, they have a constitutive nature. Moreover, the agent's autonomy is not respected because they are not free to decide which action to execute and whether to satisfy or violate the regulation. These characteristics are also responsible for decreasing the understandability of the specification, the individuation of regulations and, consequently, the possibility of adapting or customising the specification.

Similar limitations can be ascribed to the use of interaction diagrams [For03]. This approach basically equals to enumerate all the allowed sequences (as for procedural approaches). This rigidity makes the specification difficult to be reused and customized, and clashes with the autonomy of the agents.

The use of precedence logic by Mallya and Singh [Sin03b, MS06] is better because it is declarative. For this reason the approach suits better the agents' autonomy, the adaptation and the customization of the specification, than the others. However, precedence relations are defined in terms of events (actions) and therefore constitutive and regulative specifications are still tied, thus it is difficult to modify them separately (this is the meaning of ½ in Figure 5.7: better than procedural approaches, but still not completely satisfactory).

Finally, in the expectation-based approach combined with the ConDec graphical language [Mon09, CMMT09b, MPvdA$^+$10], constraints are defined directly on actions (events). As we have described this impacts on the adaptation and the customization. Moreover, events are not associated to a meaning or to a definition in the specification. For this reason, protocols do not foresee a constitutive specification. The events that can occur during an interaction can be deduced from the set of constraints.

2CL constraints preserve the flexibility that is typical of commitment protocols because, in general, they do not force agents to execute given paths but rather let them freely choose their courses of actions. They are also free to violate a constraint. In this case, however, they take the risk to be punished (we do not treat sanctions or punishments in this thesis). The explicit representation of declarative patterns and the choice of specifying constraints on facts and commitments, rather than directly on actions, allows to increase the separation between constitutive and regulative rules in the specification. This separation brings many advantages, mostly as direct effects of the obtained *modularity*: *easier re-use* of actions in different contexts, *easier understanding* of the regulations, *easier customization* of the protocol, *easier composition, adaptation and extension* of protocols (see Chapter 7). As a consequence, MAS would gain greater *openness*, *interoperability*, and *modularity of design*. Interoperability would be better supported because it would be possible to verify it with respect to specific aspects (e.g. at the level of actions [CS08, Cho09, CS09b] or at the level of regulation rules [BBM11a]). Similarly, agents can check whether they are able to conform to the protocol, by considering separately the constitutive and the regulative specifications [BBM11a]. Finally, protocols would be more open in the sense that their modularity would allow designers to easily adapt them to different needs (see Section 3.4.3).

6 A TOOL FOR 2CL PROTOCOL VISUALIZATION

CONTENTS

In this chapter we present a tool for 2CL protocol visualization. It allows exploring all the possible executions of an interaction protocol, showing the *regulative violations*, i.e. both those states in which some constraint is violated and those that contain unsatisfied commitments. This tool can be profitably used as a means for the *analysis of the risks* the interaction could encounter. Examples of this analysis are provided by means of two case studies, presented in Chapter 7. As a consequence of this analysis, it would be possible to reduce risks by defining proper operational strategies, like regimentation (aimed at preventing the occurrence of violations) or enforcement (introduction of warning mechanisms) [JS94].

We show that the presented tool is correct w.r.t. 2CL-GCM (presented in Chapter 4).

6.1 STATE EVALUATION OF 2CL CONSTRAINTS

The implementation that we present is realized in *tuProlog* [tup] and it builds upon the *enhanced commitment machine* realized by Winikoff *et al.* [WLH05]. By relying on it, we inherit the mechanisms for the computation of the possible interactions. Specifically, the enhanced commitment machine features the generation of the reachable states, the transitions among them and the management of commitments (like the operations of discharge, creation and so on). Our extension [BBC$^+$12] equips it with the possibility of evaluating 2CL constraints.

Overall view of possible interactions

The main characteristic of our tool is that it provides an overall graphical view of the possible interactions, highlighting those that will bring to a violation and those that will not. To this aim, constraints are used as a means to classify the possible interactions, rather than to prune the search space. The interacting parties, indeed, are not prevented from entering in illegal paths (due to the agent's autonomy), but they are made aware of the risks they are encountering and that they may incur in penalties as a consequence of the violations they caused [BBMP11]. This is a difference compared with those proposals where only the set of legal paths is shown, or with other proposals that aim at properties verification. In these cases, the verification ends when a path that does not satisfy the property is found. Alternatively, only one path at a time is considered [CMMT09a, EMBD11]. Thus, none of these proposals provide an overview of possible interactions.

Reachable states and state evaluation

Given a protocol specification, our implementation is able to determine the corresponding graph of reachable states. The computation of the reachable states is done in the same way as [WLH05], thus as a depth-first search. Specifically, given a state the program finds the set of applicable actions and computes the set of successors. A successor is added to the graph only if it is new, otherwise only the transition is added. For what concerns the evaluation of protocol constraints, we implemented it as a *state evaluation*, that is to say that given a constraint its evaluation is done considering its content only. In this way, each possible state (reachable given the starting state and the protocol actions) is considered only once and it is classified as a state of violation if some constraint is violated in it or as a legal state when no constraint is violated. This is a great difference with respect to path evaluation, where a state belonging to different paths can be classified as a state of violation or not depending on the path that is considered. The advantage is practical: given the set of reachable states, the user is able to immediately determine which of them are legal and which violate some constraints. Moreover, the overall representation results to be more compact because each state appears only once.

In order to perform the *state evaluation* we consider states whose content is given in terms of commitments and positive facts only. The characteristic of a fact is that it is false until it becomes invariably true. In this setting, the evaluation of 2CL constraints can be made on single states. For instance, if in a state 'b' holds but 'a' does not, we can infer that the constraint 'a *before* b' is violated. Moreover, besides facts asserted by the protocol actions, in our implementation we additionally consider a set of facts associated to the operations performed on commitments. Specifically, along the line of Mallya *et al.* [MS05], whenever an operation is performed on a commitment, a corresponding predicate is automatically asserted in the state. For instance, when a commitment $C(x, y, r, p)$ is created, the predicate $\textsc{created}(C(x, y, r, p))$ is added to the state and when the commitment is discharged, the predicate DISCHARGED$(C(x, y, r, p))$ is added, and so forth for the other operations. Notice that these predicates are not meant to express whether a commitment is active or not. For instance, CREATED does not mean that the commitment is active in the state but simply that the corresponding operation has been performed on the commitment. 2CL constraints can be defined by considering these facts also.

In order to achieve the benefits of a state evaluation while guaranteeing the soundness of the verification with the theoretical framework presented in Chapter 4, we need to make some assumptions on the way protocols are specified:

1. Actions should be defined in such a way to do not retract facts;
2. The condition involved in constraints must involve conditions that persist (i.e. that involve DNFs of facts without negation), and constraints cannot be of kind *premise*;
3. Constraints expressed on commitments are to be opportunely transformed into constraints concerning operations preformed on commitments.

Assumptions on Actions

More in details, for what concerns actions, we consider only those definitions that are given in terms of operations on commitments and in terms of facts to be added to the state (this requirement is similar to the definition of *Monotonicity* given in [Win07]). Negation can be used in actions' definition for the specification of preconditions. These conditions will be verified as negation as failure to verify whether a predicate is present or not in a state and, consequently, whether the effects of the action can be applied or not.

Assumptions on Constraints

For what concerns constraints, we basically require them to be defined on positive facts or on facts representing operations that have been performed on commitments. This choice is motivated by the fact that it allows for a correct state evaluation of constraints. However, the impossibility of capturing conditions on commitments would represent a restriction too strong for a commitment-based approach. Commitments, indeed, are central to our proposal. The possibility of expressing conditions on the operations that are performed on them (e.g. CREATED, DISCHARGED) represents a *compromise* that allows on the one hand to perform state evaluation, and on the other hand to express requirements on commitment manipulation. Those specifications that include constraints on commitments can be adapted rewriting the constraints in terms of the special predicates we automatically assert. We can also envisage to perform it automatically, by translating each condition on commitments into a condition on the creation of the commitments. However, we decided not to do so because other translations are possible. Depending on the intended meaning of the original constraint, for instance, a condition on a commitment can be associated with the moment it is created, with the moment it is discharged and so on.

Example 6.1. *Consider constraints* (c2) *and* (c3) *of NetBill protocol specification, given in Section 3.4.1:*

(c2) $\mathsf{C}(m, c, pay, receipt) \wedge goods$ *before* pay
(c3) pay *cause* $receipt$

According to Winikoff et al.*'s implementation constraint* (c2) *is satisfied when before the payment occurs, (i) the merchant sends the goods and (ii) the merchant takes the commitment of sending the receipt* or *directly sends the receipt. Therefore, according to this interpretation a possible translation of constraint* (c2) *could be:*

(c2') $(\textsc{created}(\mathsf{C}(m, c, pay, receipt)) \vee receipt) \wedge goods$ *before* pay

Constraint (c3)*, however, requires that the receipt is sent only after the payment. Therefore, the actual behaviour captured by constraint* (c2) *is that the merchant takes the commitment of sending the receipt before the payment. This can be captured, in terms of facts only, by defining the constraint in terms of the operation* create *performed on the commitment, rather than directly on the commitment:*

(c2') $\textsc{created}(\mathsf{C}(m, c, pay, receipt)) \wedge goods$ *before* pay

The fact $\textsc{created}(\mathsf{C}(m, c, pay, receipt))$ *is automatically asserted when the merchant creates the commitment. In this way, constraint* (c2') *captures the desired requirement.*

In order to intuitively explain the impact of our restriction let us consider an example. Let us consider the constraint '$\mathsf{C}(x, y, p) \wedge q$ *correlate* r' and its translation in terms of positive predicates: '$\textsc{created}(\mathsf{C}(x, y, p)) \wedge q$ *correlate* r'. The difference between the two is that the first constraint is activated if there exists a state in which both the commitment and the predicate q hold. The second constraint, instead, does not require the commitment and the predicate to hold in the same state. It is activated in every path in which the commitment is created (even if later cancelled or discharged) and the predicate becomes true. Roughly, the second constraint captures a weaker condition.

Another assumption we make on protocol constraints is that they are not defined in terms of *premise* (both in its negative and positive form). Intuitively, indeed, for *premise* we are not able to perform state evaluation since it involves considering two subsequent states: the current state and the subsequent.

We named *state constraints* the constraints that respect the requirements described above. These are the kind of constraints we are able to evaluate.

State constraint

Definition 6.1 (State constraint)**.** *Let* $c = \mathtt{dnf}_1 \ \mathtt{op} \ \mathtt{dnf}_2$ *be a constraint.* c *is a state constraint when:*

- $\mathtt{op}$ *is a* 2CL *operator except premise; and*
- *conditions* $\mathtt{dnf}_1$ *and* $\mathtt{dnf}_2$ *are given in terms of positive predicates only.*

Given the assumptions that allow for a correct state evaluation of protocol constraints, let us describe how we implemented it.

6.1.1 Labelled Graph of Possible Interactions

Given the assumptions on the protocol specification described above, we are able to compute the set of reachable states (given the protocol actions) and to label each of them by means of a state evaluation of protocol constraints. The result is a labelled graph of the possible interactions, where labels allow to distinguish legal states from those that violate some constraints. They allow also to determine if in a state there are active commitments not yet fulfilled.

Let us start by describing how we implemented 2CL constraints state evaluation. We named *state formula* (*sf*) the formulas corresponding to 2CL operators, but whose verification can be done on the content of a state, independently from the path it belongs to.

State formula

Definition 6.2 (State Formula)**.** *Let* c *be a state constraint.* $sf(c)$ *is the state formula of* c*. State formulas of* 2CL *constraints are reported in Table 6.1.*

Each state formula is meant to be verified in every state of a path, in order to determine if it represents a legal or illegal interaction. In the following we describe the intuition behind state formulas and how we handle them.

Before state formula

Consider a before constraint of the kind '$\mathtt{dnf}_1$ *before* $\mathtt{dnf}_2$'. It requires that $\mathtt{dnf}_1$ is met before or in the same state of $\mathtt{dnf}_2$. So, given a run π, if in π there is a state such that $\mathtt{dnf}_2$ holds while $\mathtt{dnf}_1$ does not, that is a state where a violation occurred. This amounts to check that, in every state, the state formula $\neg\mathtt{dnf}_2 \vee \mathtt{dnf}_1$ is satisfied. In case a state does not satisfy it, then it can be labelled as a state of violation.

$$sf(\mathtt{dnf}_1 \ \textit{before} \ \mathtt{dnf}_2) \ = \neg\mathtt{dnf}_2 \vee \mathtt{dnf}_1 \tag{6.1}$$

	Constraint	*State Formula sf*$(\mathtt{c})$
Correlation	$\mathtt{dnf}_1$ *correlate* $\mathtt{dnf}_2$	$\neg\mathtt{dnf}_1 \vee (\mathtt{dnf}_1 \wedge \mathtt{dnf}_2)$
	$\mathtt{dnf}_1$ *not correlate* $\mathtt{dnf}_2$ *	$\neg(\mathtt{dnf}_1 \wedge \mathtt{dnf}_2)$
Co-existence	$\mathtt{dnf}_1$ *co-exist* $\mathtt{dnf}_2$	*sf*$(\mathtt{dnf}_1$ *correlate* $\mathtt{dnf}_2) \wedge$ *sf*$(\mathtt{dnf}_2$ *correlate* $\mathtt{dnf}_1)$
	$\mathtt{dnf}_1$ *not co-exist* $\mathtt{dnf}_2$	*sf*$(\mathtt{dnf}_1$ *not correlate* $\mathtt{dnf}_2) \wedge$ *sf*$(\mathtt{dnf}_2$ *not correlate* $\mathtt{dnf}_1)$
Response	$\mathtt{dnf}_1$ *response* $\mathtt{dnf}_2$	$\neg\mathtt{dnf}_1 \vee (\mathtt{dnf}_1 \wedge \mathtt{dnf}_2)$
	$\mathtt{dnf}_1$ *not response* $\mathtt{dnf}_2$ *	$\neg(\mathtt{dnf}_1 \wedge \mathtt{dnf}_2)$
Before	$\mathtt{dnf}_1$ *before* $\mathtt{dnf}_2$ *	$\neg\mathtt{dnf}_2 \vee \mathtt{dnf}_1$
	$\mathtt{dnf}_1$ *not before* $\mathtt{dnf}_2$ *	$\neg(\mathtt{dnf}_1 \wedge \mathtt{dnf}_2)$
Cause	$\mathtt{dnf}_1$ *cause* $\mathtt{dnf}_2$	*sf*$(\mathtt{dnf}_1$ *response* $\mathtt{dnf}_2) \wedge$ *sf*$(\mathtt{dnf}_1$ *before* $\mathtt{dnf}_2)$
	$\mathtt{dnf}_1$ *not cause* $\mathtt{dnf}_2$	*sf*$(\mathtt{dnf}_1$ *not response* $\mathtt{dnf}_2) \wedge$ *sf*$(\mathtt{dnf}_1$ *not before* $\mathtt{dnf}_2)$

Table 6.1: State Formulas corresponding to 2CL operators. Each formula is meant to be verified in all the states of a path but it can be evaluated by considering one state at a time. Formulas whose corresponding operators are labelled with * immediately lead to a *violation*. The other formulas lead to a *pending condition*.

Correlation state formula

A constraint of kind '$\mathtt{dnf}_1$ *correlate* $\mathtt{dnf}_2$' requires that if $\mathtt{dnf}_1$ is achieved in a run, then also $\mathtt{dnf}_2$ is achieved in the same run (before or after $\mathtt{dnf}_1$ is not relevant). This can be simplified by noticing that when $\mathtt{dnf}_1$ is met, it must not be the case that all the subsequent states satisfy $\mathtt{dnf}_1$ but not $\mathtt{dnf}_2$ (since predicates are never retracted, if $\mathtt{dnf}_2$ is achieved before $\mathtt{dnf}_1$ it will remain true until $\mathtt{dnf}_1$ is achieved). Therefore, the condition to be checked in a state is that either condition $\mathtt{dnf}_1$ is false, or that condition $\mathtt{dnf}_1 \wedge \mathtt{dnf}_2$ holds.

$$sf(\mathtt{dnf}_1 \textit{ correlate } \mathtt{dnf}_2) = \neg\mathtt{dnf}_1 \vee (\mathtt{dnf}_1 \wedge \mathtt{dnf}_2) \qquad (6.2)$$

The states that do not satisfy this condition, however, cannot be immediately labelled as states of violation. The constraint, indeed, does not require $\mathtt{dnf}_2$ to hold *whenever* $\mathtt{dnf}_1$ holds. It only requires $\mathtt{dnf}_2$ to hold before the end of the interaction. Therefore, a state in which $\mathtt{dnf}_1 \wedge \neg\mathtt{dnf}_2$ holds is a state in which there is a *pending* condition. It will be considered a state of violation if the interaction does not continue after it. In this case, indeed, certainly condition $\mathtt{dnf}_2$ will not be achieved.

Response state formula

Considerations similar to *correlate* hold for the *response* constraint. Consider '$\mathtt{dnf}_1$ *response* $\mathtt{dnf}_2$'. It requires that when $\mathtt{dnf}_1$ is met, $\mathtt{dnf}_2$ is achieved at least once later (even if it already occurred in the past).

Since predicates that are asserted in the state cannot be retracted, if dnf_2 is achieved before dnf_1 it will be true also in the state and also after the achievement of dnf_1. Therefore, as well as correlation, this condition amounts to check that in a state either dnf_1 does not hold, or that $\mathrm{dnf}_1 \wedge \mathrm{dnf}_2$ holds.

$$sf(\mathrm{dnf}_1 \textit{ response } \mathrm{dnf}_2) = \neg \mathrm{dnf}_1 \vee (\mathrm{dnf}_1 \wedge \mathrm{dnf}_2) \tag{6.3}$$

All the states that satisfy $\mathrm{dnf}_1 \wedge \neg \mathrm{dnf}_2$ are marked as states where there is a pending condition.

Negative operators state formula

The intuition behind the negative operators is the same. Intuitively, a constraint of the kind dnf_1 *not correlate* dnf_2 requires that if dnf_1 holds, dnf_2 is not achieved. Since predicates on which constraints are defined cannot be retracted, this amounts to check whether the two conditions hold in the same state. For each state in which the condition $\mathrm{dnf}_1 \wedge \mathrm{dnf}_2$ holds we can state that there is a violation. Therefore, the state formula corresponding to *not correlate* is the following.

$$sf(\mathrm{dnf}_1 \textit{ not correlate } \mathrm{dnf}_2) = \neg(\mathrm{dnf}_1 \wedge \mathrm{dnf}_2) \tag{6.4}$$

Negative response and *negative before* add a *temporal aspect* to *not correlate*. Negative response requires that if dnf_1 holds then dnf_2 does not hold later. But since once a predicate is asserted it is not successively retracted, when dnf_2 starts to hold it will remain true also after the achievement of dnf_1. Thus, in those states in which both conditions are satisfied we can say that there is a violation.

$$sf(\mathrm{dnf}_1 \textit{ not response } \mathrm{dnf}_2) = \neg(\mathrm{dnf}_1 \wedge \mathrm{dnf}_2) \tag{6.5}$$

Negative before requires that if dnf_2 is achieved then dnf_1 does not hold before. If dnf_1 and dnf_2 are achieved in this order, then when dnf_2 is achieved also condition dnf_1 holds (since predicates are not retracted). Thus, in this state we can detect the violation by checking the condition $\mathrm{dnf}_1 \wedge \mathrm{dnf}_2$. As the other negative operators, these states are marked as states of violation.

$$sf(\mathrm{dnf}_1 \textit{ not before } \mathrm{dnf}_2) = \neg(\mathrm{dnf}_1 \wedge \mathrm{dnf}_2) \tag{6.6}$$

Derived operators.

Derived operators are evaluated in the states by evaluating the two constraints they are defined on. Therefore, for derived operators we apply the simplifications of the operators we have already described.

For what concern *cause* constraints, in each state we evaluate the corresponding *before* and *response* constraints that make the *cause*.

$$sf(\mathtt{dnf}_1\ \textit{cause}\ \mathtt{dnf}_2) = sf(\mathtt{dnf}_1\ \textit{before}\ \mathtt{dnf}_2) \wedge sf(\mathtt{dnf}_1\ \textit{response}\ \mathtt{dnf}_2) \tag{6.7}$$

If a state does not satisfy the response component of the cause, it is marked as PENDING; if it violates the before component, it is marked as a "violation". Both labels are applied when the state does not satisfy any of the two.

Similarly, the *co-exist* constraint is obtained from the state formulas of the *correlate* constraints that compose it:

$$sf(\mathtt{dnf}_1\ \textit{co-exist}\ \mathtt{dnf}_2) = sf(\mathtt{dnf}_1\ \textit{correlate}\ \mathtt{dnf}_2) \wedge sf(\mathtt{dnf}_2\ \textit{correlate}\ \mathtt{dnf}_1) \tag{6.8}$$

We apply the same mechanisms for the negative forms of derived operators.

Summary Summarizing, in our implementation given a state constraint and a state in which to verify it there are three possibilities:

(i) the state satisfies the constraint;

(ii) the state does not satisfy the constraint and this leads to a violation;

(iii) the state does not satisfy the constraint but the violation is potential, depending on future evolution (there is a pending condition to satisfy).

Considering all the constraints of a protocol, a state can both violate some constraints and have pending conditions. Besides classifying a state according to the constraints it satisfies or violate, we also consider the presence of unsatisfied commitments. We say that a state containing unsatisfied active commitments is *non-final*.

Labelled Graph According to these evaluations we are able to generate the labelled graph of the possible interactions. Specifically, given the protocol actions we are able to determine the graph of reachable states. Given a state the following states that are reachable from it by applying the protocol actions are computed applying the actions effects and by manipulating commitments according to commitments life cycle. In other words, we implemented the commitment axioms of [Sin07, Section 2.2] reported in Table 4.3. According to the evaluations we have described, to each state we associate a set of labels by considering violations, pending conditions and the unsatisfied commitments. Definition 6.3 reports the definition of the labelled graph. For the sake of simplicity, in the definition we consider E_F as a logical expression (possibly true) concerning facts only and E_{op} as a logical expression (possibly true) concerning operations on commitments only. In this way, an action definition can be written as (a **means** $E_F \wedge E_{op}$ **if** Cond). In the definition, $\vdash$ and $\equiv$ are respectively the logical consequence and the logical equivalence of propositional logics.

Definition 6.3 (Labelled Graph)**.** *Let* A *be a set of actions such that each action does not retract predicates. Let* Cst *be a set of state constraints. Given a constraint* $c \in \mathsf{Cst}$*, let* $sf(c)$ *be the state formula corresponding to* c*. Let* $\mathcal{P} = \langle \mathsf{Ro}, \mathsf{F}, s_0, \mathsf{A}, \mathsf{Cst} \rangle$ *be a protocol.* $\mathsf{G}(\mathcal{P}) = (\mathsf{S}, \delta, \mathrm{I})$ *is the labelled graph of protocol* $\mathcal{P}$ *where:*

- S *is a set of states represented as logical expressions such that* $\forall s, s' \in \mathsf{S}, s \not\equiv s'$*;*
- $\delta \subseteq \mathsf{S} \times \mathsf{A} \times \mathsf{S}$ *is a transition relation such that* $\forall (s, a, s') \in \delta$ *then* $s, s' \in \mathsf{S}$ *and* $\exists a \in \mathsf{A}$ *of the form* (a **means** $E_F \wedge E_{op}$ **if** Cond) *s.t.* $s \vdash \mathrm{Cond}$ *and:*
  (1) $\forall e_{op}$ *s.t.* $E_{op} \vdash e_{op}$
    - *if* $e_{op} = \textsc{create}(C(x, y, e, e'))$ *(with* e *possibly true) then*
      · $s' \vdash C(x, y, e, e') \wedge \neg e \wedge \neg e'$ *or*
      · $s' \vdash \neg C(x, y, e, e') \wedge e'$ *or*
      · $s' \vdash \neg C(x, y, e, e') \wedge e \wedge \neg e' \wedge C(x, y, e')$
    - *if* $e_{op} = \textsc{cancel}(C(x, y, e, e'))$ *then* $s' \vdash \neg C(x, y, e, e')$
    - *if* $e_{op} = \textsc{release}(C(x, y, e, e'))$ *then* $s' \vdash \neg C(x, y, e, e')$

  (2) $\forall e_F$ *s.t.* $E_F \vdash e_F$ *then* $s' \vdash e_F$ *and:*
    (2.1) *if* $s \vdash C(x, y, e, e_F)$ *(with* e *possibly true) then* $s' \vdash \neg C(x, y, e, e_F)$ *and*
    (2.2) *if* $s \vdash C(x, y, e_F, e')$ *then:*
      - *if* $s' \nvdash e'$ *then* $s' \vdash \neg C(x, y, e_F, e') \wedge C(x, y, e')$*;*
      - *otherwise* $s' \vdash \neg C(x, y, e_F, e') \wedge \neg C(x, y, e')$*.*
- $\mathrm{I} \subseteq \mathsf{S} \times 2^{\{\mathsf{F} \cup \{\textsc{pending}, \textsc{violation}, \textsc{final}, \textsc{non-final}\}}$ *is a labelling relation such that given* $s \in \mathsf{S}$*:*
  - F *is a set of predicates and commitments and given* $l \in \mathsf{F}$*,* $l \in \mathrm{I}(s)$ *if* $s \vdash l$*;*
  - $\textsc{violation} \in \mathrm{I}(s)$ *iff* $\exists c \in \mathsf{Cst}$ *s.t.* $s \nvdash sf(c)$ *and* c *is not a response or a correlation;*

- PENDING $\in I(s)$ *iff* $\exists c \in \mathsf{Cst}$ *s.t.* $s \nvDash sf(c)$ *and* c *is a response or a correlation;*
- FINAL $\in I(s)$ *iff there are no unsatisfied active commitments in* s*;*
- NON-FINAL $\in I(s)$ *iff* s *contains unsatisfied active commitments.*

Basically, the graph that we obtain is made of a set of states, represented as logical expressions, a transition relation δ, that accounts for the actions definitions and for the commitment life cycle, and a labelling function I. This latter associates to a state the set of predicates and commitments holding in the state, and a set of labels determined according to the evaluation of protocol constraints and to the presence or absence of active commitments. For what concerns δ, the interpretation that we adopt is similar to that given for 2CL-GCM in Section 4.3.2. Specifically, the interpretation of operations performed on commitments is that in case they fail, because preconditions are not satisfied, then the protocol action does not necessarily have to fail. Consider, for instance, an action that has as an effect, the cancellation of a commitment. In case it is applied in a state in which a commitment does not hold we can say that the operation of CANCEL on the commitment fails, but the action does not. This only in case the commitment does not hold in the target state, and the other effects of the action are achieved (in accordance to the definition as well).

Legal paths Given a labelled graph we can say that a path is legal if it is a path of the graph, if it does not contain states that violate some constraints and if it does not contain pending conditions or active commitments in what is considered as its last state. This is captured by the following definition.

Definition 6.4 (Legal path)**.** *Let* $G(\mathcal{P}) = (S, \delta, I)$ *be a labelled graph,* $\pi = \langle(\pi_0, a_0, \pi_1), \dots, (\pi_{n-1}, a_{n-1}, \pi_n)\rangle$ *be a path of at least one state.* π *is a legal path of* $G(\mathcal{P})$ *when:*

(i) $\forall i, 0 \leqslant i < n$ π_i *is a state of the graph and* $(\pi_i, a_i, \pi_{i+1}) \in \delta$.

(ii) $\nexists i \geqslant 0$ *such that* $\pi_i \in \pi$ *and* VIOLATION $\in I(\pi_i)$*;*

(iii) PENDING $\notin I(\pi_n)$ *and* FINAL $\in I(\pi_n)$.

Given a legal path π, of a labelled graph produced as described, we can prove its soundness w.r.t. the 2CL-GCM built on the same protocol specification. This actually corresponds to prove that π is a path of the 2CL-GCM. These aspects are investigated in the following section.

6.2 SOUNDNESS OF THE IMPLEMENTATION

In the previous sections we have described the assumptions we made on the characteristics of protocol specifications that we consider and how we implemented constraints evaluation as state formulas. In order to prove the soundness of our implementation, the first step is to prove that the state evaluation of constraints, that our program implements, is correct with respect to the LTL interpretation of 2CL constraints φ_{ltl}, given in Chapter 4 (see Definition 4.8). Given a constraint c, however, the corresponding state formula $sf(c)$ cannot be directly compared with the formula $\varphi_{ltl}(c)$. This latter, indeed, refers to a path, while $sf(c)$ concerns one state only. Our aim, instead, is to check the formula on one state at a time but verifying it in all the states of a given path. To this aim we define φ_{sf} as the LTL interpretation of a state formula. Roughly, since state formulas are formulas to be verified in all states, the corresponding LTL interpretations $\varphi_{sf}(c)$ are obtained by adding the *always* ($\Box$) operator in front of each state formula. For instance, the state formula of a before constraint is: $sf(dnf_1$ *before* $dnf_2) = (\neg dnf_2 \lor dnf_1)$. The corresponding LTL interpretation is: $\varphi_{sf}(dnf_1$ *before* $dnf_2) = \Box(\neg dnf_2 \lor dnf_1)$. The only exceptions are the *response* and the *correlate* operators. For these constraints the LTL interpretation is $\varphi_{sf}(c) = \Box(\neg dnf_1 \lor \Diamond(dnf_1 \land dnf_2))$, thus involving also the *eventually* operator. This formula captures a condition to be verified in all states ($\Box$ operator), but such that if dnf_1 holds then, in order for the formula to be satisfied, there must exist a state where $dnf_1 \land dnf_2$ holds.

Definition 6.5 (State formulas LTL interpretation)**.** *Let* c *be a state constraint and let* $sf(c)$ *be the corresponding state formula.* $\varphi_{sf}(c)$ *is the LTL interpretation of the state formula* $sf(c)$*. LTL interpretations for states formulas are reported in Table 6.2.*

In order to prove that the verification of state constraints that we implemented (φ_{sf}) is correct w.r.t. to the LTL interpretation of 2CL constraints (φ_{ltl}), we first have to define a model that respects the restriction we impose on the protocol specification. Specifically, we define a *factual model* as a model where an atomic proposition is false until it becomes invariably true. In our implementation, indeed, we assume that once asserted predicates cannot be retracted by protocol actions.

Definition 6.6 (Factual Model)**.** *Let AP be a set of atomic propositions. Let* $M = (S, \delta, I)$ *be an LTL model (as defined in Definition 4.2), where* $I : S \rightarrow 2^{AP}$*. Let* $s \in S$ *be a state.* M *is a* factual model *iff for all atomic propositions* $f \in AP$ *then* $M, s \models_{LTL} \neg f \; U \; \Box f$.

By relying on factual models and on state constraints only (see Definition 6.1), we are able to prove the soundness of our verification, as stated in Property 6.1, by proving the equivalences reported in Table 6.2.

Correlation	*Constraint*	$c = \mathrm{dnf}_1$ *correlate* dnf_2
	LTL Interpretation	$\varphi_{ltl}(c) = \Diamond \mathrm{dnf}_1 \to \Diamond \mathrm{dnf}_2$
	State Formula Interp.	$\varphi_{sf}(c) = \Box(\neg \mathrm{dnf}_1 \vee \Diamond(\mathrm{dnf}_1 \wedge \mathrm{dnf}_2))$
	Constraint	$c = \mathrm{dnf}_1$ *not correlate* dnf_2
	LTL Interpretation	$\varphi_{ltl}(c) = \Diamond \mathrm{dnf}_1 \to \neg\Diamond \mathrm{dnf}_2$
	State Formula Interp.	$\varphi_{sf}(c) = \Box\neg(\mathrm{dnf}_1 \wedge \mathrm{dnf}_2)$
Co-existence	*Constraint*	$c = \mathrm{dnf}_1$ *co-exist* dnf_2
	LTL Interpretation	$\varphi_{ltl}(c) = \varphi_{ltl}(\mathrm{dnf}_1$ *correlate* $\mathrm{dnf}_2) \wedge$ $\varphi_{ltl}(\mathrm{dnf}_2$ *correlate* $\mathrm{dnf}_1)$
	State Formula Interp.	$\varphi_{sf}(c) = \varphi_{sf}(\mathrm{dnf}_1$ *correlate* $\mathrm{dnf}_2) \wedge$ $\varphi_{sf}(\mathrm{dnf}_2$ *correlate* $\mathrm{dnf}_1)$
	Constraint	$c = \mathrm{dnf}_1$ *not co-exist* dnf_2
	LTL Interpretation	$\varphi_{ltl}(c) = \varphi_{ltl}(\mathrm{dnf}_1$ *not correlate* $\mathrm{dnf}_2) \wedge$ $\varphi_{ltl}(\mathrm{dnf}_2$ *not correlate* $\mathrm{dnf}_1)$
	State Formula Interp.	$\varphi_{sf}(c) = \varphi_{sf}(\mathrm{dnf}_1$ *not correlate* $\mathrm{dnf}_2) \wedge$ $\varphi_{sf}(\mathrm{dnf}_2$ *not correlate* $\mathrm{dnf}_1)$
Response	*Constraint*	$c = \mathrm{dnf}_1$ *response* dnf_2
	LTL Interpretation	$\varphi_{ltl}(c) = \Box(\mathrm{dnf}_1 \to \Diamond \mathrm{dnf}_2)$
	State Formula Interp.	$\varphi_{sf}(c) = \Box(\neg \mathrm{dnf}_1 \vee \Diamond(\mathrm{dnf}_1 \wedge \mathrm{dnf}_2))$
	Constraint	$c = \mathrm{dnf}_1$ *not response* dnf_2
	LTL Interpretation	$\varphi_{ltl}(c) = \Box(\mathrm{dnf}_1 \to \neg\Diamond \mathrm{dnf}_2)$
	State Formula Interp.	$\varphi_{sf}(c) = \Box\neg(\mathrm{dnf}_1 \wedge \mathrm{dnf}_2)$
Before	*Constraint*	$c = \mathrm{dnf}_1$ *before* dnf_2
	LTL Interpretation	$\varphi_{ltl}(c) = \neg\ \mathrm{dnf}_2\ \ \mathsf{U}\ \ \mathrm{dnf}_1$
	State Formula Interp.	$\varphi_{sf}(c) = \Box\neg(\mathrm{dnf}_2 \wedge \neg \mathrm{dnf}_1)$
	Constraint	$c = \mathrm{dnf}_1$ *not before* dnf_2
	LTL Interpretation	$\varphi_{ltl}(c) = \Box\ (\Diamond\ \mathrm{dnf}_2\ \to\ \neg\ \mathrm{dnf}_1)$
	State Formula Interp.	$\varphi_{sf}(c) = \Box\neg(\mathrm{dnf}_1 \wedge \mathrm{dnf}_2)$
Cause	*Constraint*	$c = \mathrm{dnf}_1$ *cause* dnf_2
	LTL Interpretation	$\varphi_{ltl}(c) = \varphi_{ltl}(\mathrm{dnf}_1$ *before* $\mathrm{dnf}_2) \wedge$ $\varphi_{ltl}(\mathrm{dnf}_2$ *response* $\mathrm{dnf}_1)$
	State Formula Interp.	$\varphi_{sf}(c) = \varphi_{sf}(\mathrm{dnf}_1$ *before* $\mathrm{dnf}_2) \wedge$ $\varphi_{sf}(\mathrm{dnf}_2$ *response* $\mathrm{dnf}_1)$
	Constraint	$c = \mathrm{dnf}_1$ *not cause* dnf_2
	LTL Interpretation	$\varphi_{ltl}(c) = \varphi_{ltl}(\mathrm{dnf}_1$ *not before* $\mathrm{dnf}_2) \wedge$ $\varphi_{ltl}(\mathrm{dnf}_2$ *not response* $\mathrm{dnf}_1)$
	State Formula Interp.	$\varphi_{sf}(c) = \varphi_{sf}(\mathrm{dnf}_1$ *not before* $\mathrm{dnf}_2) \wedge$ $\varphi_{sf}(\mathrm{dnf}_2$ *not response* $\mathrm{dnf}_1)$

Table 6.2: LTL Interpretations of 2CL constraints and of 2CL constraints state formulas. $\varphi_{ltl}(c)$ is the LTL interpretation of constraint c. $\varphi_{sf}(c)$ is the LTL interpretation of a state formula associated to constraint c.

Property 6.1. *Let* M *be a factual model. Let* π *be a path of* M *(according to Definition 4.3). Let* c *be a state constraint (according to Definition 6.1). Let*

$\varphi_{ltl}(c)$ *be the LTL interpretation of* c. *Let* $\varphi_{sf}(c)$ *be the LTL interpretation of sf*(c). $\pi \models_{LTL} \varphi_{ltl}(c)$ *iff* $\pi \models_{LTL} \varphi_{sf}(c)$

Before giving the proof of Property 6.1 let us introduce some Lemmas. Specifically, it is possible to prove that a conjunctive normal form formula of positive atomic propositions is false until it becomes invariably true (Lemma 6.1). Also a disjunctive normal form formula, where each conjunct is given in terms of positive atomic propositions, is false until it becomes invariably true in the model (Lemma 6.2).

***Lemma* 6.1.** *Let* $M = (S, \delta, I)$ *be an LTL model. Let* s *be a state in* S. *Let* $cf = f_1 \wedge \dots \wedge f_i \wedge \dots f_n$ *be a conjunctive formula of atomic propositions. If* M *is a factual model then* $M, s \models_{LTL} \neg cf \ \ U \ \ \Box cf$.

In the proofs below we adopt the logical consequence of propositional logic ($\vdash$). Notice that conjunctive formulas are formulas of atomic propositions. Therefore, verifying them by means of the satisfaction relation $\models_{LTL}$ amount to verify them on the first state of the path at hand.

***Proof* 6.2.1.** *Being* $M = (S, \delta, I)$ *a factual model, then* $\forall f \in AP \ M, s \models_{LTL} \neg f \ \ U \ \ \Box f$. *That by definition is* $\forall \pi^s$, *where* π *is a path of* M $\pi^s \models_{LTL} \neg f \ U \ \Box f$.

Given a conjunctive formula $cf = f_1 \wedge \dots \wedge f_i \wedge \dots \wedge f_n$ *and a path* π^s *starting at* s *then* $\pi^s \models_{LTL} \neg \Diamond cf \vee \Diamond cf$.

If $\pi^s \models_{LTL} \neg \Diamond cf$ *is the case then the thesis is proved.*

If $\pi^s \models_{LTL} \Diamond cf$ *is the case then, by definition,* $\exists j \geqslant s \ \pi^j \models_{LTL} cf$. *Let us consider* j *as the smallest index that satisfies the condition. Therefore,* $\pi^j \models_{LTL} f_1 \wedge \dots \wedge f_i \wedge \dots \wedge f_n$. *By hypothesis,* $\forall j' \geqslant j$ *and* $\forall f_i, cf \vdash f_i \ \pi^{j'} \models_{LTL} f_i$. *Thus* $\forall j' \geqslant j \ \pi^{j'} \models_{LTL} cf$. *This, by definition, entails that* $\pi^j \models_{LTL} \Box cf$. *Being* j *the smallest index such that* $\pi^j \models_{LTL} cf$, *it entails that* $\exists f_i, cf \vdash f_i$ *and* $\pi^{j-1} \models_{LTL} \neg f_i$. *By hypothesis,* $\forall k, s \leqslant k < j, \pi^k \models_{LTL} \neg f_i$. *Thus* $\forall k, s \leqslant k < j \ \pi^k \models_{LTL} \neg cf$ *and this proves the thesis.*
$\Box$

***Lemma* 6.2.** *Let* $M = (S, \delta, I)$ *be an LTL model. Let* s *be a state in* S. *Let* $dnf = cf_1 \vee \dots \vee cf_i \vee \dots cf_n$ *be a disjunctive formula where each conjunct is given in terms of positive atomic propositions. If* M *is a factual model then* $M, s \models_{LTL} \neg dnf \ \ U \ \ \Box dnf$.

***Proof* 6.2.2.** *Given a disjunctive formula* $dnf = cf_1 \vee \dots \vee cf_i \vee \dots cf_n$ *and a path* π^s *starting at* s *then* $\pi^s \models_{LTL} \neg \Diamond dnf \vee \Diamond dnf$.

If $\pi^s \models_{LTL} \neg \Diamond dnf$ *is the case then the thesis is proved.*

If $\pi^s \models_{LTL} \Diamond dnf$ *is the case then, by definition,* $\exists j \geqslant s \ \pi^j \models_{LTL} dnf$. *Let* j *be the smallest index such that* $\pi^j \models_{LTL} dnf$. *This means that* $\exists cf_i \ cf_i \vdash$

dnf *such that* $\pi^j \models_{LTL} \mathrm{cf}_i$. *By Lemma 6.1 and being* j *the smallest state that satisfies* dnf, *then* $\forall k, s \leqslant k < j\ \pi^k \models_{LTL} \neg\mathrm{dnf}$ *and* $\forall k' \geqslant j\ \pi^{k'} \models_{LTL} \mathrm{dnf}$ *i.e.* $\pi^j \models_{LTL} \Box\mathrm{dnf}$. *This ends the proof.*

$\square$

Now we can prove Property 6.1. The proof is given by cases on the kind of constraint.

***Proof* 6.1.1.**

***Case Before* (⇒)** Given $\pi \models_{LTL} \neg\ \mathrm{dnf}_2\ \ \mathsf{U}\ \ \mathrm{dnf}_1$ we have to prove that it implies $\pi \models_{LTL} \Box\neg(\mathrm{dnf}_2 \wedge \neg\mathrm{dnf}_1)$.

By definition of until (U) LTL operator we have that: $\exists j \geqslant 1\ \pi^j \models_{LTL} \mathrm{dnf}_1$ and $\forall k, 1 \leqslant k < j\ \pi^k \models_{LTL} \neg\mathrm{dnf}_2$; or $\pi \models_{LTL} \Box\neg\mathrm{dnf}_2$. If $\pi \models_{LTL} \Box\neg\mathrm{dnf}_2$ is the case than it implies $\pi \models_{LTL} \Box(\neg\mathrm{dnf}_2 \vee \mathrm{dnf}_1)$. Otherwise, $\exists j \geqslant 1\ \pi^j \models_{LTL} \mathrm{dnf}_1$. Let us consider j as the smallest index in π such that $\pi^j \models_{LTL} \mathrm{dnf}_1$. We have that $\forall k, 1 \leqslant k < j\ \pi^k \models_{LTL} \neg\mathrm{dnf}_2$ that implies the condition $\forall k, 1 \leqslant k < j\ \pi^k \models_{LTL} (\neg\mathrm{dnf}_2 \vee \mathrm{dnf}_1)$. By Lemma 6.2 we have that $\forall i \geqslant j\ \pi^i \models_{LTL} \mathrm{dnf}_1$ thus implying $\forall i \geqslant j\ \pi^i \models_{LTL} (\neg\mathrm{dnf}_2 \vee \mathrm{dnf}_1)$. Thus we have proved that the condition $(\neg\mathrm{dnf}_2 \vee \mathrm{dnf}_1)$ holds in all states, that is $\pi \models_{LTL} \Box(\neg\mathrm{dnf}_2 \vee \mathrm{dnf}_1)$, that equals to $\pi \models_{LTL} \Box\neg(\mathrm{dnf}_2 \wedge \neg\mathrm{dnf}_1)$.

***Case Before* (⇐)** Given $\pi \models_{LTL} \Box\neg(\mathrm{dnf}_2 \wedge \neg\mathrm{dnf}_1)$ we have to prove that it implies $\pi \models_{LTL} \neg\ \mathrm{dnf}_2\ \ \mathsf{U}\ \ \mathrm{dnf}_1$.

There are two cases.

Case 1. $\pi \models_{LTL} \neg\Diamond\mathrm{dnf}_2$. In this case dnf_2 never holds and the desired consequence trivially holds. By definition of U (until), if dnf_2 never holds then $\neg\ \mathrm{dnf}_2\ \ \mathsf{U}\ \ \mathrm{dnf}_1$ holds.

Case 2. $\pi \models_{LTL} \Diamond\mathrm{dnf}_2$, i.e. dnf_2 eventually holds. Let us consider k as the first state (the lowest k) in which dnf_2 holds. Hence, $\forall j < k$ we have $\pi^j \models_{LTL} \neg\mathrm{dnf}_2$. In order to show the desired property we just need to establish that $\pi^k \models_{LTL} \mathrm{dnf}_1$ (as well as dnf_2). This condition trivially holds since the assumption $\pi \models_{LTL} \Box\neg(\mathrm{dnf}_2 \wedge \neg\mathrm{dnf}_1)$ is logically equivalent to $\pi \models_{LTL} \Box(\mathrm{dnf}_2 \rightarrow \mathrm{dnf}_1)$. So, since k is such that $\pi^k \models_{LTL} \mathrm{dnf}_2$, so does dnf_1 and this ends the proof.

***Case Negative Before* (⇒)** Given $\pi \models_{LTL} \Box\ (\Diamond\ \mathrm{dnf}_2\ \ \rightarrow\ \ \neg\ \mathrm{dnf}_1)$ we have to prove that it implies $\pi \models_{LTL} \Box\neg(\mathrm{dnf}_1 \wedge \mathrm{dnf}_2)$.

The proof is given by contradiction. Thus, by negating the thesis we have that $\pi \models_{LTL} \Diamond(\mathrm{dnf}_1 \wedge \mathrm{dnf}_2)$ and, by the premise, we have $\pi \models_{LTL} \Box\ (\neg\mathrm{dnf}_1 \vee \neg\Diamond\ \mathrm{dnf}_2)$.

By applying the definition of $\Diamond$ LTL operator we have that $\exists j \geqslant 1\ \pi^j \models_{LTL} (\mathrm{dnf}_1 \wedge \mathrm{dnf}_2)$ and $\pi^j \models_{LTL} (\neg\mathrm{dnf}_1 \vee \neg\Diamond\ \mathrm{dnf}_2)$. This can be rewritten as $\exists j \geqslant 1\ \pi^j \models_{LTL} (\mathrm{dnf}_1 \wedge \mathrm{dnf}_2 \wedge \neg\mathrm{dnf}_1)$ or $\pi^j \models_{LTL}$

$(\mathit{dnf}_1 \wedge \mathit{dnf}_2 \wedge \neg\Diamond\ \mathit{dnf}_2)$, that in both cases leads to a contradiction. Notice, indeed, that $\pi^j \models_{LTL} \neg\Diamond\ \mathit{dnf}_2$ implies $\pi^j \models_{LTL} \neg\mathit{dnf}_2$.

Case Negative Before (⇐) Given $\pi \models_{LTL} \Box\neg(\mathit{dnf}_1 \wedge \mathit{dnf}_2)$ we have to prove that it implies $\pi \models_{LTL} \Box\ (\Diamond\ \mathit{dnf}_2\ \rightarrow\ \neg\ \mathit{dnf}_1)$.

The proof is given by contrapposition.

Suppose that $\pi \models_{LTL} \Diamond\neg(\neg\Diamond\ \mathit{dnf}_2\ \vee\ \neg\ \mathit{dnf}_1)$. It can be rewritten as $\pi \models_{LTL} \Diamond(\Diamond\ \mathit{dnf}_2\ \wedge\ \mathit{dnf}_1)$. By definition of $\Diamond$ LTL operator we have that $\exists j \geqslant 1\ \pi^j \models_{LTL} (\Diamond\ \mathit{dnf}_2\ \wedge\ \mathit{dnf}_1)$, that is $\exists k \geqslant j\ \pi^k \models_{LTL} \mathit{dnf}_2$ and, by Lemma 6.2, $\pi^k \models_{LTL} \mathit{dnf}_1$. Thus we have $\exists k \geqslant j\ \pi^k \models_{LTL} \mathit{dnf}_1 \wedge \mathit{dnf}_2$ that equals to $\pi \models_{LTL} \Diamond(\mathit{dnf}_1 \wedge \mathit{dnf}_2)$, that is the negation of our premise.

Case Response (⇒) Given $\pi \models_{LTL} \Box(\mathit{dnf}_1 \rightarrow \Diamond\mathit{dnf}_2)$ we have to prove that it implies $\pi \models_{LTL} \Box(\neg\mathit{dnf}_1 \vee \Diamond(\mathit{dnf}_1 \wedge \mathit{dnf}_2))$

The proof is given by contrapositive.

Suppose $\pi \models_{LTL} \neg\Box(\neg\mathit{dnf}_1 \vee \Diamond(\mathit{dnf}_1 \wedge \mathit{dnf}_2))$, that equals to $\pi \models_{LTL} \Diamond(\mathit{dnf}_1 \wedge \Box(\neg\mathit{dnf}_1 \vee \neg\mathit{dnf}_2))$. By definition, $\exists j \geqslant 1\ \pi^j \models_{LTL} \mathit{dnf}_1$ and $\forall k \geqslant j\ \pi^j \models_{LTL} \neg\mathit{dnf}_1 \vee \neg\mathit{dnf}_2$. By Lemma 6.2 and by the definition of $\Diamond$ LTL operator, it must be the case that $\exists j \geqslant 1\ \pi^j \models_{LTL} \mathit{dnf}_1$. Therefore, since dnf_1 remains true after j $\forall k \geqslant j\ \pi^j \models_{LTL} \neg\mathit{dnf}_2$, that is $\exists j \geqslant 1\ \pi^j \models_{LTL} (\mathit{dnf}_1 \wedge \Box\neg\mathit{dnf}_2)$. Therefore $\pi \models_{LTL} \neg\Box(\mathit{dnf}_1 \rightarrow \Diamond\mathit{dnf}_2)$.

Case Response (⇐) Given $\pi \models_{LTL} \Box(\neg\mathit{dnf}_1 \vee \Diamond(\mathit{dnf}_1 \wedge \mathit{dnf}_2))$ we have to prove that it implies $\pi \models_{LTL} \Box(\mathit{dnf}_1 \rightarrow \Diamond\mathit{dnf}_2)$

The proof is given by contradiction.

For the sake of contradiction we assume that $\pi \models_{LTL} \neg\Box(\mathit{dnf}_1 \rightarrow \Diamond\mathit{dnf}_2)$, that equals to $\pi \models_{LTL} \Diamond(\mathit{dnf}_1 \wedge \neg\Diamond\mathit{dnf}_2)$. Thus, by Lemma 6.2 and by definition of *always* ($\Box$) LTL operator, $\exists j \geqslant 1\ \pi^j \models_{LTL} \mathit{dnf}_1$ and $\forall k \geqslant j\ \pi^k \models_{LTL} \mathit{dnf}_1 \wedge \neg\mathit{dnf}_2$. Thus $\exists j \geqslant 1\ \pi^j \models_{LTL} \mathit{dnf}_1$ and $\pi^j \models_{LTL} \Box(\mathit{dnf}_1 \wedge \neg\mathit{dnf}_2)$. By hypothesis, however, we have that $\forall j \geqslant 1\ (\pi^j \models_{LTL} \neg\mathit{dnf}_1$ or $\pi^j \models_{LTL} \Diamond(\mathit{dnf}_1 \wedge \mathit{dnf}_2))$. This leads to a contradiction.

Case Negative Response (⇒) Given $\pi \models_{LTL} \Box(\mathit{dnf}_1 \rightarrow \neg\Diamond\mathit{dnf}_2)$ we have to prove that it implies $\pi \models_{LTL} \Box\neg(\mathit{dnf}_1 \wedge \mathit{dnf}_2)$.

The hypothesis can be rewritten as $\pi \models_{LTL} \Box(\neg\mathit{dnf}_1 \vee \neg\Diamond\mathit{dnf}_2)$. Since $\pi \models_{LTL} \neg\Diamond\mathit{dnf}_2$ implies $\pi \models_{LTL} \neg\mathit{dnf}_2$ we have that $\pi \models_{LTL} \Box(\neg\mathit{dnf}_1 \vee \neg\Diamond\mathit{dnf}_2)$ implies $\pi \models_{LTL} \Box(\neg\mathit{dnf}_1 \vee \neg\mathit{dnf}_2)$, that can be rewritten as $\pi \models_{LTL} \Box\neg(\mathit{dnf}_1 \wedge \mathit{dnf}_2)$.

Case Negative Response (⇐) Given $\pi \models_{LTL} \Box\neg(\mathit{dnf}_1 \wedge \mathit{dnf}_2)$ we have to prove that it implies $\pi \models_{LTL} \Box(\mathit{dnf}_1 \rightarrow \neg\Diamond\mathit{dnf}_2)$.

The proof is given by contrapositive.

Suppose that $\exists j \geqslant 1\ \pi^j \models_{LTL} \neg(\neg\mathit{dnf}_1 \vee \neg\Diamond\mathit{dnf}_2)$. This can be rewritten as $\exists j \geqslant 1\ \pi^j \models_{LTL} (\mathit{dnf}_1 \wedge \Diamond\mathit{dnf}_2)$. By definition of $\Diamond$ operator this means $\exists j \geqslant 1\ \pi^j \models_{LTL} \mathit{dnf}_1$ and $\exists k \geqslant j\ \pi^k \models_{LTL} \mathit{dnf}_2$.

By Lemma 6.2, $\pi^k \models_{LTL} dnf_1$. Thus we have that $\exists k \geqslant 1\ \pi^k \models_{LTL} (dnf_1 \wedge dnf_2)$. This proves the contradiction of our premise.

Case Correlate **(⇒)** Given $\pi \models_{LTL} \Diamond dnf_1 \rightarrow \Diamond dnf_2$ we have to prove that it implies $\pi \models_{LTL} \Box(\neg dnf_1 \vee \Diamond(dnf_1 \wedge dnf_2))$.

The proof is given by contrapositive.

Suppose that $\pi \models_{LTL} \Diamond(dnf_1 \wedge \Box(\neg dnf_1 \vee \neg dnf_2))$. By definition of *eventually* ($\Diamond$) and *always* ($\Box$) LTL operators it entails that $\exists j \geqslant 1\ \pi^j \models_{LTL} dnf_1 \wedge \forall k \geqslant j\ \pi^k \models_{LTL} \neg dnf_1 \vee \neg dnf_2$. By Lemma 6.2 and since $\pi^j \models_{LTL} dnf_1$ condition above is $\exists j \geqslant 1\ \pi^j \models_{LTL} dnf_1 \wedge \forall k \geqslant j\ \pi^k \models_{LTL} \neg dnf_2$. By Lemma 6.2 we have that $\forall k \geqslant j\ \pi^k \models_{LTL} \neg dnf_2$ implies $\forall j \geqslant 1\ \pi^j \models_{LTL} \neg dnf_2$. This implies that $\pi \models_{LTL} \Diamond dnf_1 \wedge \neg \Diamond dnf_2$. Therefore $\pi \models_{LTL} \neg(\Diamond dnf_1 \rightarrow \Diamond dnf_2)$.

Case Correlate **(⇐)** Given $\pi \models_{LTL} \Box(\neg dnf_1 \vee \Diamond(dnf_1 \wedge dnf_2))$ we have to prove that it implies $\pi \models_{LTL} \Diamond dnf_1 \rightarrow \Diamond dnf_2$.

The proof is given by contradiction.

Let us assume $\pi \models_{LTL} \neg(\Diamond dnf_1 \rightarrow \Diamond dnf_2)$. This equals to $\pi \models_{LTL} \Diamond dnf_1 \wedge \neg \Diamond dnf_2$. By definition of $\Diamond$ operator and by Lemma 6.2, $\exists j \geqslant 1\ \pi^j \models_{LTL} dnf_1 \wedge \neg \Diamond dnf_2$. By Hypothesis, $\forall i \geqslant 1, \pi^i \models_{LTL} \neg dnf_1 \vee \Diamond(dnf_1 \wedge dnf_2)$. Let $i = j$, $\pi^j \models_{LTL} dnf_1 \wedge \neg \Diamond dnf_2$ and $\pi^j \models_{LTL} \neg dnf_1 \vee \Diamond(dnf_1 \wedge dnf_2)$. This means that $\pi^j \models_{LTL} (dnf_1 \wedge \neg \Diamond dnf_2 \wedge \neg dnf_1)$ or $\pi^j \models_{LTL} (dnf_1 \wedge \neg \Diamond dnf_2 \wedge \Diamond(dnf_1 \wedge dnf_2)$. This leads to a contradiction.

Case Negative Correlate **(⇒)** Given $\pi \models_{LTL} \Diamond dnf_1 \rightarrow \neg \Diamond dnf_2$ we have to prove that it implies $\pi \models_{LTL} \Box \neg(dnf_1 \wedge dnf_2)$.

The proof is given by contrapposition.

Suppose $\pi \models_{LTL} \Diamond(dnf_1 \wedge dnf_2)$. By definition, this implies $\pi \models_{LTL} \Diamond dnf_1 \wedge \Diamond dnf_2$. That is $\pi \models_{LTL} \neg(\Diamond dnf_1 \rightarrow \neg \Diamond dnf_2)$.

Case Negative Correlate **(⇐)** Given $\pi \models_{LTL} \Box \neg(dnf_1 \wedge dnf_2)$ we have to prove that it implies $\pi \models_{LTL} \Diamond dnf_1 \rightarrow \neg \Diamond dnf_2$.

The proof is given by contrapposition.

Suppose $\pi \models_{LTL} \neg(\Diamond dnf_1 \rightarrow \neg \Diamond dnf_2)$. Then $\pi \models_{LTL} \Diamond dnf_1 \wedge \Diamond dnf_2$. Thus $\exists j \geqslant 1\ \pi^j \models_{LTL} dnf_1$ and $\exists k \geqslant 1\ \pi^k \models_{LTL} dnf_2$. By Lemma 6.2, $\forall j' \geqslant j\ \pi^{j'} \models_{LTL} dnf_1$ and $\forall k' \geqslant k\ \pi^{k'} \models_{LTL} dnf_2$. Let us consider z as the larger between k and j. $\pi^z \models_{LTL} dnf_1 \wedge dnf_2$. Thus $\pi \models_{LTL} \neg \Box \neg(dnf_1 \wedge dnf_2)$.

$\Box$

Obtaining an infinite path

Let us now define the infinite path corresponding to a finite path. Intuitively, it is obtained by adding a self loop to the last state of the finite path. The introduction of the infinite path is functional to prove the soundness of our implementation. Indeed, we have to prove that, given a protocol, a legal path for the corresponding labelled graph is a path of a 2CL-GCM of the protocol (recall that 2CL-GCM is defined on infinite paths).

Definition 6.7. *Let $\equiv$ be the logical equivalence according to propositional logic. Let $\pi = \langle(\pi_0, a_0, \pi_1), \dots, (\pi_{n-1}, a_{n-1}, \pi_n)\rangle$ be a finite path of at least one state. Let '$\mathtt{act}$ **means** $\mathtt{true}$ **if** π_n' be a protocol action defined according to Definition 3.2. π^∞ is the corresponding infinite path where: (i) $\forall i,\ 0 \leqslant i \leqslant n\ \pi_i \equiv \pi_i^\infty$; and (ii) $\forall i,\ 0 \leqslant i < n\ (\pi_i, a_i, \pi_{i+1}) \in \pi$ iff $(\pi_i, a_i, \pi_{i+1}) \in \pi^\infty$ and (iii) π_i^∞ contains the loop $(\pi_n, \mathtt{act}, \pi_n)$.*

We now have all the elements for defining and proving the soundness theorem[1].

Theorem 6.3 (Soundness)**.** *Let $\mathcal{P} = \langle \mathsf{Ro}, \mathsf{F}, s_0, \mathsf{A}, \mathsf{Cst}\rangle$ be a protocol s.t. Cst is a set of state constraints and actions in A do not retract predicates. Let $\pi = \langle(\pi_0, a_0, \pi_1), \dots, (\pi_{n-1}, a_{n-1}, \pi_n)\rangle$ be a path and π^∞ be the corresponding infinite path according to Definition 6.7. We denote with π_i^∞ the states belonging to π^∞. Let $\mathsf{G}(\mathcal{P}) = (\mathsf{S}, \delta, \mathrm{I})$ be the labelled graph corresponding to $\mathcal{P}$. There exists a 2CL-GCM **P** of protocol $\mathcal{P}$ s.t. if π is a legal path of $\mathsf{G}(\mathcal{P})$ then π^∞ is a path of **P**.*

***Proof* 6.3.1.** *Let us consider the 2CL-GCM **P** $= \langle \mathsf{S}_\pi, \mathsf{L}_\mathsf{A}, s_0, \Delta, \mathsf{G}, \mathsf{Cst}\rangle$ where:*

- *S_π is the set of states in π;*
- *L_A is the set of actions labels in $\mathsf{A} \cup \{act\}$;*
- *G is the subset of S_π s.t. each state does not contain unsatisfied commitments;*
- *Δ is the action theory corresponding to $\mathsf{A} \cup \{act\}$.*

*In order for π^∞ to be a path of the 2CL-GCM **P** it must satisfy the conditions* (i) –(iii) *of Definition 4.13:*

i. *$\forall(\pi_i^\infty, a_i, \pi_{i+1}^\infty)$ in π^∞ then (i.1) $\pi_i^\infty, \pi_{i+1}^\infty \in \mathsf{S}_\pi$, (i.2) $a_i \in \mathsf{L}_\mathsf{A}$, and (i.3) $\pi_i^\infty \overset{a_i}{\hookrightarrow} \pi_{i+1}^\infty \in \Delta$. Condition (i.1) holds by construction of **P**. Condition (i.2) holds trivially by definition of $\mathcal{P}$ (see Definition 4.11). Condition (i.3). Let us assume, by absurd, that $\pi_i^\infty \overset{a_i}{\hookrightarrow} \pi_{i+1}^\infty \notin \Delta$. This is possible when one of the conditions in Definition 4.10 is not satisfied. For construction of π^∞ then $\exists(\pi_i, a_i, \pi_{i+1}) \in \pi$ and consequently $\pi_i^\infty \vdash \mathtt{Cond}$ of a_i. Condition* (a) *holds because Condition* (1) *of Definition 6.3 satisfies the axioms on commitments reported in Table 4.3 (see [Sin07]). Condition* (b) *holds because it corresponds to Condition* (2) *of Definition 6.3 Therefore, $\pi_i^\infty \overset{a_i}{\hookrightarrow} \pi_{i+1}^\infty \in \Delta$.*

1 An alternative formulation of the Theorem given directly on the Prolog program implementing the commitment machine and the corresponding proof can be found in [BBC+12]

ii. *Being* $\mathrm{inf}(\pi^{\infty})$ *the set of states that occur infinitely often in* π^{∞}*, by construction of* π^{∞} *we have that* $\pi_n \in \mathrm{inf}(\pi^{\infty})$*. Being* π *legal for* $G(\mathcal{P})$ *then* FINAL $\in I(\pi_n)$ *(by Definition 6.4). This entails that* π_n *does not contain unsatisfied commitments (by Definition 6.3). Therefore,* $\pi_n \in G$ *and thus* $\pi_n \in \mathrm{inf}(\pi^{\infty}) \cap G$.

iii. *Let us consider the transition system* $T(\pi^{\infty})$ *obtained according to Definition 4.12. Since actions in* A *do not retract predicates and* π *is a path of* $G(\mathcal{P})$*,* $T(\pi^{\infty})$ *is a factual model according to Definition 6.6. Therefore, Property 6.1 holds.*

By Definition 6.4, $\nexists i \geqslant 0$ *such that* $\pi_i \in \pi$ *and* VIOLATION $\in I(\pi_i)$*. Therefore, by Definition 6.3* $\forall i \geqslant 0, \nexists c \in \mathrm{Cst}$ *s.t.* c *is not a response or a correlation and* $\pi_i \nvDash sf(c)$*. This equals to say that* $\pi \models_{\mathrm{LTL}} \varphi_{sf}(c)$ *(according to Table 6.2). Since* c *is a state constraint for hypothesis, by Property 6.1 we have that* $\pi^{\infty} \models_{\mathrm{LTL}} \varphi_{ltl}(c)$.

For what concerns response and correlation constraints we have that, by Definition 6.4, PENDING $\notin I(\pi_n)$*. This entails that* $\nexists c \in \mathrm{Cst}$ *s.t.* $\pi_n \nvDash sf(c)$ *and* c *is a response or a correlation constraint. This entails that* $\forall c \in \mathrm{Cst}$*,* c *of kind response or correlation,* $\pi \models_{\mathrm{LTL}} \varphi_{sf}(c)$*. Since* c *is a state constraint for hypothesis, by Property 6.1 we have that* $\pi^{\infty} \models_{\mathrm{LTL}} \varphi_{ltl}(c)$.

□

Concerning the soundness of the implementation of state transitions w.r.t. the inference of state transitions according to the 2CL-GCM, in the proof above we rely on [Sin07]. According to it, indeed, the implementation in [WLH05] is correct according to the formulation of the axioms on commitment manipulations. We inherit this property since we do not modify the mechanism for determining the states resulting form the execution of an action (and consequently the mechanism for commitments manipulation).

Theorem 6.3 allows us to infer that every path that is legal according to the labelled graph obtained by our implementation, is a path that can be inferred from a 2CL-GCM corresponding to the protocol. Our implementation is not complete with respect to the semantics because of the assumptions it is based on. The easiest way to see it is by noticing that the implementation is not able to handle *premise* constraints (for the reasons we have already explained). Even not being complete, we still be able to handle many examples, as shown in Appendix B, and by the analysis of the real case studies presented in Chapter 7.

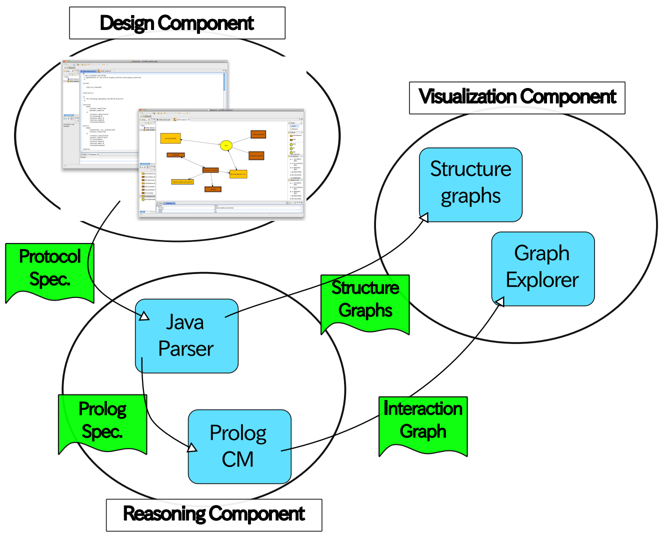


Figure 6.1: Components and functionalities supplied by the system.

6.3 2CL TOOLS FOR PROTOCOL DESIGN AND VISUALIZATION

The technical framework described in the previous sections is able to provide a labelled graph of the possible interactions and of the possible regulative violations. In order to better support the user we have developed a tool, based on this technical framework, that supports him/her in two different ways: (i) it features two graphical editors for specifying the protocol actions and the constraints; (ii) it generates different kinds of graphs for supporting the analysis of the possible interactions and the understanding of which of them are legal [BBM+11b, BBMP11, BBPM12]. The system is realized as an Eclipse plug-in, available at the URL `http://di.unito.it/2cl`.

The functionalities that the system supports can be grouped into three components: *design*, *reasoning* and *visualization* (see Figure 6.1).

Design Component.

The design component provides the tools that are necessary for defining the protocol. It supplies two editors [Cap10]: one for the definition of the actions and one for the definition of constraints (Figure 6.2).

Action definition

The editor for *action definition* is basically a text editor. The content of the initial state and the specification of the actions are introduced by means of two labels: *initial* and *constitutive*. The former is followed by a list of facts and commitments holding in the initial state. They are separated by commas and the list ends with a full stop. The possibility of specifying an initial state allows to represent those situations in which some commitments or facts already hold at the beginning of the interaction. The Prolog program will start to compute the possible evolutions of the social state by evaluating which actions are applicable in the state. When the label *initial* is omitted in the specification, it is assumed to be empty.

Example 6.1. *Listing 6.1 reports an example of initial state for the NetBill protocol. It allows to determine the possible interactions that start with a commitment of the merchant* (m) *to the customer* (c) *to send some goods if it pays for them.*

```
initial:
  cc(m,c,pay,good).
```

Listing 6.1: Specification of an example of *initial* state for the NetBill protocol.

The label *constitutive* introduces the definition of the actions. These are given in terms of a precondition and a set of effects (see the example in Listing 6.2). Differently from the specification of the initial state, the constitutive component is mandatory. A protocol action is defined by giving the action name, the list of effects introduced by the keyword **means** and separated by commas, and, if any, a precondition introduced by the keyword **if**. In the actions' precondition it is possible to use the operator '!' that is evaluated as *negation by failure*.

Example 6.2. *Listing 6.2 reports the action* `send_accept` *of NetBill protocol, by which the customer* (c) *accepts a quote from the merchant* m*.*

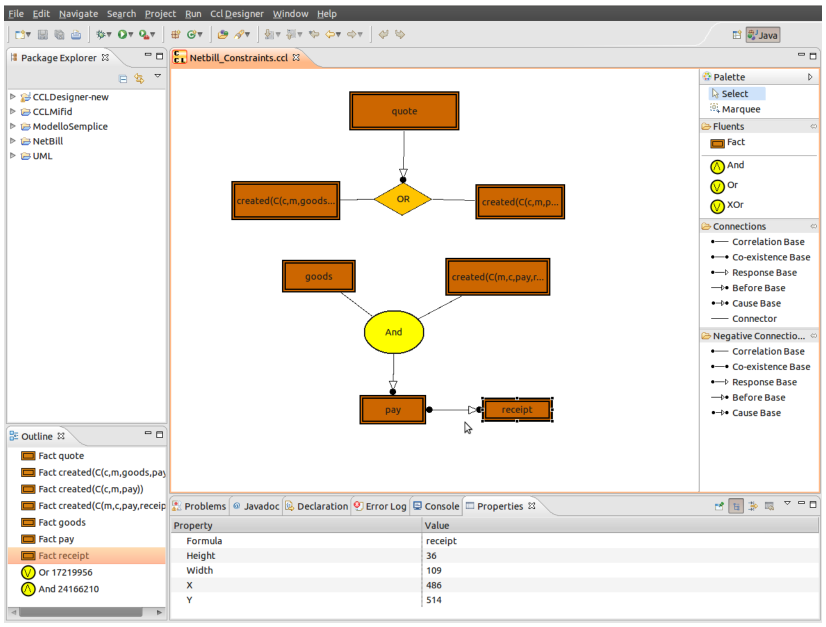
```
send_accept
  means
    create(C(c,m,goods,pay)),
  if
    !pay.
```

Listing 6.2: Definition of protocol actions

The effect of its execution is the creation of the commitment from the customer to the merchant to pay in case goods are sent. This effect is achieved only if the customer has not already paid (i.e. the fact pay *cannot be derived in the state where the action is applied).*

Constraints definition

The *regulative specification editor* allows the user to graphically define



Property	Value
Formula	receipt
Height	36
Width	109
X	486
Y	514

Figure 6.2: Editor of the Eclipse plug-in for constraints specification.

a set of constraints. Constraints are represented by drawing facts, connecting them with 2CL arrows (following the graphical representation of 2CL operators reported in Table 3.4), or with logical connectives so as to design DNF formulas. The advantage of having a graphical editor is that it supplies a global view of constraints, thus giving the perception of the *flow* imposed by them, without actually specifying any rigid sequence: no-flow-in-flow principle (see Section 3.3).

Figure 6.2 shows a snapshot of the constraint editor with a representation of the NetBill constraints. On the right the user can select the element to introduce in the graph. By editing the properties (bottom of the figure), instead, he/she can specify the name of facts and other graphical aspects.

Reasoning Component.

The actions and the constraints specifications, graphically provided by the user by means of the *Design* tools, are exported into a text file. This latter is then parsed thanks to a Java Parser[2] that is part of the *reasoning component* (Figure 6.1). The parser is able to provide different kinds of graphs statically generated from the protocol specification. For instance, Figure 6.3 graphically reports the impacts of the constraints on the protocol actions. It is generated by considering on the one hand the facts that are involved in the protocol constraints, on the

2 The Java Parser has been realised by Matteo Baldoni.

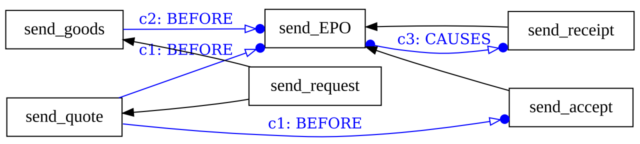


Figure 6.3: Graph representing the impacts of the protocol constraints on the protocol actions.

other hand the facts that are effects or precondition of the actions. By joining these, the parser is able to infer how constraints affect the protocol actions. In the graph that it generates, constraints are represented in blue following 2CL graphical notation. The black arrows, instead, represent connections given by preconditions-effects between the actions. Other variants of this graph report only the regulative (or the constitutive) dependencies. Finally, for all these graphs it is possible to produce two versions: one explicitly showing the facts involved and one without showing them (as in Figure 6.3). These kinds of graphs are called *Structure Graphs*.

The parser is also responsible for the translation of the protocol specification into a Prolog program. Given this program, our implementation computes the labelled graph of all the possible interactions and associates with each state a set of labels concerning its compliance with protocol constraints. The output that the program generates is coded into a Dot[3] file, where each state in the graph represents a possible configuration of the social state. Arrows correspond to actions and are directed. The source is a state where the "if" condition in the definition of the action labelling the arc holds. The target is the state obtained by applying the meaning of the executed action to the source state. The graphical convention for state representation (reported in Table 6.3) is:

(i) a state of violation is represented as a red diamond, with an incoming red arrow;

(ii) a state in which there is a pending condition is yellow-coloured;

(iii) a state with a single outline, independently from the shape, is a state that contains unsatisfied commitments; conversely, a state with a double outline, independently from the shape, does not contain active commitments.

Graphical notations can be combined, e.g. a yellow diamond with single outline is a state where there are unsatisfied active commitments, where a constraint is violated and where there is a pending condition. A legal path connects the initial state with one of the final states (white

3 DOT is a graph description language. It represents the format for input files of *graphviz*, a package specifically thought for visualizing and managing graphs.

State representation	*Unsatisfied commitments*	*Violated constraints*	*Pending constraints*
○	Yes	No	No
🟠	Yes	No	Yes
◇	Yes	Yes	No
🔶	Yes	Yes	Yes
◈	No	Yes	Yes
⟐	No	Yes	No
◉	No	No	Yes
◎	No	No	No

Table 6.3: State representation and corresponding meaning in the social state's reachability graph.

state with double outline) and is made by all black arrows (black arrows denote legal moves).

Visualization Component.

All the graphs produced by the reasoning component can be visualized as images. Additionally, the *labelled graph* can be explored by means of the tool *Graph Explorer* [Pir11], which is realized in Java and relies on iDot (Incremental Dot Viewer), an open source project that uses the prefuse[4] visualization framework for Dot graph display. The Graph Explorer supplies different functionalities, like the visualization of the shortest path given a source and a target state, and the visualisation of legal (or illegal) paths only. The user can add or delete a node in a path; search a state starting from its labels; search all the states that contain a certain fact or commitment. Moreover, the tool allows the exploration of the graph one state at a time, by choosing which node to expand, as shown in Figure 6.4. This aspect is particularly useful in dealing with graphs with a high number of states.

4 http://prefuse.org/

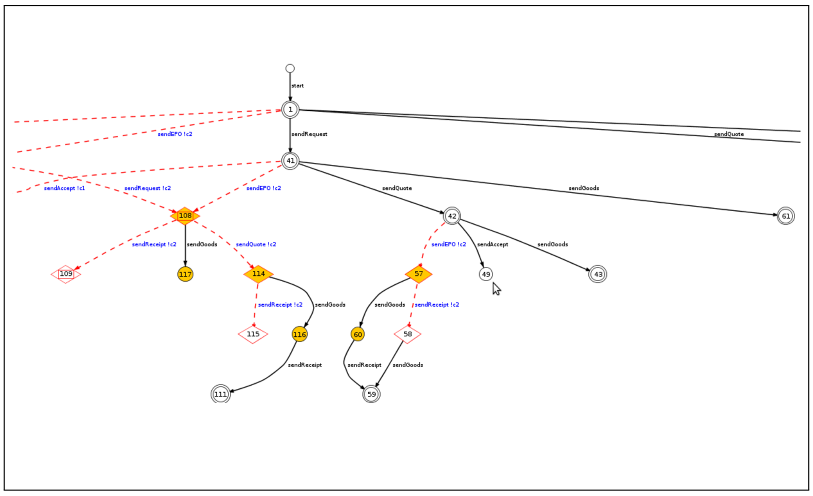


Figure 6.4: Exploration of the labelled graph by means of the *Graph Explorer*. The user can choose which state to expand.

6.4 PROTOCOL ANALYSIS

The analysis of the protocol is an important task both for an interacting party and for a protocol designer [BBMP11, BBPM12]. The tool that we have described can be used as a support in this task [BBMP11]. Declarative approaches, indeed, allow to specify easily and in a flexible way a set of requirements on the interaction but in many cases they are not very intuitive. In particular, from a declarative specification it is difficult to have an intuition of which behaviours are allowed and which are not. This aspect, however, is important both for an interacting party and for a designer to reason about possible risks of violations. For what concerns the designer, it is not always easy, when specifying a protocol, to individuate which constraints to introduce but, with the help of the tool, it becomes easy to identify misbehaviours and revise the constraints so as to avoid them. Moreover, a designer can decide, by analysing the graph, to modify the specification so as to regiment [JS94], if possible, some of the patterns expressed as constraints. This can be done in order to avoid violations that may occur frequently or that may cause losses or damages of some kinds. From an interacting party point of view it necessary to understand how regulations impact on the internal functioning, to reason about possible *risks of violation*. This need is even more evident in complex scenarios (like banking, trading services, or personal data flow management), characterized by a high degree of regulations which often change along time (see Chapter 7 for the description of two real case studies presenting these characteristics).

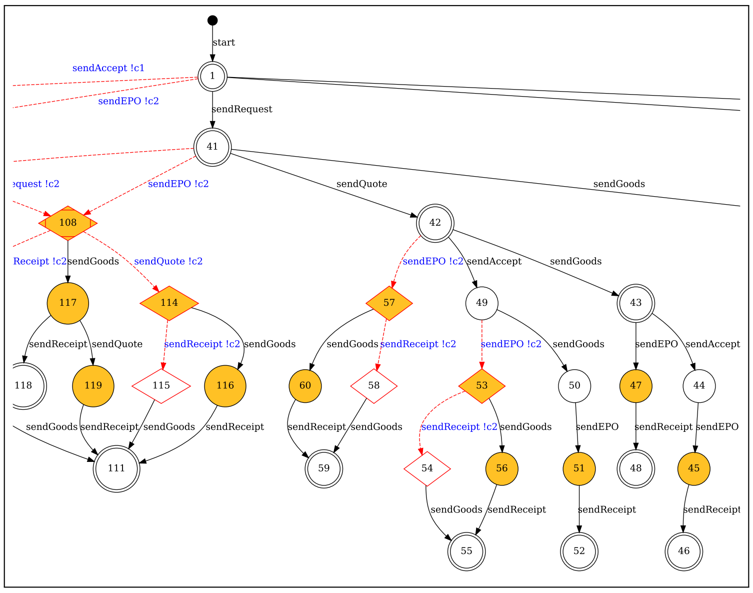


Figure 6.5: Partial view of the reachability graph of NetBill Protocol.

Example 6.1. *For instance, consider the labelled graph of NetBill protocol reported in Figure 6.5. From the graph it is possible to infer that the protocol does not allow the customer to pay* ($\mathsf{sendEPO}$) *before the merchant sends the goods. This is due to the constraint*

$$\textsc{created}(\mathsf{C}(m, c, pay, receipt)) \wedge goods \longrightarrow\!\!\triangleright\!\bullet\ pay$$

If this behaviour was not in the intention of the designer, he/she can discover it and, e.g., relax the before *constraint* ($\longrightarrow\!\!\triangleright\!\bullet$) *transforming it into a* co-existence ($\bullet\!\!-\!\!\bullet$)*. If, instead, that is exactly the desired behaviour, one may decide (if possible) to regiment* $\mathsf{sendEPO}$ *so as to enable the payment only after the goods have been sent.*

Particularly interesting for this kind of analysis is the possibility of exploring the labelled graph by means of the Graph Explorer, which can be used to predict whether performing a certain sequence of actions results in a violation and, in this case, if there is a way to return on a legal path.

Notice that these kinds of analysis are different from [MTC+10]. In this work, the authors propose a way for verifying static properties on the specification, like existential or universal properties based on abductive logic programming and expressed in terms of expectations. The system is able to answer with a *yes* or *no* (depending on the satisfaction of the property) and, when a property is not satisfied, it provides a counterexample. Other works, like [DGG+10, Gov10], focus on the problem of verifying the compliance of a business process to a body of norms. This issue is different in that the business process is rigidly modelled as a (YAWL or BPM) workflow, and the verification aims at checking if this process strictly respects the norms, providing, in some cases, a yes or no answer and, in some others, a degree of compliance. These kinds of verifications are surely interesting, but they are not meant to provide a global overview of the possible interactions. Therefore, it could be difficult to analyse the possible behaviours and to recognize those leading to a violation. Similar considerations hold for the work in [CMMT09a]. Here the authors propose a mechanism for commitment tracking. This kind of verification is known as *a-posteriori* trace compliance and it is able to verify (or to simulate the execution of) one trace at a time.

Another interesting work, which however does not tackle protocols and regulations, is [Ser08] by Sergot. This work discusses the relationship between the norms that govern a single agent with those that express a designer's view on what overall system behaviours are deemed to be legal. This proposal foresees two models that are to be compared, and to this aim a coloured labelled transition system is used. The main difference w.r.t. our proposal is that in Sergot's approach the focus is on verifying the behaviour of a single agent against a global model. This is a complementary task w.r.t. the one we face. It would be interesting to study how to combine the two approaches in order to supply a complete tool-kit to the business analyst.

7 CASE STUDIES

CONTENTS

In this chapter we consider two real case studies: the OECD Guidelines for private data protection and the MiFID, ruling the offer of financial investment services off-site. By means of them we will show how the 2CL approach can be used for addressing practical scenarios. We will discuss their 2CL specification and we show how the tool we have presented can be used for the analysis of risks of violation.

7.1 MODULARITY AND GRAFTING OF NEW REGULATIONS

When considering practical settings, the reality in which protocols operate is characterized by a high degree of regulation. This is, for instance, the case of banking and of trading services, and of personal data flow management. The interacting parties need to actively determine their processes, to understand how regulations impact on their internal functioning, to reason about possible *risks of violation*, and to ensure *compliance* to directives and regulations.

In these settings, protocols must accommodate different needs. Interactions are usually *cross-business*: the minimization of the effort needed to define proper interfaces and of the altering of internal implementations calls for abstractions that capture the contractual relationships among the interacting parties [TS10]. Protocols must enable a *flexible enactment*, in order to allow the interacting parties, who are heterogeneous, autonomous, and basically self-interested entities, to find the way of interacting that better suits their characteristics and requirements. Flexibility is important to allow the interacting parties to profit from opportunities or to make the most efficient use of their time that is possible. Moreover, protocol specifications must be *modular* in a way that simplifies keeping them compliant to regulations (which often change along time). Existing approaches to protocol specification (e.g. BPEL, WS-CDL) rely on the specification of the flow of the interaction. This procedural view makes protocols not suitable to easily take in new regulations. The first reason is because the composition techniques, that can be applied, tend to impose unnecessary orderings of the interactions w.r.t. what is foreseen by the regulation. Additionally, new regulations often impose the execution of new activities that are to be interleaved with the previously existing ones. In other words, these standards by and large require to rewrite the protocols from scratch.

2CL specifications answer to all the above requirements. Specifically, they are based on commitments, thus allowing to express contractual relationships; they allow for flexible enactments, by expressing requirements in a declarative way and by not expressing strict sequences; they allow for a modular specification thus simplifying reuse and adaptation of the specification. These characteristics make 2CL approach suitable for handling and representing practical settings [BBMP11, BBPM12]. In this chapter we provide evidence of it by analysing two case studies. Specifically, the two examples that we consider are two cases of directives issued by supranational authorities and institutions, like the European Union (EU) or the Organisation for Economic Co-operation and Development (OECD), that must be reconciled with the laws of the single nations. One example is the Markets in Financial Instruments Directive (MiFID for short), directive number 2004/39/EC [Mifb], issued by the European Commission within the Financial Services Action Plan, which represents a fundamental step in the creation of an integrated and harmonized financial market within EU. The MiFID case study is one of the benchmarks of the ICT4LAW project (`http://www.ict4law.org`). Another example is that of the OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data [Org80]. These guidelines regulate the management of personal data by imposing new activities aimed at protecting the data owners.

The common characteristic to MiFID and OECD Guidelines is that they do not represent protocols per se, but they specialize the previously existing sales and data flow protocols: they *graft* onto them, adding new activities which are interleaved with those of the previous protocols. For example, OECD guidelines require that when some data are asked, before sending them, it is necessary to verify their accuracy and that the purpose for which they are requested matches with the purpose for which the data was stored. In the case of MiFID, the bank must verify the order before sending the contract. In both cases, asking data and sending data (or the contract) are activities foreseen by the original protocol, while the others are introduced by the new regulations.

Grafting of regulations

In order to represent this kind of composition of protocols in a modular way, we define a new operation named *grafting* [BBMP11]. It is represented with the symbol $\uplus$ and the grafting of two protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ is defined as follow:

Definition 7.1 (Grafting)**.** *Let* $\mathcal{P}_1 = \langle \mathsf{Ro}_1, \mathsf{F}_1, \mathsf{s}_1, \mathsf{A}_1, \mathsf{Cst}_1 \rangle$ *and* $\mathcal{P}_2 = \langle \mathsf{Ro}_2, \mathsf{F}_2, \mathsf{s}_2, \mathsf{A}_2, \mathsf{Cst}_2 \rangle$ *be two protocols. The grafting* $\mathcal{P}_1 \uplus \mathcal{P}_2$ *is the tuple:*

$$\mathcal{P}_1 \uplus \mathcal{P}_2 = \langle \mathsf{Ro}_1 \cup \mathsf{Ro}_2, \mathsf{F}_1 \cup \mathsf{F}_2, \mathsf{s}_1 \cup \mathsf{s}_2, \mathsf{A}_1 \cup \mathsf{A}_2, \mathsf{Cst}_1 \cup \mathsf{Cst}_2 \rangle$$

We assume that the action names of the two protocols are disjoint; when this is not the case, a renaming can be applied (for instance following the methodology proposed by Desai *et al.* [DCS09]). The idea behind our definition of grafting is to use 2CL constraints as "zippers" that tie temporally stratified specifications, by introducing coordination patterns. This way of sewing two specifications is easier and more flexible than procedural approaches or approaches based on preconditions. In these cases, the introduction of a new requirement would entail the analysis of all the allowed executions, or of the definition of the actions, and their modification.

Another approach based on commitment is that by Telang and Singh [TS10]. In this work the authors propose a commitment-based approach for representing business protocols and identify a set of common patterns of interaction, that can be used by the business analyst. Along this line, also Chopra and Singh [CS11] propose commitment patterns that capture common business patterns, showing which robustness requirements are met by each of them. These requirements are supposed to guide the protocol designer in the selection and composition process. Concerning composition, Desai *et al.* [DCA+07] propose temporal operators to compose the data flow in a commitment-based approach. 2CL approach extends the ones above by enriching the protocol with expressive temporal constraints. This is an added value in the modelling of practical and high regulated scenarios because it enables the embedding of regulations that stratify along time.

Let us now enter into the details of the approach by discussing the two case studies.

7.2 OECD GUIDELINES FOR PERSONAL DATA PROTECTION

The Organisation for Economic Co-operation and Development is the organism responsible for the definition of the "Guidelines on the Protection of Privacy and Transborder Flows of Personal Data" [Org80]. These guidelines represent an agreement between different nations concerning the management of the flow of personal data. Their aim is twofold. On the one hand, they aim at protecting data owners by preventing the violation of their fundamental rights. These violations can be caused, for instance, by the unauthorised use of personal data or the storage of inaccurate data. On the other hand, guidelines sustain the international flow of personal data: different nations, in fact, have different laws on personal data management. OECD Guidelines, by imposing additional and shared regulations, give guarantees on how personal data will be treated, increasing trust between countries and, thus, encouraging the flow of data. These Guidelines, indeed, are meant to be a "minimum standard" which should be provided by nations and potentially enriched with additional internal measures for the protection of privacy and individual liberties. A country is allowed to refuse the forwarding of personal data when the addressee country does not observe the guidelines. OECD Guidelines address many aspects of personal data management, by defining different principles organized into different parts. Among these they define the "*Basic Principles of International Application: Free Flow and Legitimate Restrictions*", aiming at avoiding an unjustified over-protection of data which can limit their flow [Org80, Principle 18] and, at the same time, guaranteeing the flow only toward nations that provide an adequate treatment of the data [Org80, Principle 17]. Another set of guidelines address the "*International Cooperation*". Their aim is to guarantee cooperation between nations. For instance, they state that, when necessary, a country should disclose details of the observance of the principles set forth in these Guidelines [Org80, Principle 20]. In this thesis we consider the "*National Application Principles*". Their aim is to define guidelines that countries should follow in order to guarantee a proper treatment of data.

OECD Guidelines are meant to graft onto the countries' internal regulation for Personal Data Management. Therefore, one of the main aspect to take into account for making the guidelines working in practice, is to evaluate and define *how the new regulations impact on the pre-existing internal regulations* of the different nations. In general, when new regulations should be added to rule a pre-existing behaviour, maybe al-

```
(a)  ask_data means asked_data
(b)  send_data means sent_data
         if asked_data ∧ ¬ refuse_data
(c)  refuse_data means refuse_data ∧
         CANCEL(C(dc, asker, sent_data))
         if asked_data ∧ ¬ sent_data
```

Table 7.1: Constitutive specification for the Data Flow protocol.

ready subject to other regulations, this aspect should be considered very carefully. Without considering this aspect, the introduction of new regulations may be unfeasible, needless or so difficult to realize to make costs higher than benefits. On the other hand, members affected by these guidelines need to determine the impact of the new regulations by, for instance, understanding which of the previously allowed interactions becomes illegal. In the following we will show how the tool we have presented can be profitably used in these kinds of analysis [BBMP11].

In order to analyse how the introduction of new regulation affect the pre-existing process of sending data, we start by modelling the sending data process. Then, we individuate the new activities to be performed according to the guidelines and we represent the *grafting* points of them on the existing process by means of the 2CL language.

7.2.1 Pre-OECD Data Flow Protocol

The basic interaction for Data Flow between two countries can be simply represented by the protocol $DF = \langle Ro_{DF}, F_{DF}, s_{DF}, A_{DF}, Cst_{DF} \rangle$, where Ro_{DF} is the set of involved roles which, in this model, are the data controller (dc) and the asker of the data (asker). The data controller is the person who is responsible for storing and managing the data of someone, the asker can be anyone who requests information about someone (in the OECD Guidelines context it would be most likely a foreign nation). s_{DF} is the content of the initial state. In this case we assume that s_{DF} contains the commitment C(dc, asker, asked_data, sent_data). It captures that the data controller is expected to cooperate and to send some data if someone asks for them. A_{DF} is the set of actions (or activities) involved in the data flow process as described later. The set of facts F_{DF} can be derived by collecting those that are involved in the actions. Thus it is F_{DF} = {asked_data, sent_data, refuse_data}. The set of constraints Cst_{DF} is, in this case, empty.

Constitutive specification

Table 7.1 reports the set of actions, and their definition, for the data flow protocol. Let us comment on their definition.

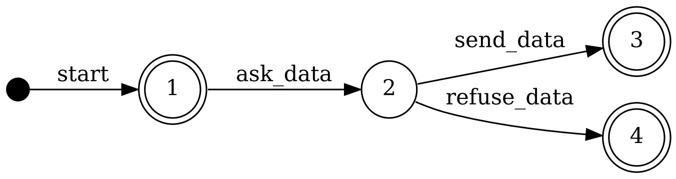


Figure 7.1: Labelled graph of the data flow protocol.

Ask Data *(Table 7.1(a)).* By means of the action *'ask_data'*, the asker can request for some data stored by the data controller. The meaning of this action, therefore, is that data has been requested.

Send Data *(Table 7.1(b)).* By means of the action *'send_data'* the data controller dc sends the data to the asker. This action acquires this meaning only if data have been previously asked and if the dc has not already refused the request. The first condition is meant to capture that the dc can send an information only if it knows which information has been required. The second condition captures that for every request there cannot be both a refusal and a send. Once the dc has taken its decision he/she cannot change its mind.

Refuse Data *(Table 7.1(c)).* The action *'refuse_data'* allows the data controller dc to refuse a request. Therefore, its meaning is to cancel the commitment of sending the data to the asker. As before, it acquires this meaning only if data were not already sent and if the asker asked for them.

Figure 7.1 reports the labelled graph obtained by the described specification. The possible executions are only two: one in which data are sent, and one in which the request of data is refused. Since the regulative specification is empty, all these executions are legal and states 1, 3 and 4 do not contain unsatisfied commitments (the shape of the state is a circle with a double outline). State 2, instead, contains the active commitment $\mathsf{C}(dc, asker, sent\_data)$. Therefore, due to the regulative nature of commitments, in case the interaction ends in this state there would be a violation of the engagement of the data controller towards the asker.

7.2.2 The OECD Guidelines Specification

OECD Guidelines concerning the "*National Application*" foresee the following principles:

- *Data Quality Principle.* Personal data should be relevant, accurate, complete and up-to-date;

```
(d) periodically_verify_accuracy means accuracy_verified
        if ¬ asked_data
(e) check_accuracy means accuracy_verified
        if asked_data
(f) verify_purpose means purpose_verified
        if asked_data
(g) notify_owner means owner_notified
        if sent_data
```

Table 7.2: Constitutive specification of the activities foreseen by the OECD Guidelines.

- *Purpose Specification Principle.* The purposes for which personal data are collected should be clearly specified and the subsequent uses of data are limited to the fulfilment of such purposes (any change of purpose must have the consent of the data subject);
- *Use Limitation Principle.* Personal data should not be disclosed, made available or otherwise used for purposes other than those specified in accordance with the "Purpose Specification Principle", except with the consent of the data subject or by the authority of law.

The application of these principles requires the introduction of new activities that are not part of the data flow protocol. Specifically, the actions for checking the purpose, checking the accuracy of data (periodically or on demand) and for notifying the data subject. The definitions of these actions are reported in Table 7.2.

PERIODICALLY VERIFY ACCURACY *(Table 7.2(d)).* By means of the *'periodically_verify_accuracy'* a data controller dc can check that data are accurate, relevant, complete and up-to-date. This activity is meant to be executed periodically, if data have not been requested.

CHECK ACCURACY *(Table 7.2(e)).* The action *'check_accuracy'* is similar to the periodically check of accuracy but is meant to be executed on demand. For instance, when data are requested by the asker and the accuracy has not been verified yet, then the data controller can check it before sending the data.

VERIFY PURPOSE *(Table 7.2(f)).* The request of the data by the asker is motivated by some purpose. By means of the action *'verify_purpose'* we meant to capture the activity of the data controller of checking that the reason for which data have been asked is compliant with the reason for which data are stored. For instance, if the data of a particular data owner are stored for national security reasons, of course they could not be sent to a third person for advertising reasons. For the sake of simplicity, the specification adopted here abstracts from represent-

(c1)	purpose_verified *before* sent_data
(c2)	accuracy_verified *before* sent_data
(c3)	sent_data *response* owner_notified
(c4)	purpose_verified *before* refuse_data
(c5)	accuracy_verified *before* refuse_data

Table 7.3: OECD Guidelines Regulative Specification.

ing details such as the reason for asking or for storing data. Also the criteria for evaluating the "purpose compliance" are not represented. However, in order for the purpose to be verified the intent of the asker should be known, and this means that data should have been asked by someone already. Therefore, this action can state that purposes are compliant only in case data have been asked. This is the reason of the precondition asked_data.

NOTIFY OWNER *(Table 7.2(g)).* When data are sent to someone, the action *'notify_owner'* acquires the meaning of notifying the data owner that someone has requested for his/her data and that the data controller has sent them.

OECD grafting constraints

The set of activities that the OECD Guidelines add to the Data flow protocol is not sufficient, because it is necessary to specify *how* the new regulation grafts onto the original protocol. For instance, it is necessary to specify when the new activities are to be executed w.r.t. the original Data Flow Protocol. By analysing OECD Guidelines it is possible to find this information and to find the set of 2CL constraints that best represents them. The set of constraints we have identified is reported in Table 7.3 and is graphically represented in Figure 7.2.

Constraint (C1) is of kind *before*. The condition that it captures is that the purpose for which data are requested must be verified (purpose_verified) before data are sent (sent_data) to the asker. Therefore, this constraint answers to the requirement that data can be sent only if the purposes for which they are stored and for which they are required, coincide.

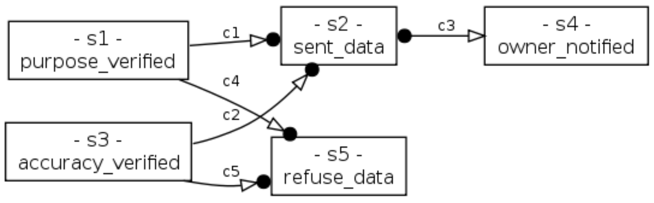


Figure 7.2: Graphical representation of OECD Guidelines constraints.

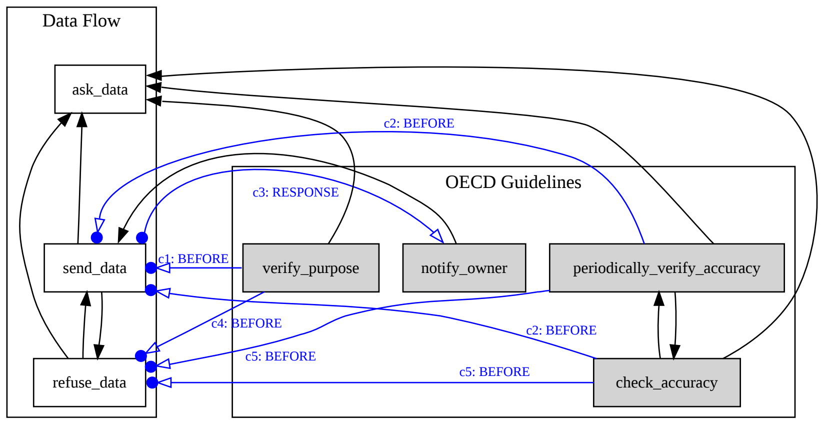


Figure 7.3: Data Flow protocol and OECD Guidelines structure graph.

Constraint (c2) requires that before sending the data the data controllers verifies their accuracy (i.e. that data are complete, accurate and up-to-date). This is in accordance with the *data quality principle*.

Constraint (c3) is of kind response. It captures the requirement that whenever data are sent to someone, then the owner of them should be notified. Notice that this constraint does not require that the notification is performed only after data are sent. But when they are sent then the data owner is required to send the notification at least once, after.

Constraints (c4) and (c5) are similar to constraints (c1) and (c2) respectively. This is because verification should be performed before sending the data, but also in case of the refusal of a request. Therefore, constraint (c4) requires that before refusing a request of some data, the data controller checks the purpose. Constraint (c5) requires that before refusing a request the accuracy of data should have been checked.

Guidelines grafting on Data Flow

We now have all the elements for defining the OECD Guideline specification $G = \langle Ro_G, F_G, s_G, A_G, Cst_G \rangle$. Specifically, $Ro_G = \{dc, owner\}$, A_G is given by the actions listed in Table 7.2, Cst_G is the set of constraints in Table 7.3, the initial state is empty and the set of facts are those involved in the set of actions and constraints of the specification.

Given the Data Flow Protocol DF, previously described, the grafting $DF \uplus G$ is formally defined as follow:

$$DF \uplus G = \langle Ro_{DF} \cup Ro_G, F_{DF} \cup F_G, s_{DF} \cup s_G, A_{DF} \cup A_G, Cst_{DF} \cup Cst_G \rangle$$

Thus, the new protocol includes all the activities of the original data flow protocol and those of the OECD Guidelines. Additionally, the interaction should respect temporal regulations that were not foreseen initially. Figure 7.3 is a structure graph that highlights how 2CL constraints zip the activities of the Data Flow Protocol DF and of the OECD Guidelines G. Blue arrows represents the constraints defined in the regulative part of the guidelines specification Cst_G, while black arrows represents how actions are related to one another in terms of preconditions-effects. For instance, from the figure it is possible to see that the action *'send_data'* that depends on *'ask_data'* in terms of preconditions (black arrow), with the introduction of OECD Guidelines is tied to the actions *'verify_purpose'*, *'check_accuracy'* and *'periodically_verify_accuracy'* by a *before* constraint. These latter, indeed, are activities to be executed before sending the data.

This picture, however, is not sufficient to clearly grasp the impacts of the new regulations on the possible interactions between countries. Specifically, it is important for a country to understand and to clearly visualize which executions are no longer legal. In this way, it can modify the internal process on data management so as to be compliant with the new regulations. The 2CL-GCM implementation we have presented in the previous chapter aims at supplying this kind of analysis. Let us describe how.

7.2.3 Analysis of the Protocol

Figure 7.4 reports the labelled graph for the protocol $\mathsf{DF} \uplus \mathsf{G}$ obtained from our implementation of 2CL-GCM (Section 6.1), thus showing the resulting set of legal and illegal possible executions (the code of the specification is reported in Appendix B). The legal executions are highlighted in green and gray (the highlighting is added by hand). Notice how the actions of the OECD Guidelines (highlighted in green) are immersed in the original data flow protocol (highlighted in gray). The colors also help to highlight the composition of the regulations. All other paths amount to possible violations.

A legal path connects the initial state with one of the final states and is made by all black arrows. For instance, the execution "ask_data, verify_purpose, check_accuracy, send_data, notify_owner" is legal. Instead, "ask_data, check_accuracy, send_data, verify_purpose, notify_owner" contains a violations: constraint (c1) is not respected by *'send_data'*. As Figure 7.4 shows, *'check_accuracy'* and *'verify_purpose'* should be executed after *'ask_data'* but since there is no relation between them, they can be executed in any order. The protocol, however, does not need to specify explicitly each of the interleavings. Moreover, state 9 in the figure, which is reached after sending the data to the asker by executing legal paths, is not final as it would have, instead, been before the introduction of OECD Guidelines: this state does not contain unsatisfied commitments but constraint (c3) is not yet satis-

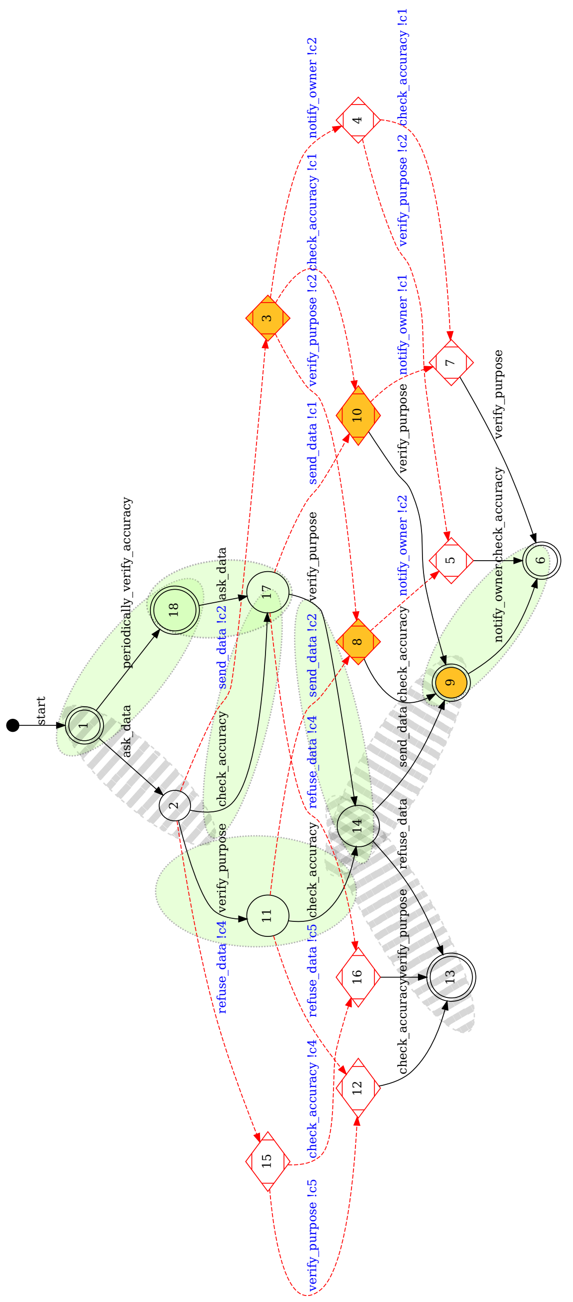


Figure 7.4: Labelled graph for the Data Flow protocol extended with OECD Guidelines.

fied because the data subject has not been notified yet. Notice that we could not achieve the same result by using action preconditions nor by adding new commitments as an additional effect of *'send_data'*. The former solution fails because preconditions do not compel agents to execute applicable actions. The latter solution, instead, modifies an action belonging to the basic data flow protocol making it fit the new protocol. However, in this case, it is difficult to foresee the costs of the modification. For instance: that action might be used in different protocols, normed by different regulations; the implementation of that action might be expensive to update (especially because it would be embedded in a complex software system). One could think to bypass the problem by adding the new commitment as an effect of one of the new actions added by the grafted regulation. This, however, generally makes no sense because the commitment at issue is not naturally part of the constitutive specification of those actions.

Regimenting regulations

By analysing the graph a designer may decide to change the protocol constraints, because he/she has found some misbehaviours in the current specification, or may decide to regiment, where possible, some of the regulations. Let us consider, for instance the action *'refuse_data'*. When is the refusal to send some data allowed? By looking at that graph it is possible to notice that there is only one point (state 14) in which its execution does not lead to a violation. Depending on the gravity of the violation, the designer may consider the possibility to modify the action and, if possible, to regiment the requirement avoiding violations. The resulting definition for the action would be:

```
refuse_data **means** refuse_data,
    CANCEL(C(dc, asker, sent_data))
    **if** asked_data & accuracy_verified &
        !sent_data & !refuse_data.
```

The resulting labelled graph is reported in Figure 7.5. As can be noticed the executions that lead to a violation are decreased, as well as the possible executions. The precondition accuracy_verified, indeed, prevents the action to be executable before any of the actions *'periodically_verify_accuracy'* and *'check_accuracy'*. In the regimented version, *'refuse_data'* raises a violation only if executed before verifying the purposes.

As already mentioned, however, preconditions are not always sufficient to avoid undesired violations (see Section 5.1.1). Considering the action *'refuse_data'* it is possible to make it executable only when the accuracy of the data is verified if this requirement can be guaranteed, for instance, by an intermediary, like an institution, an organization or by the system. Imagine, indeed, that the employee can accept or refuse

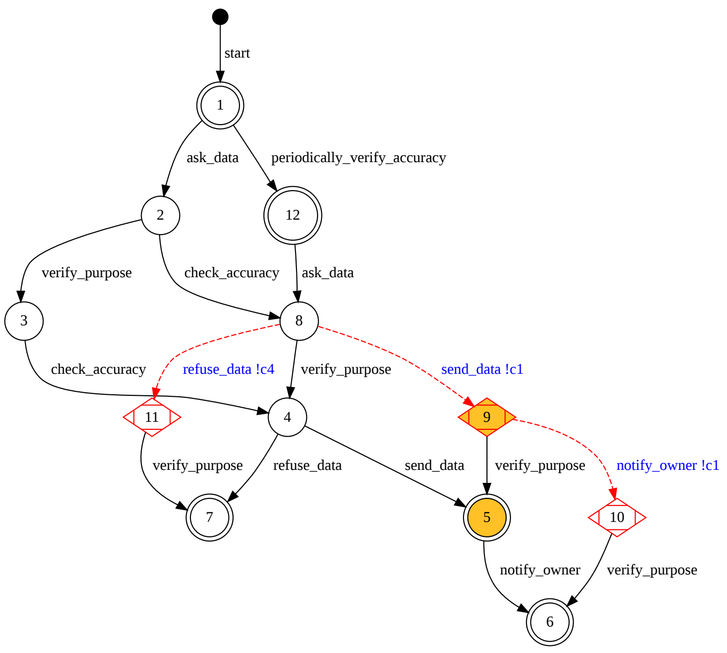


Figure 7.5: Labelled graph of the regimented version of the OECD Guidelines.

the data through a software interface. In this case, in order to modify the action as in the regimented version, the software interface should be changed in order to enable the refuse only after the accuracy verification. If this is not possible the regimentation may not be applicable in practice. Besides the feasibility of the regimentation, the designer may consider also other aspects like, for instance, the costs for realizing it.

7.3 MIFID

The MiFID is part of the *Financial Services Action Plan* (FSAP), emanated in 1999 by the European Commission, in order to face the changes and the evolution of the European financial markets. These changes were caused by the entrance of new subjects on the financial markets and by the growing possibility for the investment firms to propose new products and to operate in foreign countries. Thus, it becomes immediately clear to the European Commission the need for centralized and common rules for the European members that, up to that moment, were ruled only by local laws of the country of origin (in Italy the set of norms for financial markets is defined by an organism named Consob [Con]). The aim was the definition of a new and shared financial market.

The acronym MiFID stands for *Markets in Financial Instruments Directive*. It is the directive 2004/39/CE [Mifa] of the European Parliament and of the Council of the European Union, with the aim of regulating the Community financial market. The MiFID case study [Cap10] is more complex than the OECD Guidelines case study described before. It underlines more strongly the need of a model which accounts explicitly for a regulative specification and of a tool for understanding the impacts of regulations on the pre-existing protocol. Similarly to OECD Guidelines, MiFID grafts onto the previously existing financial product sales protocol. A natural question that may arise is: what happens if an intermediary buys a financial product for a client, violating some of the constraints imposed by MiFID? The *sale is valid*, the client results to be the owner of the product. This happens because MiFID does not define *sales* (sales are defined by a different regulation) but dictates how the interaction with the client should be carried on by adding a new layer of regulations on top of existing ones. So, the violation of some constraint does not affect the sale directly but creates both a *risk of sanction* and a *risk of exposure* for the intermediary. This is witnessed by a sentence by the Italian Supreme Court (*Cassazione civile a sezioni unite*, num. 26724 and 26725 [Gib07]) which state that in case of violations, like the above, if the client was economically damaged he/she can ask for a compensation and, in the most serious cases, for the cancellation of the contract between the client and the intermediary. This will be transparent to the seller, who will not be involved in the quarrel and will have no consequences (specifically he/she will not have to give money back). 2CL protocol specification allows to capture precisely this situation, thanks to the separation of the constitutive and regulative parts.

In the following, we first provide the definition of the sale protocol. We then model the MiFID principles focusing on the offer of investment services off-site. This applies when a bank promotes and sells financial products with the help of external collaborators (called "tied agents" or intermediaries). We finally show how MiFID grafts onto the sale protocol [BBPM12, BBMP11].

7.3.1 Pre-MiFID sale protocol

As for the OECD case study, let us begin by presenting a sales protocol $S = \langle Ro_S, F_S, s_S, A_S, Cst_S \rangle$, that is used before the introduction of MiFID. In the sales process, the set of roles Ro_S is made of an investor (inv), an intermediary (the financial promoter fp), and a bank ($bank$). The task of the financial promoter is to assist the investor to find an investment that satisfies his/her needs. For this reason, in this model

(a) propose_solution **means** proposed_RiskL
(b) reject_proposal **means** rejected_proposal $\wedge$
    RELEASE(C(fp, inv, invested)))
    if $\neg$accepted_proposal $\wedge$ proposed_RiskL
(c) sign_order **means** CREATE(C(inv, bank, contract_ended)) $\wedge$
    accepted_proposal $\wedge$ order_signed
    if proposed_RiskL $\wedge$ $\neg$rejected_proposal
(d) countersign_contract **means** contract_countersigned $\wedge$
    CREATE(C(bank, inv, executed_order)) $\wedge$ invested
    if order_signed
(e) send_contract **means** contract_sent
    if contract_countersigned
(f) notify **means** notified
    if contract_countersigned $\wedge$ $\neg$contract_ended
(g) end **means** executed_order $\wedge$ contract_ended
    if contract_sent $\wedge$ $\neg$contract_ended $\wedge$ $\neg$contract_abort
(h) withdraw **means** contract_abort
    RELEASE(C(bank, inv, executed_order))
    CANCEL(C(inv, bank, contract_ended))
    if contract_sent $\wedge$ $\neg$contract_ended $\wedge$ $\neg$contract_abort

Table 7.4: Constitutive specification of the Pre-MiFID sale protocol.

the initial state s_S contains a commitment: C(fp, inv, invested). It represents the engagement of the intermediary to the investor to find a good investment.

We now define the set of actions A_S (Table 7.4), making the constitutive specification of the sales protocol.

PROPOSE SOLUTION *(Table 7.4(a)).* With the action *'propose_solution'* the intermediary presents a selected financial product to the investor. The fact proposed_RiskL captures that the investor has received a proposal which is catalogued at a precise level of risk.

REJECT PROPOSAL *(Table 7.4(b)).* Once the proposal is submitted to the investor, then he/she is free to decide to accept or reject it. The case in which he/she decided to reject it is captured by the action *'reject_proposal'*. The effects of this action are the assertion of the fact rejected_proposal and the release of the commitment from the promoter to make the investor make a good investment. This meaning is achieved when there is a proposal to reject and when the proposal has not been accepted yet, as captured by the precondition.

SIGN ORDER *(Table 7.4(c)).* By means of the action *'sign_order'* the investor signs the order proposed by the financial promoter. This action acquires the intended meaning when the investor has received a proposal and he/she has not rejected it yet ($\neg$rejected_proposal $\wedge$ proposed_RiskL). In this case, the effects of this action are the acceptance of the order and the creation of the commitment C(inv, bank,

$\mathsf{contract\_ended}$) by means of which the investor commits to the bank to carry on the contract. Notice indeed, that the financial promoter assists the client in the task of finding a good investment. Once the proposal is accepted, however, the contract is between the investor and the bank.

COUNTERSIGN CONTRACT *(Table 7.4(d)).* Once the contract is signed by the investor, the bank can perform the action *'countersign_contract'* whose effect is the countersign of the contract, the discharge of the promoter's commitment to make the client invests and the creation of a commitment from the bank to the investor, to execute the order of the investor ($C(\mathsf{bank}, \mathsf{inv}, \mathsf{executed\_order})$).

SEND CONTRACT *(Table 7.4(e)).* By means of the action *'send_contract'* the bank sends a copy of the contract to the investor. In order for the *'send_contract'* to count as sending the investor's copy of the contract this latter must be countersigned by the bank.

NOTIFY *(Table 7.4(f)).* If the bank countersigns a contract then the financial promoter must be notified. This can be done by means of the action *'notify'*. The notification is meant to guarantee the intermediary that in case the sale ends with the bank signing a contract with the investor, then he/she will be notified so that he/she can get his/her commission. In order for the *'notify'* to succeed, the contract must be countersigned ($\mathsf{contract\_countersigned}$) and it makes sense if performed before the end of the contract ($\neg\mathsf{contract\_ended}$).

CONTRACT END *(Table 7.4(g)).* The natural end of the contract is captured by the action *'end'* which causes the discharge of the commitments from the investor and from the bank. Indeed, this action asserts the fact $\mathsf{executed\_order}$ which captures that the bank has made his part by absolving what was established in the contract. Analogously, the fact $\mathsf{contract\_ended}$ expresses that the investor has made his/her part and reached the end of the contract. The contract can end only if it was sent to the investor. In this way he/she is able to check that the bank has absolved all its duties. Moreover, in order for this action to have the specified meaning, the contract should not be already ended.

WITHDRAW *(Table 7.6(h)).* An investor has the possibility to withdraw a contract by means of the action *'withdraw'*. The consequences of performing this action are that the contract is aborted ($\mathsf{contract\_abort}$), that the bank is released from its commitment to execute the order and that the commitment of the investor to end the contract is cancelled. These effects are achieved only in when there is a contract to end, i.e. when the contract has been sent to the investor ($\mathsf{contract\_sent}$), and it is not concluded yet ($\neg\mathsf{contract\_ended} \wedge \neg\mathsf{contract\_abort}$).

(c1)	notified *before* contract_ended
(c2)	contract_sent *response* notified

Table 7.5: Regulative specification of the Pre-MiFID sale protocol.

Sales regulations

The protocol also includes a few temporal regulations that are part of the set of constraint Cst_S in the specification. They basically concern temporal requirements affecting the notification (see Table 7.5). These constraints aim at ruling the sending of the notification from the bank to the financial promoter. Constraint (c1) requires that the financial promoter is notified before the end of the contract. Constraint (c2) expresses that if the contract is sent to the investor then the bank has to notify the financial promoter. This means that the promoter can be notified even before the contract is sent, but in any case it cannot happen that the promoter is not notified. Notice that, substituting these constraints with action preconditions (in particular that the contract was sent before notifying the intermediary) does not allow this flexibility and has the further disadvantage that it does not compel the bank to notify the intermediary. Imagine, indeed, to modify the action *'notify'* by adding the precondition contract_sent:

(f) notify **means** notified
if contract_countersigned $\wedge$ $\neg$contract_ended $\wedge$ **contract_sent**

In this case, the precondition makes the action *'notify'* reach the desired meaning only after the contract has been sent. However, it does not require the bank to perform this action. In fact, preconditions allow the identification of the context in which actions acquire a certain meaning but they cannot actively cause their enactment.

Figure 7.6 reports the graph of the possible interactions concerning the sale, given the set of actions defined above. States 9, 10 and 11 are yellow. Such color represents that there is at least one pending condition to satisfy. Thus, yellow states cause a violation if the interaction ends at that point. In particular, states number 9 and 10 cannot be final since constraint (c2) requires that once the contract is sent, then the intermediary must be notified (contract_sent *response* notified). In states 9 and 10, the contract has been sent, but the *'notify'* has not occurred yet. State 11, beside being yellow, is represented as a red diamond. Indeed, if the interaction get through that state constraint (c1) is *violated*, since in that state the fact contract_ended holds, without the fact notified having been true before.

7.3.2 The MiFID specification

Similarly to OECD Guidelines, the MiFID directive introduces new regulations. Roughly, the main aims of the regulation concerning the

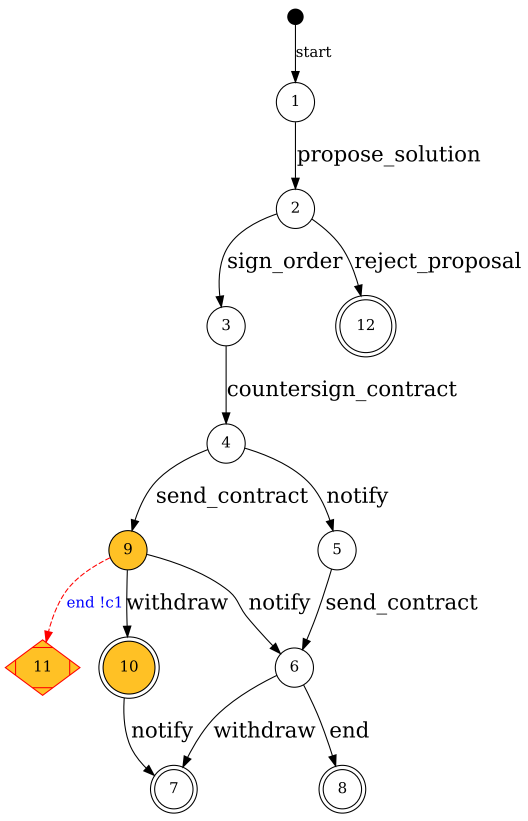


Figure 7.6: Labelled graph of the pre-MiFID sale regulation.

offer of investment services off-site are the identification and the profiling of the investor and the proposal of a *financial instrument* (fi) adequate to his/her needs. Specifically, the principles capturing these requirements are:

- **Identification**: the client must be identified by an identity card or equivalent document;
- **Qualification**: the intermediary supplies all the foreseen documentation about his/her professional qualification and the rules that he/she must stick to;
- **Profiling**: the intermediary must profile the client, gathering information about the balance sheet, knowledge about financial subjects, investment aims. This phase requires the filling of a form, which explicitly specifies the resulting category of the investor. This document should be signed also by the client;
- **Selection**: the proposed financial products must agree with the client's profile. This requires that financial products are classified w.r.t. the different client profiles;

(i)	interview **means** investor_identified $\wedge$ document_supplied **if** $\neg$contract_abort $\wedge$ $\neg$contract_ended $\wedge$ $\neg$rejected_proposal $\wedge$ $\neg$fi_discarded
(j)	profile **means** CREATE(C(fp, inv, evaluation)) $\wedge$ investor_classified **if** investor_identified $\wedge$ $\neg$contract_ended $\wedge$ $\neg$contract_abort $\wedge$ $\neg$rejected_proposal $\wedge$ $\neg$fi_discarded
(k)	classify **means** classified **if** $\neg$contract_abort $\wedge$ $\neg$contract_ended $\wedge$ $\neg$rejected_proposal $\wedge$ $\neg$fi_discarded $\wedge$ $\neg$proposed_RiskL
(l)	fi_evaluation **means** CREATE(C(fp, inv, proposed_RiskL)) $\wedge$ evaluation **if** classified $\wedge$ investor_identified $\wedge$ $\neg$contract_abort $\wedge$ $\neg$contract_ended $\wedge$ $\neg$rejected_proposal $\wedge$ $\neg$fi_discarded

(m)	fi_discard **means** fi_discarded $\wedge$ CANCEL(C(fp, inv, invested)) $\wedge$
	CANCEL(C(fp, inv, proposed_RiskL))
	if evaluation $\wedge$ $\neg$proposed_RiskL $\wedge$ $\neg$contract_abort $\wedge$
	$\neg$contract_ended

(n)	order_verification **means** order_verified $\wedge$
	CREATE(C(bank, inv, executed_order))
	if order_signed

Table 7.6: Constitutive specification of MiFID.

- **Evaluation**: the proposal is evaluated through a simulation: if it is adequate an order is filled and signed both by the client and by the intermediary, otherwise the product is discarded;

- **Verification**: the documentation is sent to the investment trust, which must check that there are no errors or missing data. In case of errors, the documentation is correct and it is sent to the client. Otherwise, it is sent back to the intermediary;

- **Withdrawal**: The client can decide to cancel an order.

Let us now describe how the above MiFID requirements can be modelled in 2CL protocol specification. The actors (role) involved in the interaction are the same as the sale. The set of actions, instead, should be enriched with new, specific actions (reported in Table 7.6).

INTERVIEW *(Table 7.6(i)).* The interview of the client aims at identifying the investor (investor_ identified), and supplying the foreseen documentation (document_ supplied). However, this action makes sense only if performed when the contract is not ended yet, if the investor has not rejected a proposal and if the financial instrument has not been evaluated as non-adequate for the investor profile (fi_discarded).

PROFILE *(Table 7.6(j)).* One of the main goal of the MiFID is to ensure that the investor will receive a financial instrument that is adequate to his/her needs. This objective is reached by classifying both the investor and the financial products according to a certain level of risk. Likely, the more the risk is high the more the possibility to profit from the investment is high. Adopting, instead, a more cautious strategy corresponds to an investment with low risks. By means of the action *'profile'*, the investor is classified ($\mathtt{investor\_classified}$) and the financial promoter commits to evaluate a financial instrument for him/her. This action acquires this meaning when the contract is not ended yet, if the instrument has not been discarded yet and if the investor has been identified.

Classify *(Table 7.6(k))*. In order to find the best solution for a certain client, each financial instrument must be categorised according to the level of risk it entails. This activity can be performed at any time but usually, since it does not require the presence of an investor, it is performed when the system and the financial promoter are not busy. For instance, it can be performed early in the morning or late in the night. It can even be a task performed automatically during the night, but the important aspect is that the classification must be ready when a financial instrument has to be selected, evaluated and proposed to the investor. In the interaction we are modelling, the classification makes sense if performed before the end of the contract, if it is made before a proposal and if the financial instrument has not been discarded yet.

Financial Instrument Evaluation *(Table 7.6(l))*. The evaluation of a financial instrument ($\mathtt{fi}$) is a phase in which the level of risk of the instrument is compared with the investor's profile. This is usually performed by means of a simulation. If the result of the simulation is that the instrument is adequate for the investor, then the financial promoter commits to propose it to the investor ($\mathtt{C(fp, inv, proposed\_RiskL)}$). Moreover, the performance of this action discharges the fp's commitment $\mathtt{C(fp, inv, evaluation)}$ taken during the profiling of the investor, by making the fact $\mathtt{evaluation}$ to hold in the state.

A prerequisite in order to obtain the above effects, is that the financial instruments were classified before the evaluation and that the investor was identified. Otherwise the evaluation has no sense, since it is not possible to compare the levels of risks of the investor and of the product.

Financial Instrument Discard *(Table 7.6(m))*. If the evaluation determines that the financial instrument is not adequate for the investor, then this solution can be discarded by means of the action *'fi_discard'*. When performed, this action cancels the commitments of the financial promoter to make the investor find a good investment ($\mathtt{C(fp, inv, invested)}$) and to propose a solution at the right level of risk ($\mathtt{C(fp, inv, proposed\_RiskL)}$). The proposal and the evaluation of a new product,

(c3)	C(fp, inv, invested) *response* investor_identified ∧ document_supplied
(c4)	investor_classified *before* C(fp, inv, propose_riskL)
(c5)	evaluation *before* proposed_RiskL
(c6)	fi_discarded *not co-exist* proposed_RiskL
(c7)	order_verified *before* contract_countersigned

Table 7.7: Regulative specification of the MiFID protocol.

indeed, can be modelled as a new interaction (potentially starting from a state in which the investor is already identified and products are already classified). The conditions necessary to obtain these effects are that the proposal has been evaluated before (evaluation), the financial promoter has not made the proposal yet (¬proposed_RiskL) and the contract is not ended.

Order Verification *(Table 7.6(n)).* The action *'order_verification'* is an action performed by the bank, while the previous actions were performed by the financial promoter. The aim is to verify the order and, if the bank considers it correct, then it takes the commitment to the investor to execute the order (C(bank, inv, executed_order)). This commitment, however, is taken by the bank only in case the order has been signed (order_signed) by the investor.

MiFID grafting constraints

These actions alone are not enough to implement the directive. Actions *(i–m)* should be executed before the actual sale occurs, while *(n)* completes the sales process. There are two ways to achieve this result. One solution is to modify the action that implements a sale but this is not in the powers of the MiFID regulation, as already explained. The other solution is the identification of a set of 2CL constraints capturing the desired behaviour. The set of constraints we identify for MiFID are reported in Table 7.7. Specifically, constraint (c3) states that once the intermediary took the commitment to serve the investor, he/she must have the investor identified and must supply the necessary documentation to him/her. This is captured by a *response* constraint, that does not impose any order on the achievement of the conditions, but if the commitment is taken during the execution, then at least once after the investor must be identified.

Constraint (c4) expresses the fact that, in order for the financial promoter to select a solution with a certain degree of risk and committing to propose it (C(fp, inv, propose_riskL)), the investor must have been classified *before*. This constraint does not prevent the financial promoter to classify the investor even if he/she does not have a proposal for him/her.

Constraints (c5) and (c6) aim at capturing that before proposing a financial product it is necessary to have it positively evaluated by the simulation. This means requiring that before a solution can be proposed to the investor, the product should have been evaluated (c5) and that the financial promoter cannot propose a solution that is not adequate (c6). A solution not adequate is captured by the discard of the financial instrument. Thus, the constraint (c6) captures that a financial instrument that has been discarded, or is going to be discarded, cannot be proposed as a solution to the investor.

Finally, constraint (c7) requires that before the contract is countersigned by the bank, the data of the order must have been verified.

MiFID grafting on Sales protocol

We define the MiFID specification as $G = \langle Ro_G, F_G, s_G, A_G, Cst_G \rangle$, where the set of roles Ro_G is, as for sales, made by the investor, the bank and the financial promoter. The set of actions A_G is given by the actions listed in Table 7.6 and the set of constraints Cst_G is the set of constraints reported in Table 7.7. The initial state is empty and the set of facts are those involved in the set of actions and constraints of the specification.

Given the Sale Protocol S, previously described, the grafting $S \uplus G$ is formally defined as follow:

$$S \uplus G = \langle Ro_S \cup Ro_G, F_S \cup F_G, s_S \cup s_G, A_S \cup A_G, Cst_S \cup Cst_G \rangle$$

Figure 7.7 depicts how the constraints "zip" sales and MiFID. The arrows going from the MiFID cluster to the sale cluster define how the MiFID regulation grafts onto the sales process. In particular, constraint (c3) ties the initial state (where the commitment $C(fp, inv, invested)$ holds) and the action *'interview'*. Constraints (c5) requires that the action 'fi_evaluation' is performed before the action 'propose_solution'. Similarly, constraint (c7) requires the action 'order_verification' to be performed before the action 'countersign_contract'. Finally, constraint (c6) relates the actions *'fi_discard'* and *'propose_solution'*: being of kind *not co-exist* it states that the two actions cannot be executed both in the same interaction.

Constraints (c1) and (c2) and constraint (c4), instead, are "intra-protocol" regulations acting, respectively, on MiFID activities and on sales activities. They do not act as zippers between the sales protocol and the MiFID.

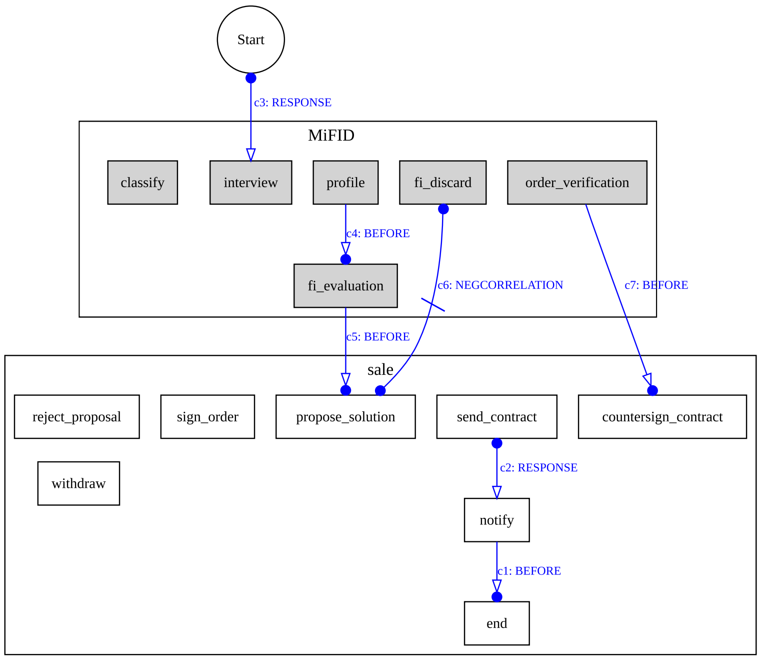


Figure 7.7: MiFID-sale structure graph showing how constraints tide actions.

7.3.3 Analysis of the protocol

Figure 7.8 reports part of the labelled graph for the protocol $S \uplus G$, where only interactions that satisfy all constraints are reported. When a constraint is violated, in the graph it is reported only the first action causing the violation and the resulting state. The complete labelled graph is too big to be fully reported in a readable way (it is available at `http://di.unito.it/2cl`). As for OECD, the legal executions are highlighted in green and gray. The actions of the sales protocol are highlighted in gray, while MiFID is highlighted in green.

The graph is obtained by implementing the protocol $S \uplus G$, following the specification described above (implementation is reported in Section B.5). Specifically, the description of the initial state and the definitions of actions can be implemented exactly as described. For what concerns constraints, instead, in order to meet the assumptions on which our implementation is based on, they cannot be defined directly on commitments. Therefore, we have modified constraints (c3) and (c4) in the following way:

(c3) $\textsc{created}(\mathsf{C}(\mathsf{fp}, \mathsf{inv}, \mathsf{invested}))$ *response*
$\qquad \mathsf{investor\_identified} \wedge \mathsf{document\_supplied}$
(c4) $\mathsf{investor\_classified}$ *before* $\textsc{created}(\mathsf{C}(\mathsf{fp}, \mathsf{inv}, \mathsf{propose\_riskL}))$

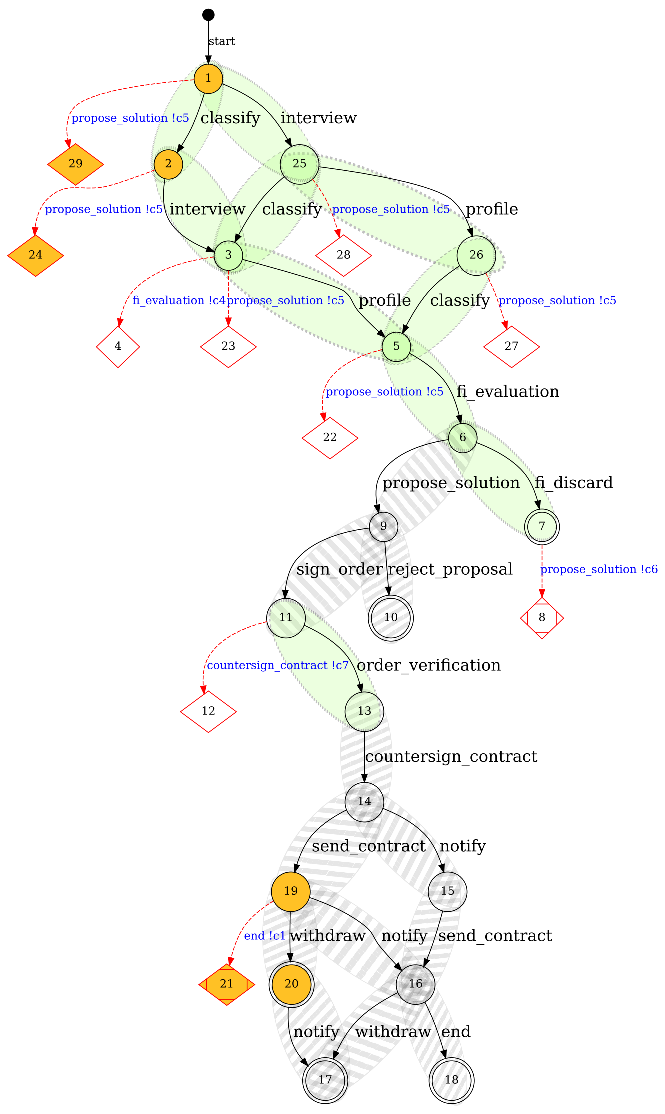


Figure 7.8: Partial view of the MiFID and sale labelled graph.

Roughly, we substituted conditions expressed on commitments with similar conditions expressed on *operations* performed on commitments. In order to do so, we have interpreted the intent of the original constraints, thus choosing the operation *create* as the more appropriate to represent the requirement. Constraint (c3) in its original formulation captures that the commitment of the financial promoter (fp) to the investor to propose him/her an investment entails that, sooner or later, the investor must be identified and the fp has to supply the necessary documentation. We reformulate it by stating that *the creation* of such a commitment triggers the requirement for the financial promoter to achieve the specified conditions. Similarly, constraint (c4) is changed so as to capture that before the commitment from the financial promoter to the investor to propose a certain solution can be created, the financial promoter has to classify the investor.

The designer, by analysing the labelled graph, can identify the points where it could be helpful to intervene to reduce the possible violations, for instance, by applying enforcement policies or by regimenting some steps. For example, one action on which it would make sense to intervene is *'propose_solution'*. The reason is that most of the illegal paths start from a bad use of this action: either because it is performed when the financial instrument has been discarded, or because it is performed before the evaluation. These two conditions ($\neg$fi_discarded $\wedge$ evaluation) can be added in the action definition in the following way:

(a) propose_solution **means** proposed_RiskL
  if $\neg$rejected_proposal$\wedge$
  $\neg$fi_discarded $\wedge$ evaluation

The labelled graph corresponding to this version is reported in Figure 7.9. The zoom on a portion of the graph allows make a comparison with the portion of the portion of labelled graph reported in Figure 7.8. As it is reasonable to expect, the regimented graph is smaller (so it can be reported in one page) and some of the violations caused by the action *'propose_solution'* are no more possible.

Of course, the choice of regimenting these requirements depends on many factors, e.g. the cost of the implementation of the prospected solution, or the time needed to update the software used by financial promoters. A research carried on by the *Cognizant* company has estimated the costs for industries to achieve MiFID compliance between 2006 and 2008, in 1 to 1.5 billion of Euro [Com]. These costs include the complexity of adapting a pre-existing way to carry on these activities in order to be compliant with the directive.

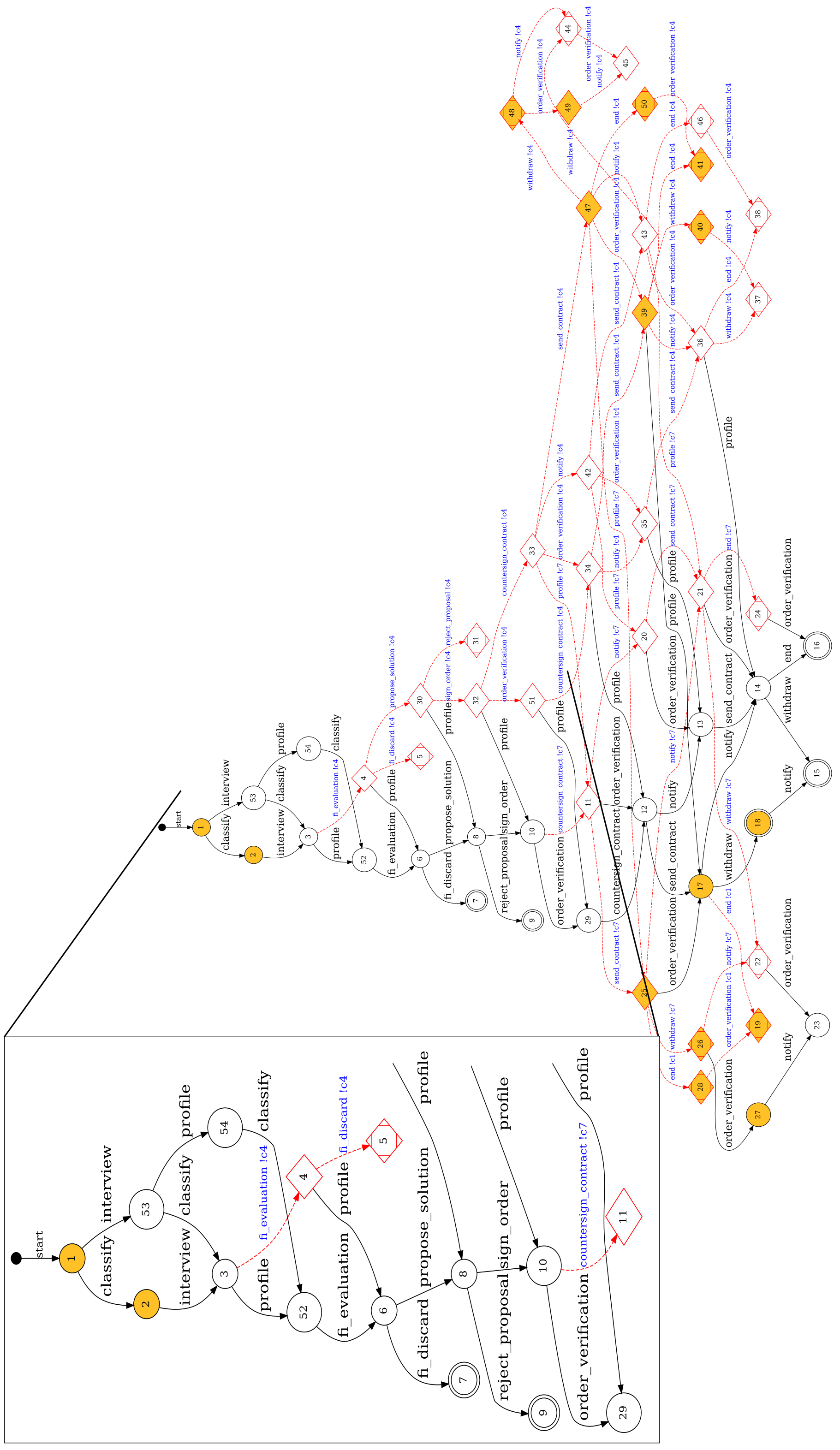


Figure 7.9: Labelled graph of the regimented version of MiFID.

8 CONCLUSIONS

Commitment protocols present some interesting characteristics that make them fit well in the context of multiagent systems, dealing with autonomous agents. Specifically, they allow agents to have flexible behaviours leaving them free to decide which commitments to take and how to satisfy them. Moreover, the verification can be performed by simply observing the ongoing interaction, thus without assuming access to the internal functioning of the interacting agents.

As discussed in Chapters 2 and 3, however, in our opinion there is a need for protocol specification of representing patterns of interaction. Patterns of interaction that commitments alone are not expressive enough to represent. Commitments, indeed, basically capture conditions to be achieved but they do not allow to represent requirements on how agents are desired to satisfy them. To overcome this lack, we propose a protocol specification that explicitly accounts for patterns representation (see Chapter 3). Patterns are expressed as constraints among facts and commitments.

In Chapter 4 we provide an operational semantics of 2CL protocol specifications. Specifically, we define the 2CL-Generalized Commitment Machine (2CL-GCM) obtained as extension of Singh's Generalized Commitment Machine [Sin07]. For its definition we introduce an LTL interpretation of 2CL constraints.

The strengths of a 2CL protocol specification are mainly given by the explicit representation of patterns in the protocol specification and by their decoupling with the protocol actions. As discussed in Chapter 5, this is a difference with current proposals in the literature, which either do not account for patterns representation or they express them without considering the agents' autonomy or the need for reusable and modular specifications. 2CL specifications, instead, are meant to capture only mandatory or forbidden behaviours, leaving agents free to behave as they wish for all the aspects that are not explicitly specified. Moreover, the modularity of 2CL specifications bring an easier modification and extension of the specification, so as to easily allow the introduction of new regulations. Two practical examples are given in Chapter 7, where the MiFID directive and the OECD Guidelines are modelled.

In order to support protocol design and protocol understandability, each 2CL operator is associated to a graphical representation. Moreover, in Chapter 6 we present an Eclipse plug-in that allows for protocol design and analysis. The tool offers different functionalities like the possibility of visualizing the overall graph of possible interactions, labelled according to their compliance or their violation of the protocol specification. The *Graph explorer* functionality allows for the visualization of large graphs by exploring one node at a time. Most of the current proposals aim at verifying properties of interest on the specification or on a single interaction. As a consequence, they do not provide an overview of the possible interactions: either they interrupt the verification when a property that is not verified is found, or they consider one path at a time. We prove also that the labelled graph that we obtain is sound with respect to 2CL-GCM.

ONGOING AND FUTURE DIRECTIONS

When introduced by Castelfranchi [Cas95] and Singh [Sin92, Sin91, Sin99] one of the novelties and main characteristics brought by social commitments was an explicit assignment of a debtor and a creditor of the engagement. This is what distinguishes a social commitment from other forms of engagement, like an individual commitment [Jen93] or a commitment to an intention [CL90]:

> "Social Commitment" is not an individual commitment shared by many agents; it is the Commitment of one agent to another. ([Cas95])

The identification of a *debtor* and a *creditor* agent is important for two main reasons: (i) the debtor agent is identified as *responsible* for the fulfilment of the commitment. If the condition is not achieved it is considered as liable of a violation; (ii) this mechanism provides incentives for the debtor agent to engage a behaviour that leads to the fulfilment of the engagements it has undertaken [CCD98].

2CL *deontic semantics*

Currently, 2CL protocol specification identifies a set of patterns the interaction is desired to respect. However, in its current form, 2CL constraints are not associated to a deontic semantics, therefore the agents' behaviour is not formally bound to them [BB12]. What we meant to investigate is the introduction of a deontic characterization of 2CL constraints. The road we meant to explore follows the idea adopted in the REGULA framework [MBB+11c]. Specifically, we meant to explore the introduction of 2CL constraints as commitment conditions. In this way: (i) constraints acquire a deontic characterization given by commitments; (ii) the regulations that it is possible to capture by 2CL constraints are more expressive than those that can be captured by the

Regula language (see Section 5.1.4); (iii) constraints would be explicitly associated to a responsible. In this way, when a violation occurs the identification of who to blame would be immediate.

This kind of extension entails the extension of commitment's life cycle as well as the adaptation of the notions of safety and control. Moreover, we will investigate whether the introduction of 2CL constraints as conditions of commitments requires the adaptation of the LTL interpretation so as to reconcile the meaning associated to 2CL constraints with the regulative characterization of commitments. Eventually, we will explore the use of other logics and compare them with the currently adopted LTL.

Infrastructure of execution

Another aspect we meant to investigate concerns the realization of a framework for agent's interaction, based on 2CL protocol specification.

One of the selling points of commitment protocols is the possibility of verifying the agents' compliance by simply observing the interaction. This characteristic was proposed as progress with respect to mentalistic approaches. However, as noticed by Baldoni and Baroglio in [BB12], currently an adequate infrastructure for such a verification is still missing. More in detail, in our specification we do not make any assumption on the kind of actions that can be part of a protocol specification. They can be conceived as messages exchanged by the agents but they are not limited to this form of communication (known as *direct* communication). We allow the representation of any action that can be performed by the agents and that has some meaning in the context of a certain interaction. For instance, in the RONR example, raising a hand can be understood as the request of the floor by an agent. This action can be observed by other agents and, if they share the same semantics, all the observers can infer the corresponding meaning. This example represents a form of *indirect* communication. In the literature there are some approaches that allow to cope with indirect forms of communication (for instance by using notifications that act as the observations of the events). Some examples are organizations and institutions [JS96, ERRAA04, ANRAS06, FHSB07, Gro07, HBB10] or what are known as *conversation moderators* [SBH05]. However, these mechanisms basically supply a centralized architecture for interaction monitoring and for moderating the interaction. Many forms of communication, instead, tend to be *peer-to-peer*, thus complicating or limiting the observation of events. Chopra [Cho09] addresses this issue by defining the notion of *commitment alignment*. This approach is at the basis of the architecture proposed by Chopra and Singh in [CS09a]. However, although relaxing the centralization constraint, this approach does not face the verification of the interaction with respect to the patterns specified in the protocol [BB12].

In our view, a suitable abstraction is supplied by the Agents & Artifacts meta-model (A&A) [WOO07, ORV08, RPV11]. It provides the explicit abstractions of *environment* and *artifact*. In particular, this latter is conceived as an entity which can be acted upon, observed, perceived, notified, and so forth. A possible direction we meant to investigate is the one we envisage in [BBB+10, BBM+11c, BBB+11a]. It consists of a new agent-programming framework, named *Mercurio*, where 2CL protocol specifications are incorporated inside the programmable environment foreseen by the A&A meta-model. The aim is to exploit the artifact abstraction so as to: (i) be able to handle both direct and indirect forms of communication; (ii) monitor the on-going interaction so as to detect incorrect interactions and manage them; (iii) implement a distributed mechanism that supports events observation; and (iv) allow protocol composition as composition/cooperation of different artifacts, each capturing different requirements.

Methodology

Finally, in this thesis we have discussed the importance for a specification to be flexible and reusable. The 2CL protocol specification that we propose: (i) allows agents to have flexible behaviours, thus taking into account their autonomy of deciding how to behave; (ii) its declarative nature and the decoupling between the constitutive and the regulative components fosters the reuse of the specification and its adaptation to different contexts; (iii) allows to handle changing regulations and regulations that may arise along time, *grafting* on an existing specification. In general, however, the more the context that the specification aims at ruling is complex, the more the definition of a protocol is difficult to be designed. Flexibility and reuse of the 2CL specifications, together with the tool and the graphical representation that we have presented, make this task easier. However, in order to further support the designer in protocol definition and to foster the use of the 2CL approach, we are currently defining a methodology named 2CM*: Constraints among Commitments Methodology.* 2CM aims at guiding a designer in the specification of protocol from scratch as well as in the definition of new specifications as composition and specialization of existing protocols. 2CM originates from the methodology Amoeba [DCS09]. This latter is generally defined for commitment protocols. We adapted it in order to account for patterns of interactions expressed as 2CL constraints, and we extended it in order to handle protocol specialization.

As a further extension it would be helpful to define a set of guidelines that can be used to "read" a labelled graph. These guidelines should help a designer in the identification of the causes of the violations, possibly highlighting those situations where regimentation could help reducing the risks of violation. We plan also to improve the tool for the generation of the labelled graph, so as to relax some of the assumptions we made. We would like to investigate the use of alternative languages (like the event calculus) and the evaluation of pros and cons with respect to the current use of Prolog.

A LABELLED GRAPH OF POSSIBLE INTERACTIONS: A PROLOG IMPLEMENTATION

The tool we have presented in Chapter 6 can be downloaded at the URL `http://di.unito.it/2cl` together with images and implementations of the examples presented along the thesis. The starting point for generating the *labelled* graph of the possible interactions is the *Enhanced Commitment Machine* by Winikoff *et al.* [WLH05] (available at `http://goanna.cs.rmit.edu.au/~winikoff/CM/`).

The main extensions we introduced are (i) a state evaluation of protocol constraints (as explained in Chapter 6) and (ii) the introduction of propositions related to commitments operations, that are automatically asserted when an operation on a commitments is performed. For the first aspect we mainly extended the `driver.pl` file provided by Winikoff *et al.*This file contains the part of the program responsible for the exploration of the search space and of the generation of the output in a DOT format. For what concerns facts associated to commitment operations, instead, we extended the `newaxioms.pl` file, which contains the set of axioms responsible of the commitments life cycle.

EXPLORATION OF THE SEARCH SPACE

The search space is explored as a depth-first search [WLH05]. As a difference with Winikoff *et al.*'s implementation, our representation of the states contains also a list of *labels*, according to Definition 6.3, which identifies the presence of active commitments and of pending or violated conditions.

The set of reachable states is generated starting from an initial state (potentially empty), by applying protocol actions. Listing A.1 reports the subset of the program that is responsible for this aspect (for completeness, Listing A.2 reports the remaining set of clauses that are used during state exploration). The mechanism is as follows: given a state, *explore* finds the set of possible successors by applying the effects of the actions whose preconditions are satisfied in the state. A state is added only if it is new (not explored yet). Before adding it, *find_labels*

```
go :- firstState, explore(1,2,_).

firstState :-
  state(1,S,_),
  update_first(S,NewS),
  retract(state(1,S,_)),
  labels(NewS,L),
  assert(state(1,NewS,L)).

update_first([],[]).
update_first([c(X,Y,P)|R],[c(X,Y,P),created(c(X,Y,P))|NewS]) :-
        update_first(R,NewS).
update_first([cc(X,Y,Q,P)|R],[cc(X,Y,Q,P),
        created(cc(X,Y,Q,P))|NewS]) :- update_first(R,NewS).
update_first([H|R],[H|NewS]) :- update_first(R,NewS).

compute_next_state([],Add,_Del,Add).
compute_next_state([X|Xs],Add,Del,Res) :- member(X,Del), !,
        compute_next_state(Xs,Add,Del,Res).
compute_next_state([X|Xs],Add,Del,[X|Res]) :-
        compute_next_state(Xs,Add,Del,Res).

nextstate(State,Action,Result) :-
        happens(Action,State),
        findall(Add,initiates(Action,Add,State), AddS),
  write(add(AddS)), nl,
        findall(Del,terminates(Action,Del,State), DelS),
  write(del(DelS)), nl,
        merge_addList(AddS,State,NewState),
        findall(StableProp,
  initiates_stable_prop(Action,StableProp,State,NewState),
  StablePropS),
        merge_addList(AddS,StablePropS,AddList),
        compute_next_state(State,AddList,DelS,New),
        remove_duplicates(New,Result).

explore(StateNum,Free,NextFree) :-
  state(StateNum,State,_),
  findall(t(StateNum,A,S2),nextstate(State,A,S2),Ts),
  add_states(Ts,Free,NextFree), add_transitions(Ts).

labels(State,Labels) :- find_labels(State,[],Labels).

find_labels(S,L1,R) :-
  check_violation(S,L1,L2),
  check_pending(S,L2,L3),
  check_commitments(S,L3,R).
```

Listing A.1: Prolog clauses that generate the labelled graph.

considers the set of constraints given in the specification and checks them on the state.

The described mechanism is performed starting from the initial state. The clause named *firstState* evaluates the protocol constraints and checks the presence of active commitments in the first state. Additionally, if a commitment holds in the state it means that it has been "created"

```
remove_duplicates([],[]).
remove_duplicates([X|Xs],R) :- member(X,Xs), !,
        remove_duplicates(Xs,R).
remove_duplicates([X|Xs],[X|R]) :- remove_duplicates(Xs,R).

merge_addList([],L,L).
merge_addList([X|R],L,[X|New]):- merge_addList(R,L,New).

add_transitions([]).
add_transitions([t(S1,A,S2)|Ss]) :- transition(S1,A,Ss2),
        seteq(S2,Ss2), !, add_transitions(Ss).
add_transitions([t(S1,A,S2)|Ss]) :- state(N2,Ss2,_),
        seteq(Ss2,S2),
        assert(transition(S1,A,N2)), add_transitions(Ss).

add_states([],N,N).
add_states([t(_,_,S)|Ss],N,N1) :-
        state(_,St,_), seteq(St,S), !, add_states(Ss,N,N1).
add_states([t(_,_,S)|Ss],N,N3) :-
        labels(S,L), assert(state(N,S,L)),
        N1 is N+1, explore(N,N1,N2), add_states(Ss,N2,N3).
```

Listing A.2: Additional clauses for the implementation of the search state exploration.

and thus the corresponding proposition CREATED must be added. *update_first* is in charge of this operation.

Constraints evaluation

Constraints are represented as '*constraint*(A, B, Id)', where *constraint* is a 2CL operator, 'A' and 'B' are the antecedent and the consequent conditions of the constraint, and 'Id' is the identifier of the constraint. For instance, '*response*(A, B, id)' stands for a constraint of kind 'A *response* B' which is assigned of the identifier 'id'.

Listing A.3 reports the clauses for constraints evaluation. Pending conditions are verified by the clauses named *check_pending*. These clauses check the presence of response or correlation constraints in the specification. If any, the program checks in every state if the antecedent condition can be derived in the state while the consequent condition cannot. If this is the case, the label '*pending*' is added to the list of labels associated to the state.

The verification of conditions that are violated in a state are demanded to the clauses named *check_violation*. The first clause of this kind is satisfied in a state when there is a constraint of kind *before* such that its consequent condition can be derived in the state, while its antecedent condition cannot. In this case, the label '*violation*' is added to the list of labels associated to the state. Constraints of the kind *negative correlation*, *negative response* and *negative before* are checked in a similar way, except for the fact that the clause checks if both the conditions

```
check_pending(State,L,[pending(Constr)|L]) :-
  response(A,B,Constr),
  consequence(A,State),
  \+consequence(B,State).

check_pending(State,L,[pending(Constr)|L]) :-
  correlation(A,B,Constr),
  consequence(A,State),
  \+consequence(B,State).

check_pending(State,L,L).


check_violation(State,L,[violation(Constr)|L]) :-
  before(A,B,Constr),
  consequence(B,State),
  \+consequence(A,State).

check_violation(State,L,[violation(Constr)|L]) :-
  negcorrelation(A,B,Constr),
  consequence(A,State),
  consequence(B,State).

check_violation(State,L,[violation(Constr)|L]) :-
  negresp(A,B,Constr),
  consequence(A,State),
  consequence(B,State).

check_violation(State,L,[violation(Constr)|L]) :-
  negbefore(A,B,Constr),
  consequence(A,State),
  consequence(B,State).

check_violation(State,L,L).


check_commitments(State,L,[final|L]) :-
  \+member(c(_,_,_),State).

check_commitments(State,L,[non-final|L]) :-
  member(c(_,_,_),State).
```

Listing A.3: Clauses for the labelling of the state according to constraints and commitments evaluation.

(antecedent and consequent) hold in the state. In this case the label 'violation' is associated to the state.

Finally, the program checks the presence of unsatisfied commitments (*check_commitments*) and adds the label *final* or *not-final* consequently. The result of running this program on a protocol specification is a graph of the reachable annotated states. Annotations follow the graphical convention explained in Section 6.3.

```
initiates(E,P,T) :-
        happens(E,T), isFluent(P), causes(E,P).
initiates(E,c(X,Y,P),T) :-
        causes(E,create(c(X,Y,P))), happens(E,T),
        \+(implied(P,T)).
initiates(E,c(X,Y,P),T) :-
        causes(E,create(cc(X,Y,Q,P))), happens(E,T),
        implied(Q,T), \+(implied(P,T)).
initiates(E,cc(X,Y,P,Q),T) :-
        causes(E,create(cc(X,Y,P,Q))), happens(E,T),
        \+(implied(Q,T)), \+(implied(P,T)).
initiates(E,c(X,Y,Q),T) :-
        holdsAt(cc(X,Y,P,Q),T), happens(E,T), subsumes(PP,P),
        initiates(E,PP,T).
%create always adds a created
initiates(E,created(c(X,Y,P)),T) :-
        causes(E,create(c(X,Y,P))), happens(E,T).
initiates(E,created(cc(X,Y,P,Q)),T) :-
        causes(E,create(cc(X,Y,P,Q))), happens(E,T).

terminates(E,c(X,Y,P),T) :-
        holdsAt(c(X,Y,P),T), happens(E,T), subsumes(PP,P),
        initiates(E,PP,T).
terminates(E,cc(X,Y,P,Q), T) :- holdsAt(cc(X,Y,P,Q),T),
        happens(E,T), subsumes(QP,Q), initiates(E,QP,T).
terminates(E,cc(X,Y,P,Q), T) :-
        holdsAt(cc(X,Y,P,Q),T), happens(E,T), subsumes(PP,P),
        initiates(E,PP,T).
terminates(E,P,T) :-
        happens(E,T), causesDel(E,P), holdsAt(P,T).
terminates(E,c(X,Y,P),T) :-
        happens(E,T), causes(E,release(c(X,Y,P))),
        holdsAt(c(X,Y,P),T).
terminates(E,cc(X,Y,Q,P),T) :-
        happens(E,T), causes(E,release(cc(X,Y,Q,P))),
        holdsAt(cc(X,Y,Q,P),T).
terminates(E,c(X,Y,P),T) :-
        happens(E,T), causes(E,cancel(c(X,Y,P))),
        holdsAt(c(X,Y,P),T).
terminates(E,cc(X,Y,Q,P),T) :-
        happens(E,T), causes(E,cancel(cc(X,Y,Q,P))),
        holdsAt(cc(X,Y,Q,P),T).
```

Listing A.4: Clauses responsible of commitment life cycle.

COMMITMENT OPERATIONS' PREDICATES

Listing A.4 reports the set of clauses that implements the commitment life cycle. With respect to Winikoff *et al.* version (`newaxioms.pl`), we modified them in order to add the predicate CREATED to the state, whenever the operation CREATE is performed. When a commitment is created and immediately discharged this is interpreted as both the creation of the commitment and its discharge. This interpretation is adopted both for conditional and unconditional commitments.

```
%The creation of a cc whose antecedent condition holds adds
%a created to the state
initiates_stable_prop(E,created(c(X,Y,P)),State,NewState) :-
        causes(E,create(cc(X,Y,Q,P))), happens(E,State),
        implied(Q,NewState).
%When a cc is detached it adds the created predicate of c
initiates_stable_prop(E,created(c(X,Y,Q)),_,NewState) :-
        holdsAt(cc(X,Y,P,Q),NewState), implied(P,NewState).
%Detach of a conditional commitment
initiates_stable_prop(E,detached(cc(X,Y,P,Q)),_,NewState) :-
        holdsAt(cc(X,Y,P,Q),NewState), implied(P,NewState).
%Create a commitment already satisfied adds the discharged
initiates_stable_prop(E,discharged(c(X,Y,P)),State,NewState) :-
        happens(E,State), holdsAt(c(X,Y,P),NewState),
        implied(P,NewState).
initiates_stable_prop(E,discharged(cc(X,Y,Q,P)),State,NewState):-
        happens(E,State), holdsAt(cc(X,Y,Q,P),NewState),
        implied(P,NewState).
%Discharge
initiates_stable_prop(E,discharged(c(X,Y,P)),_,NewState) :-
        holdsAt(created(c(X,Y,P)),NewState),
        \+member(cancelled(c(X,Y,P)),NewState),
        \+member(released(c(X,Y,P)),NewState),
        \+member(discharged(c(X,Y,P)),NewState),
        implied(P,NewState).
initiates_stable_prop(E,discharged(cc(X,Y,Q,P)),_,NewState) :-
        holdsAt(created(cc(X,Y,Q,P)),NewState),
        \+member(cancelled(cc(X,Y,Q,P)),NewState),
        \+member(released(cc(X,Y,Q,P)),NewState),
        \+member(discharged(cc(X,Y,Q,P)),NewState),
        implied(P,NewState).
%Release and cancel
initiates_stable_prop(E,released(c(X,Y,P)),State,NewState) :-
        happens(E,State), causes(E,release(c(X,Y,P))),
        holdsAt(c(X,Y,P),State), \+implied(P,NewState).
initiates_stable_prop(E,released(cc(X,Y,Q,P)),State,NewState) :-
        happens(E,State), causes(E,release(cc(X,Y,Q,P))),
        holdsAt(cc(X,Y,Q,P),State), \+implied(P,NewState).
initiates_stable_prop(E,cancelled(c(X,Y,P)),State,NewState) :-
        happens(E,State), causes(E,cancel(c(X,Y,P))),
        holdsAt(c(X,Y,P),State), \+implied(P,NewState).
initiates_stable_prop(E,cancelled(cc(X,Y,Q,P)),State,NewState) :-
        happens(E,State), causes(E,cancel(cc(X,Y,Q,P))),
        holdsAt(cc(X,Y,Q,P),State),\+implied(P,NewState).
```

Listing A.5: Additional cases for the assertion of the predicates related to commitment operations.

Listing A.5 reports the set of clauses that manage the remaining cases for the assertion of predicates related to commitment operations. It is basically performed as a comparison between the source state of the transition (the state before the application of the protocol action) and the target state (how the new state would be after the application of the action's effects and before asserting predicates concerning commitment operations). The clauses we reported allow to assert a CREATED when a commitment is created as a result of a detach. This can happen both because the antecedent condition of an existing commitment becomes true (line 3 in the listing) or because a conditional commitment is created in a state where the antecedent condition already holds (line 7).

The clause for the assertion of a DETACHED predicate is reported at line 10. The predicate is asserted when the antecedent condition of a conditional commitment is satisfied.

The discharge of a commitment is handled by different clauses. Clause at line 13 asserts a DISCHARGED when a commitment is created in a state where both a commitment and its consequent condition hold. This does not means that the resulting state will contain both the satisfied commitment and its consequent condition. According to commitment life cycle, indeed, a satisfied commitment should be removed from the state. This simply is performed in a step subsequent to the assertion of propositions related to commitment operations. Clause at line 16, and the similar clause for conditional commitments reported at line 26, are meant to capture those cases in which the consequent condition is achieved after the commitment was created. The idea is that, if a commitment has been CREATED and it has not been removed (CANCELLED, RELEASED or DISCHARGED), if the consequent condition can be inferred, then the commitment is discharged.

The remaining clauses allow to handle the release (lines 33 and 36) and the cancellation (lined 39 and 42) of the commitments. If one of these operations is listed as an effect of the action that causes the state transition, then if the commitment to be removed is present in the state, the corresponding predicate can be asserted.

For completeness, Listing A.6 reports the clauses that are functional to commitment life cycle and predicate assertion mechanism we have described in this section.

```prolog
holdsAt([],_).
holdsAt(not(C),T) :- !, \+holdsAt(C,T).
holdsAt(and(C1,C2),T) :- !, holdsAt(C1,T), holdsAt(C2,T).
holdsAt(or(C1,C2),T) :- !, holdsAtOr(C1,C2,T).
holdsAt(xor(C1,C2),T) :- !, xorHolds(C1,C2,T).
holdsAt(X,T) :- integer(T), !, state(T,Y), membert(X,Y).
holdsAt(X,T) :- membert(X,T).

holdsAtOr(C1,_,T) :- holdsAt(C1,T).
holdsAtOr(_,C2,T) :- holdsAt(C2,T).
xorHolds(C1,C2,T) :- holdsAt(C1,T), \+holdsAt(C2,T).
xorHolds(C1,C2,T) :- holdsAt(C2,T), \+holdsAt(C1,T).

membert(true,[]).
membert(X,[X|_]).
membert(X,[_|Ys]) :- membert(X,Ys).

subseteq([],_).
subseteq([X|Xs],Y) :- member(X,Y), subseteq(Xs,Y).

seteq(A,B) :- subseteq(A,B), subseteq(B,A).

implied(P,T) :- holdsAt(P,T).
implied(c(_,_,P),T) :- implied(P,T).
implied(cc(_,_,_,Q),T) :- implied(Q,T).
implied(cc(X,Y,_P,Q),T) :- implied(c(X,Y,Q),T).

subsumes(P,P).
subsumes(P,c(_,_,PP)) :- subsumes(P,PP).
subsumes(P,cc(_,_,_,PP)) :- subsumes(P,PP).
subsumes(c(X,Y,P),cc(X,Y,_Q,PP)) :- subsumes(P,PP).

happens(E,T) :- isAction(E), precond(E,P), implied(P,T).
```

Listing A.6: Set of clauses functional to the implementation of commitment life cycle.

B EXAMPLES IMPLEMENTATION

In this chapter we report the implementation of the examples adopted along the thesis. We will discuss their specification showing how they can be adapted so as to meet the assumptions we made on the implementation (see Chapter 6). Although the assumptions we made, indeed, we still be able to handle many examples and real case studies. In this chapter we will show also how the labelled graph can support some kinds of analysis on the protocol specification. All the examples and the images of the labelled graphs are downloadable at the URL http://di.unito.it/2cl.

As a general consideration, in the implementations described in this chapter we have added preconditions to the actions, so that they cannot be executed more than once in an execution. Given the characteristic of facts to persist in a state, indeed, the execution of the same action more than once on a path does not add new predicates to the state. Thus, it would result in a self-loop. The omission of this information improves the readability of the labelled graphs and does not impact on the analysis of risk of violations (self-loops are not significant to determine which actions lead to a violation of a constraint).

The adaptations we will made on the original specifications basically amount to transform constraints given directly on commitments, into constraints given on predicates expressing operations performed on commitments.

B.1 THE ROBERT'S RULES OF ORDER

In this section we report the implementation of the Robert's Rules of Order Protocol (whose specification is described in Section 3.2).

Constitutive Specification. The constitutive specification is reported in Listing B.1. It is obtained from the specification given in Section 3.2, by simply adapting the syntax so that it can be handled by the parser that is part of the tool.

```
motion
  means
    motion_introduced, create(c(ch,p,cfv))
  if
    !motion_introduced.

open_debate
  means
    create(cc(ch,p,c(p,ch,discussed),assign_floor)),
    debate_opened
  if
    motion_introduced & !debate_opened.

refuse_floor
  means
    refused_floor,
    release(cc(ch,p,c(p,ch,discussed),assign_floor)),
    release(c(ch,p,assign_floor)), cancel(c(p,ch,discussed))
  if
    motion_introduced & !refused_floor.

ask_floor
  means
    create(c(p,ch,discussed)), asked_floor
  if
    debate_opened & !asked_floor.

recognition
  means
    assign_floor
  if
    debate_opened & !assign_floor.

start_talk
  means
    talk_started
  if
    !talk_started.

stop_talk
  means
    talk_ended, discussed
  if
    talk_started & !talk_ended.

time_out
  means
    talk_ended, discussed
  if
    talk_started & !talk_ended.

cfv
  means
    cfv
  if
    motion_introduced & !cfv.

```

Operator	2CL *representation*	*Implementation representation*
correlate	•—	.-
not correlate	•-/-	./-
co-exist	•—•	.-.
not co-exist	•-/-•	./.
response	•—▷	.->
not response	•-/-▷	.-/>
before	—▷•	->.
not before	-/-▷•	->
cause	•—▷•	.->.
not cause	•-/-▷•	.-/>.

Table B.1: Mapping between 2CL operators and the corresponding symbols used in the implementation.

```
vote
  means
    voted
  if
    motion_introduced & !voted.
```

Listing B.1: Constitutive Specification of the Robert's Rules of Order Protocol: Implementation.

Regulative Specification. For what concerns the regulative specification, we modify the set of constraints reported in Section 3.2.1 so as to meet the assumptions we made for our implementation. In this way we can perform the state evaluation we have described in Section 6.1. Specifically, we have to transform constraints so as to be expressed in terms of positive predicates only. In the original specification, the set of constraints given in terms of commitments are the following:

(c2) refused_floor *not response* C(p, ch, discussed)
(c4) C(p, ch, discussed) *before* assign_floor

As already discussed, in order to transform a constraint on commitment into a constraint on positive predicates we need to consider the aim the constraint was conceived for. The aim of (c2) was to capture that once a participant has refused the floor it cannot change its mind by committing to discuss it. According to this interpretation, we represent this constraint as:

```
refused_floor .-/> created(c(p,ch,discussed)).
```

where symbol .-/> represents the *not response* operator (Table B.1 reports the mapping between 2CL operators and the corresponding symbols used in the implementation). The meaning of the constraint is that if the participant has refused the floor it cannot also declare its interest in discussing the motion by creating the commitment $\mathsf{C}(\mathsf{p}, \mathsf{ch}, \mathsf{discussed})$. Therefore, the constraint is similar to (c2), but instead of defining it in terms of the commitment, we have defined it in terms of the predicate associated to its creation.

For what concerns constraint (c4), the aim was to express that once the participant has stated its intention to discuss the motion, by committing to do so, then the chair must assign it the floor. Moreover, since the operator is of kind *cause*, the floor cannot be assigned to a participant before its has requested it. Again, the interpretation we associate to this constraint is that once the participant has created the commitment to discuss the motion, then the floor must be assigned to it. The resulting constraint is:

```
created(c(p,ch,discussed)) .->. assign_floor.
```

All the remaining constraints are already defined on positive predicates. The resulting set of constraints for RONR specification is reported in Listing B.2.

```
motion_introduced .-> (debate_opened | refused_floor).
refused_floor .-/> created(c(p,ch,discussed)).
discussed ./. refused_floor.
created(c(p,ch,discussed)) ->. assign_floor.
assign_floor .->. talk_started.
talk_started .- talk_ended.
refused_floor | discussed ->. cfv.
cfv ->. voted.
```

Listing B.2: Regulative specification of Robert's Rules of Order: Implementation.

Analysis of the graph. In Figure B.1 is reported a partial view of the Robert's Rules of Order labelled graph. The picture depicts all the legal paths resulting from the specification, and the possible violations that may occur. For readability, only the first state of a violation is reported.

By analysing the graph and the possible violations that may occur, one can notice that many of them are caused by the action `start_talk`. Given its definition in the specification, indeed, it basically can be performed at every moment. The precondition to this action is added just to avoid self-loops, therefore preventing its execution when already executed once on the same path. This definition reflects what happens in many real cases, where it is not possible to oblige people to silent. In these cases, the designer (or the judge, the chair and such like) can only be aware of the possible violations and envisage enforcement mechanisms like sanctions and punishment (obliging people that speak with-

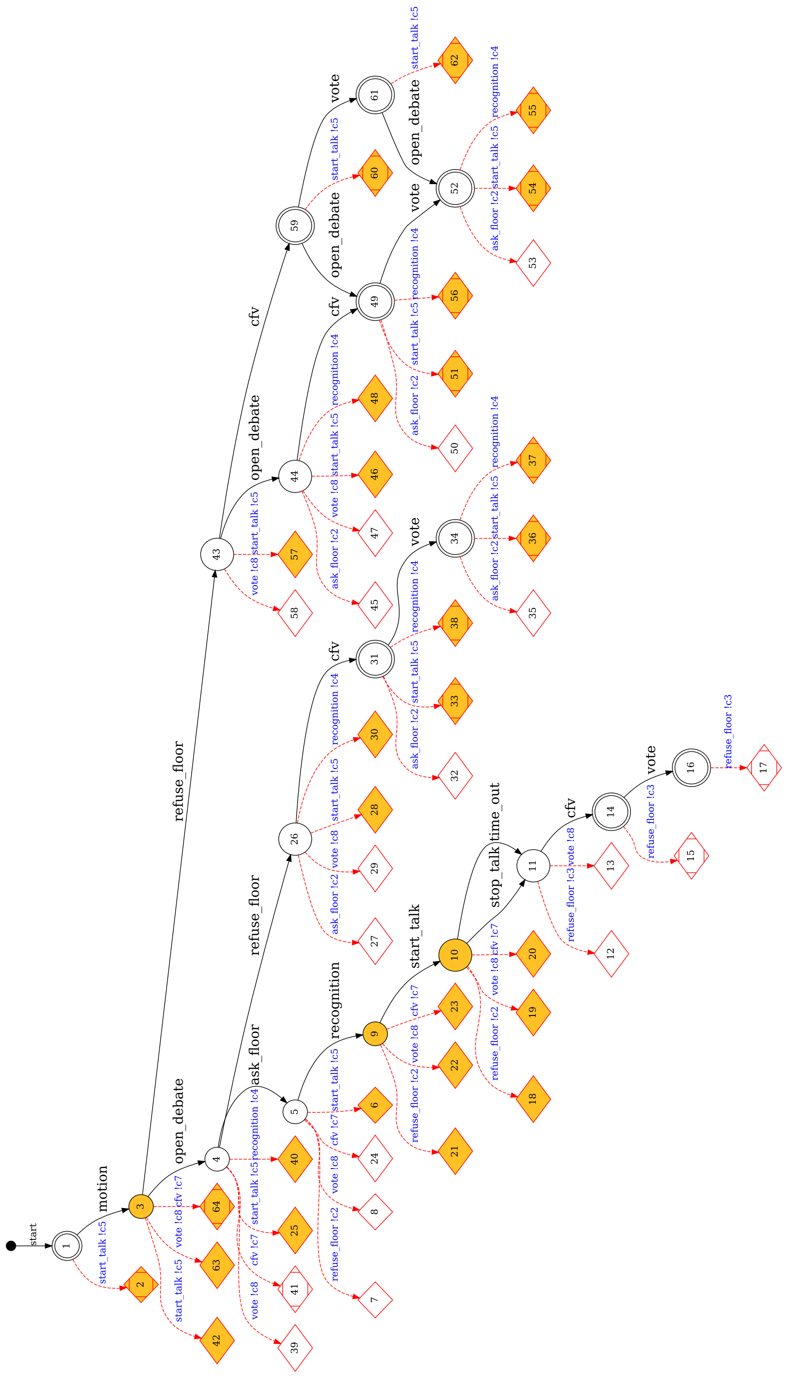


Figure B.1: Partial view of Robert's Rule of Order labelled graph.

out being allowed to quit the assembly, or by establishing a fine to pay and so on). In case the participants speak by means of a microphone, instead, it is possible to hush those who are not allowed to speak. This practically corresponds to ignoring what they are saying.

By analysing the legal paths, it is possible for a designer to determine whether the allowed behaviours conform to the behaviours he/she expected from the specification. In other words, the graph is a means for checking the correctness of the protocol specification. For instance, one can notice that once the floor has been refused, the participant is not allowed to change its mind and to ask for the floor (transition from state 26 to state 27 due to action *ask_floor* leads to a violation). This, indeed, is the requirement expressed by constraint (C2) of the specification. If this requirement was not in the intention of the designer, he/she can discover the misbehaviour and change the specification accordingly.

For what concern the interacting parties, the graph can be used as a support for the chair and the participants in order to understand not only which actions they can execute without violating any constraint, but also in order to understand which paths allow them to satisfy their commitments.

B.2 NETBILL PROTOCOL

In this section we report the implementation of NetBill protocol (whose specification is described in Section 3.4.1).

Constitutive Specification. The constitutive specification of NetBill protocol is obtained by adapting the syntax of the specification reported in Section 3.4.1 and by adding preconditions so as to avoid the representation of self-loops.

```
sendRequest
  means
    request, sendRequest
  if
    !quote & !goods & !sendRequest.

sendQuote
  means
    quote, create(cc(mer, cus, cc(cus, mer, goods, pay), goods)),
    create(cc(mer, cus, pay, receipt)), sendQuote
  if
    !sendQuote.

sendAccept
```

```
  means
    create(cc(cus, mer, goods, pay)), sendAccept
  if
    !pay & !sendAccept.

sendGoods
  means
    goods, create(cc(mer, cus, pay, receipt)),
    sendGoods
  if
    !sendGoods.

sendEPO
  means
    pay, sendEPO
  if
    !sendEPO.

sendReceipt
  means
    receipt, sendReceipt
  if
    pay & !sendReceipt.
```

Listing B.3: Constitutive Specification of the NetBill Protocol: Implementation.

Regulative Specification. For what concerns the regulative specification of NetBill, given the specification described in Section 3.4.1 there are two constraints that are defined directly on commitments. In order to guarantee the correctness of their verification, as explained in Chapter 6, we need to adapt them so as to being defined in terms of positive predicates only. Thus, we discuss how to rewrite them in terms of operations performed on commitments. The two constraints are the following:

(c1) quote *before* $C(c, m, \mathsf{goods}, \mathsf{pay}) \lor C(c, m, \mathsf{pay})$
(c2) $C(m, c, \mathsf{pay}, \mathsf{receipt}) \land \mathsf{goods}$ *before* pay

The role of constraint (c1) is to capture that before the customer can commit to pay for the goods, it should have received a quote from the merchant. The way to achieve this result is to transform the conditions on commitments into conditions on their creation. In this way, the requirement that is captured is that the customer cannot create a commitment to pay if it does not have received a quote. The resulting constraint is:

```
quote ->. created(cc(cus, mer, goods, pay)) |
                created(c(cus, mer, pay)).
```

Constraint (c2) is transformed in a similar way. Specifically, the intended meaning was to require that before the customer pays, the merchant should commit to send the receipt and should have sent the goods (as described in the Example 6.1). The constraint that capture this requirement is:

```
created(cc(mer, cus, pay, receipt)) & goods ->. pay.
```

The final set of constraints for RONR specification is reported in Listing B.4.

```
quote ->. created(cc(cus, mer, goods, pay)) |
  created(c(cus, mer, pay)).
created(cc(mer, cus, pay, receipt)) & goods ->. pay.
pay .->. receipt.
quote ->. pay.
```

Listing B.4: Regulative specification of NetBill protocol: Implementation.

Analysis of the graph. Figure B.2 reports the legal paths for NetBill protocol and the possible violations. As can be noticed, the payment (*send_EPO*) is allowed only after the shipment of the goods (*send_-goods*). Indeed, transition from state 2 to state 14, 9 to 13, 19 to 31 and 26 to 30 are marked as transitions that violate constraint (c2). This is the behaviour captured by the constraints we have just described above, in accordance with NetBill definition (see the sequence diagram reported in Figure 3.3). However, this basically represents one among the possible ways of carry on a sale of some goods. Other contexts may allow to pay before goods are shipped. From the point of view of protocol reuse, by analysing the graph a designer can determine whether a given protocol captures the desired behaviours as is, or if it needs to be modified. For instance, one can think to relax the constraints that forbid the payment to occur before the shipment. Figure B.3 is obtained by relaxing constraint (c2). As can be noticed, in this case the payment is allowed to occur before the shipment (arrows labelled *send_EPO*, starting from nodes 2, 9, 25 and 32 are in this case black and solid lines).

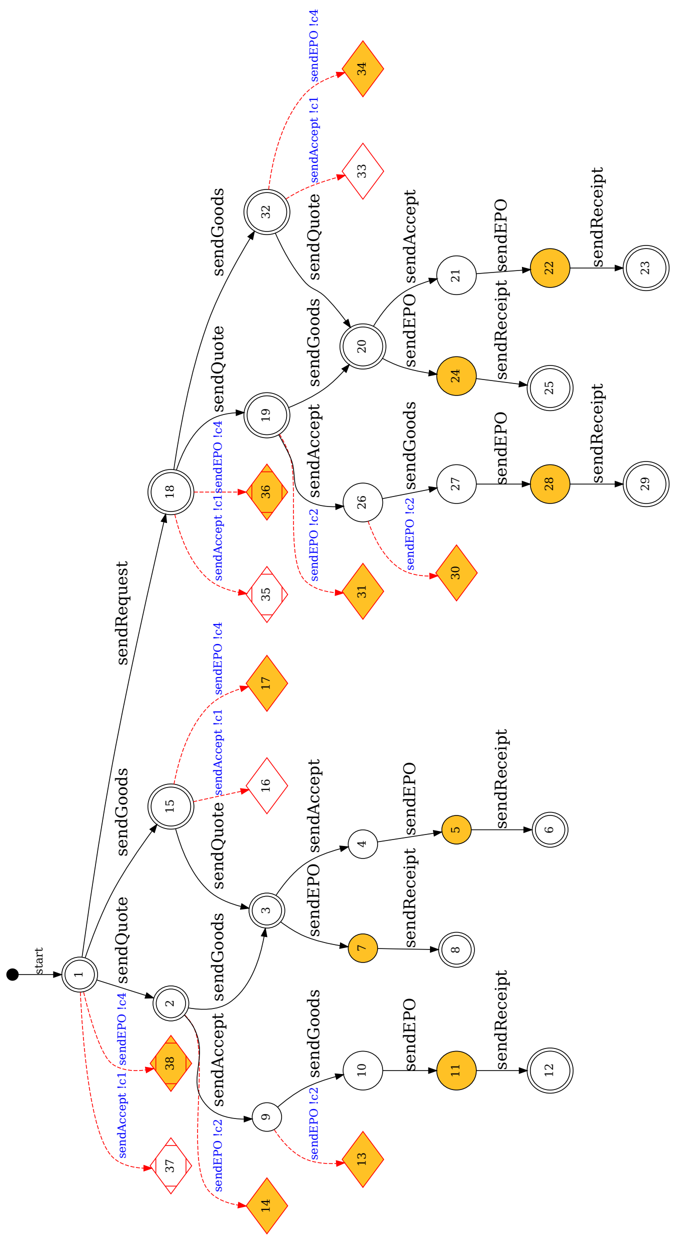


Figure B.2: Partial view of NetBill protocol labelled graph.

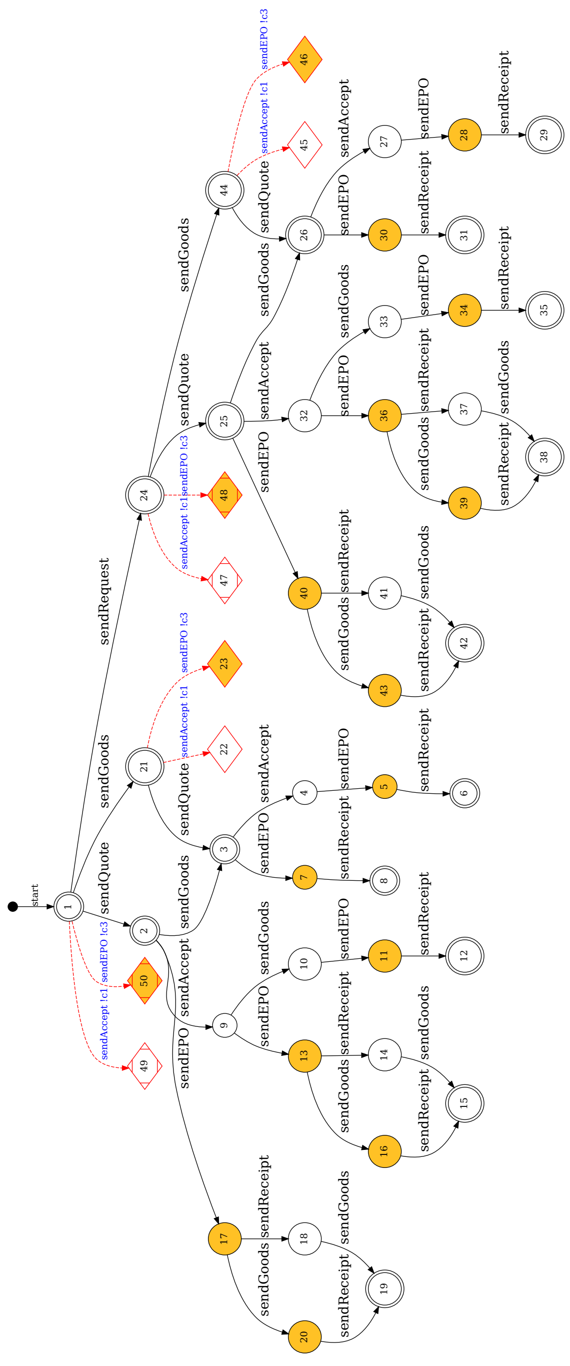


Figure B.3: Partial view of NetBill protocol labelled graph after relaxing constraint (c2).

B.3 CONTRACT NET PROTOCOL

In this section we report the implementation for the CNET specification given in Section 3.4.2.

Constitutive Specification. The constitutive specification reported in Listing B.5 is obtained by simply adapting the syntax of CNET specification given in Section 3.4.2.

```
send_cfp
  means
    cfp
  if
    !failed & !solved & !cfp.

send_proposal
  means
    create(cc(p,i,assigned,solved)), proposal
  if
    !solved & !proposal.

send_refusal
  means
    refused_task
  if
    cfp & !solved & !refused_task.

send_accept
  means
    assigned
  if
    !assigned.

send_reject
  means
    release(cc(p,i,assigned,solved)),
    release(c(p,i,solved)), rejected_proposal
  if
    proposal & !rejected_proposal.

send_done
  means
    solved
  if
    !solved.

send_failure
  means
    cancel(cc(p,i,assigned,solved)),
    cancel(c(p,i,solved)), failed
  if
```

```
    !failed.
```

Listing B.5: Constitutive Specification of the CNET Protocol: Implementation.

Regulative Specification. For what concerns the regulative specification, the set of constraints defined directly in terms of commitments are:

(c1) $\mathtt{cfp}$ *cause* $\mathtt{refused\_task}(p, i)$ OR
$\quad C(p, i, \mathtt{assigned}(i, p), \mathtt{solved}(p, i))$

(c2) $C(p, i, \mathtt{assigned}(i, p), \mathtt{solved}(p, i))$ *cause*
$\quad \mathtt{rejected\_proposal}(i, p)$ OR $\mathtt{assigned}(i, p)$

(c4) $\mathtt{refused\_task}(p, i)$ *not co-exist*
$\quad C(p, i, \mathtt{assigned}(i, p), \mathtt{solved}(p, i))$

The way these constraints can be transformed so as to be defined in terms of operations performed on commitments is similar to that described for the other examples. Consider constraint (c1). It requires that once a call for proposal is issued by the initiator, then (and only after) the participant is expected to answer with a refusal or by committing to solve the task. Thus, the way the constraint can be transformed is by considering the predicate CREATED associated to the commitment. Similarly, for constraint (c2) the intended meaning was to require that once the participant has taken the commitment to solve the task, then the initiator replies by refusing its proposal or by assigning it the task. Finally, given constraint (c4) we can interpret it as a means to avoid both the refusal of solving a task by the participant and the *creation* of the commitment to solving it (both its conditional and unconditional form).

According to the interpretation described above, the regulative specification for CNET protocol that we implemented is reported in Listing B.6.

```
(c1) cfp .->. created(cc(p,i,assigned,solved)) | refused_task.
(c2) created(cc(p,i,assigned,solved)) .->.
        rejected_proposal | assigned.
(c3) assigned .->. solved | failed.
(c4) created(cc(p,i,assigned,solved)) |
        created(c(p,i,solved)) ./. refused_task.
(c5) rejected_proposal ./. assigned.
(c6) solved ./. failed.
```

Listing B.6: Regulative specification of CNET protocol: Implementation.

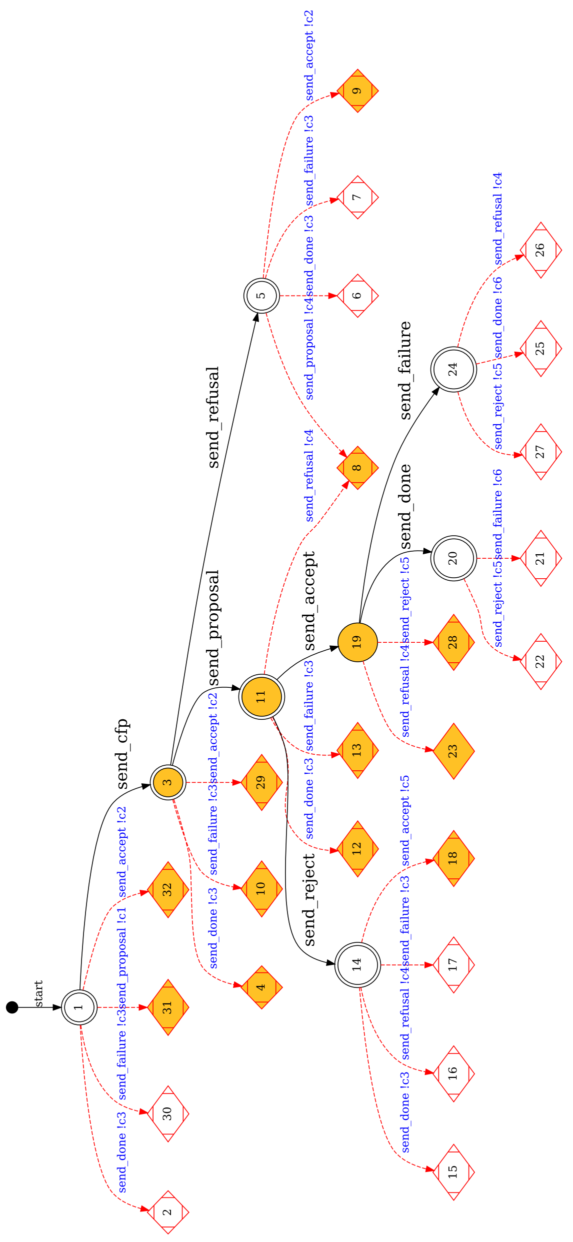


Figure B.4: Partial view of the labelled graph of CNET protocol.

Analysis of the graph. Figure B.4 reports the labelled graph of CNET protocol. Specifically, it reports the legal paths and the possible violations that may occur. For readability, the graph reports only the first state of a violation. As can be noticed, the legal paths allowed by the protocol specification are exactly those specified by the FIPA CNET specification. They, indeed, are the same sequences provided in the sequence diagram depicted in Figure 3.5. Although capturing the same requirements, our specification is given in a declarative way. Thus, as already discussed, it results to be more flexible and adaptable than the FIPA procedural specification. By describing the other variants of CNET protocol we are able to practically show how easily a specification can be adapted to different needs.

Let us consider, as a first example, the Lazy participant variant.

B.3.1 Lazy Participant Variant

The lazy participant variant differs from the CNET specification because it does not require a participant to answer to a call for proposal. As well as CNET, however, a proposal or a refusal can be done only after the call. These requirements, as explained in Section 3.4.3 can be obtained by transforming constraint (c1) from a *cause* to a *before*. The resulting regulative specification is reported in Listing B.7.

```
(c1) cfp —>. created(cc(p,i,assigned,solved)) | refused_task.
(c2) created(cc(p,i,assigned,solved)) .—>.
        rejected_proposal | assigned.
(c3) assigned .—>. solved | failed.
(c4) created(cc(p,i,assigned,solved)) |
        created(c(p,i,solved)) ./. refused_task.
  (c5) rejected_proposal ./. assigned.
(c6) solved ./. failed.
```

Listing B.7: Regulative specification of the Lazy Participant Variant of CNET protocol: Implementation.

Analysis of the graph. The labelled graph reported in Figure B.5 reports exactly the same sequences of the CNET labelled graph (Figure B.4), except for state number 3. In the lazy participant variant, indeed, the execution of the action *send_cfp* does not causes the activation of a pending condition (indeed it is white coloured), as happens, instead, in the CNET version. Thus, the interaction can end in state 3 without causing any violation.

Another variant is given by the zealous participant variant. Let us describe it.

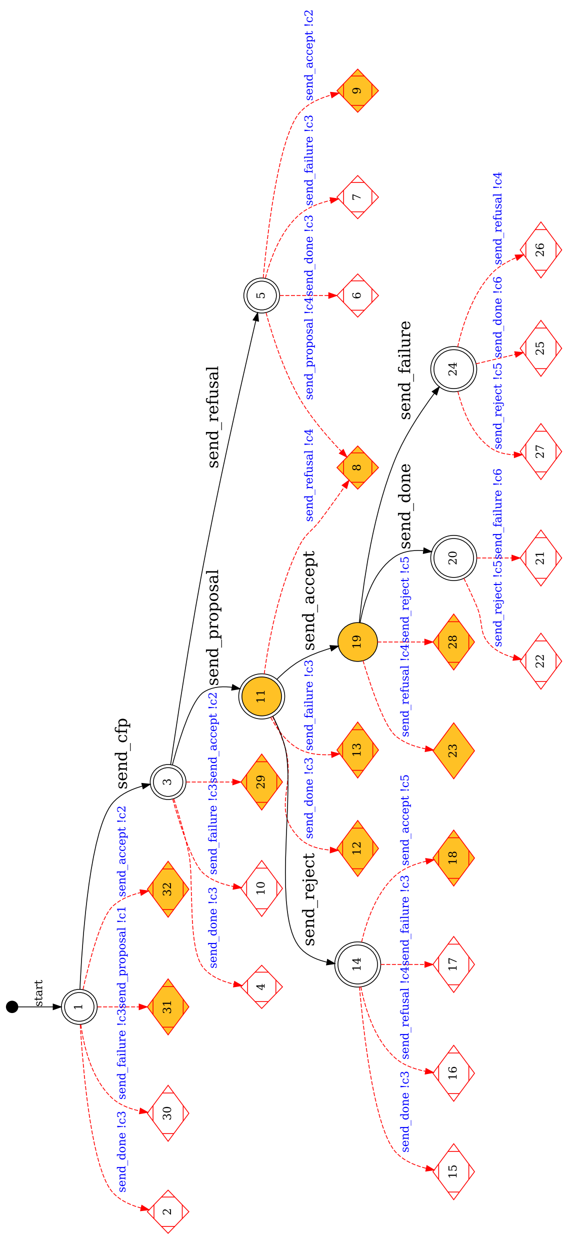


Figure B.5: Partial view of the labelled graph of Lazy Participant variant of CNET.

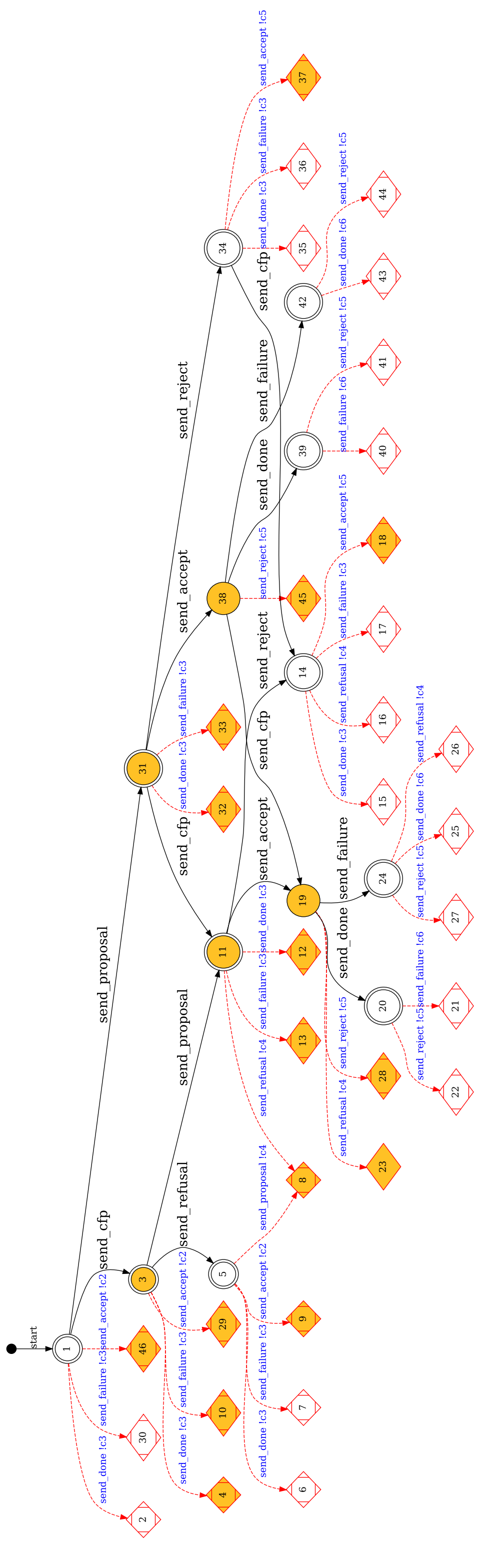


Figure B.6: Partial view of the labelled graph of Zealous Participant variant of CNET.

B.3.2 Zealous Participant variant

The Zealous participant variant has been introduced so as to allow a participant to propose a certain solution before a call for proposal. As a difference to the lazy participant variant, once a call for proposal is made, the participant is required to answer (either with a refusal or with a proposal). This behaviour can be simply obtained by transforming constraint (C1) of the original specification from a *cause* to a *response*.

```
(c1) cfp .-> created(cc(p,i,assigned,solved)) | refused_task.
(c2) created(cc(p,i,assigned,solved)) .->.
       rejected_proposal | assigned.
(c3) assigned .->. solved | failed.
(c4) created(cc(p,i,assigned,solved)) |
       created(c(p,i,solved)) ./. refused_task.
(c5) rejected_proposal ./. assigned.
(c6) solved ./. failed.
```

Listing B.8: Regulative specification of the Zealous Participant Variant of CNET protocol: Implementation.

Analysis of the graph. As can be noticed by the graph reported in Figure B.6, the zealous participant variant allows for more interactions than the CNET specifications described before. In this variant, a participant is allowed to immediately send a proposal (transition from state 1 to state 31 is legal). The call for proposal can be issued after it or it can never occur during the interaction.

B.3.3 CNET with Anticipated Failure

The CNET with anticipated failure was introduced in order to allow a participant to declare that it will not be able to perform a certain task. This behaviour is obtained by transforming, with respect to the original specification, constraint (C1) into a *before*, by transforming constraint (C3) from a *cause* to a *response*, and by a *before* constraint that requires a solution to be sent only if *assigned* holds (constraint (C7) below).

```
(c1) cfp ->. created(cc(p,i,assigned,solved)) | refused_task.
(c2) created(cc(p,i,assigned,solved)) .->.
       rejected_proposal | assigned.
(c3) assigned .-> solved | failed.
(c4) created(cc(p,i,assigned,solved)) |
       created(c(p,i,solved)) ./. refused_task.
```

```
(c5) rejected_proposal ./. assigned.
(c6) solved ./. failed.
(c7) assigned ->. solved.
```

Listing B.9: Regulative specification of the CNET with Anticipated Failure variant of CNET protocol: Implementation.

Analysis of the graph. As can be noticed from the graph reported in Figure B.7, the given specification allows to capture exactly the desired behaviour: the execution of the action *send_failure* does not cause a violation as instead happens for the other variants. The action *send_done*, instead, is allowed only after the initiator has accepted a proposal from the participant. This is because by this action the participant sends the solution of the task. The protocol aim to forbid the participant to send solutions if the task is not assigned to them (as for instance shown by violations in states 4, 12, 32).

B.3.4 Soft CNET

Finally, we report the implementation for the "soft" variant of CNET. With respect to the other variants, this specification provides only few constraints (reported in Listing B.10). As a result, the allowed executions are too many to be reported in a comprehensible way in this thesis. This is an example in which the designer can take advantage from the Graph explorer functionality (as described in Section 6.4). By means of this tool, he/she can explore one state at a time, thus checking whether the set of allowed executions correspond to those it expected from the protocol specification he/she has provided. If this is not the case, he/she can update the specification and check it until the desired specification is found.

```
created(cc(p,i,assigned,solved)) ->. assigned.
created(cc(p,i,assigned,solved)) |
  created(c(p,i,solved)) ./. refused_task.
rejected_proposal ./. assigned.
```

Listing B.10: Regulative specification of soft CNET variant: Implementation.

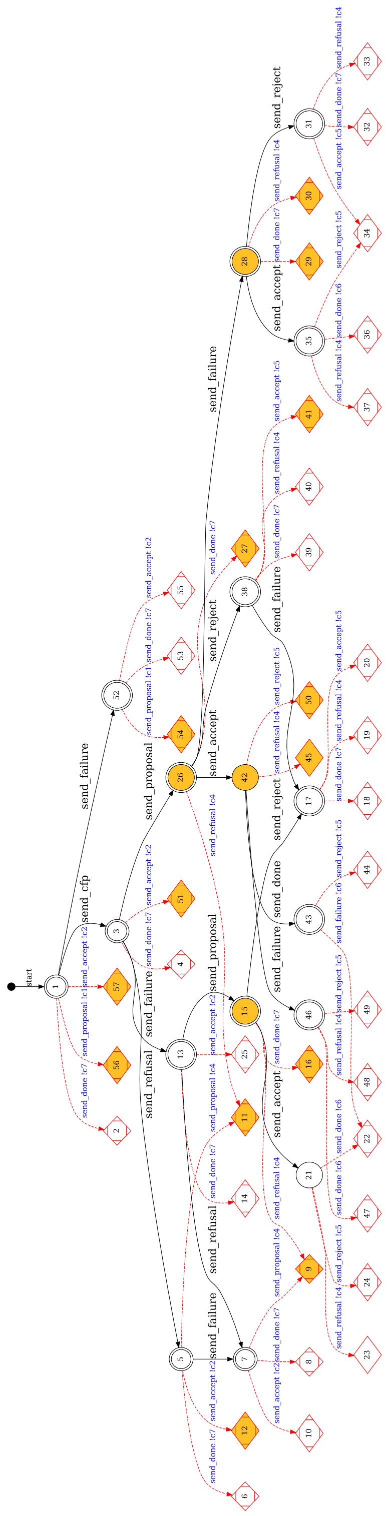


Figure B.7: Partial view of the labelled graph of CNET with Anticipated Failure variant.

B.4 OECD GUIDELINES AND DATA FLOW

For completeness we report the implementation of the OECD Guidelines and Data Flow Protocol. The labelled graph and its analysis are discussed in Chapter 7.

In Listing B.11 we report the constitutive specification of the protocol together with the specification of the initial state. The starting state, indeed, contains the conditional commitment from the data controller to the asker of the data to send the data when these are asked.

```
initial:

        cc(dc,asker,asked_data,sent_data).

constitutive:

/*
 *  Data Flow Protocol
 */

ask_data
        means
                asked_data
        if
                !asked_data.

send_data
        means
                sent_data
        if
                asked_data & !sent_data & !refuse_data.

refuse_data
        means
                refuse_data, cancel(c(dc,asker,sent_data))
        if
                asked_data & !sent_data & !refuse_data.

/*
 *  The following implements OECD Guidelines
 */


periodically_verify_accuracy
        means
                accuracy_verified
        if
                !asked_data &
                !accuracy_verified.

check_accuracy
```

```
        means
                accuracy_verified
        if
                asked_data & !accuracy_verified.

verify_purpose
        means
                purpose_verified
        if
                asked_data & !purpose_verified.

notify_owner
        means
                owner_notified
        if
                sent_data & !owner_notified.
```

Listing B.11: Constitutive specification of OECD Guidelines and Data Flow Protocol: Implementation.

Listing B.12 reports the regulative specification for OECD Guidelines.

OECD Guidelines regulative specification

```
purpose_verified ->. sent_data.
accuracy_verified ->. sent_data.
sent_data .-> owner_notified.
purpose_verified ->. refuse_data.
accuracy_verified ->. refuse_data.
```

Listing B.12: Regulative specification of OECD Guidelines and Data Flow Protocol: Implementation.

B.5 MIFID AND SALES PROTOCOL

In this section we report the implementation of the MiFID case study reported in Chapter 7. The constitutive specification for the protocol representing the MiFID and sales of financial products is reported in Listing B.13. Also in this case the initial state contains a commitment: it is the commitment from the financial promoter to the investor to make it conclude an investment.

```
initial:

        c(fp,inv,invested).

constitutive:

```

```
/*
 *  The following implements the MiFID directive
 */

interview
        means
                investor_identified , document_supplied
        if
                !investor_identified & !fi_discarded &
                !contract_abort & !contract_ended &
                !rejected_proposal.

profile
        means
                create(c(fp, inv, evaluation)), investor_classified
        if
                !investor_classified & investor_identified &
                !fi_discarded & !contract_ended &
                !contract_abort & !rejected_proposal.

classify
        means
                classified
        if
                !classified  & !fi_discarded &
                !proposed_RiskL & !contract_abort &
                !contract_ended & !rejected_proposal.

fi_evaluation
        means
                create(c(fp, inv, proposed_RiskL)),
                evaluation
        if
                classified & investor_identified &
                !evaluation & !fi_discarded &
                !contract_abort & !contract_ended &
                !rejected_proposal.

fi_discard
        means
                fi_discarded , cancel(c(fp, inv, invested)),
                cancel(c(fp, inv, proposed_RiskL))
        if
                evaluation & !fi_discarded &
                !proposed_RiskL & !contract_abort &
                !contract_ended.

order_verification
        means
                order_verified , create(c(bank,inv,executed_order))
        if
                !order_verified & order_signed.

/*
```

```
 *  The following implements the Sale
 */

propose_solution
          means
                    proposed_RiskL
          if
                    !proposed_RiskL.

reject_proposal
          means
                    rejected_proposal, release(c(fp, inv, invested))
          if
                    !accepted_proposal & proposed_RiskL &
                    !rejected_proposal.

sign_order
          means
                    create(c(inv, bank, contract_ended)),
                    accepted_proposal, order_signed
          if
                    !order_signed & proposed_RiskL &
                    !rejected_proposal.

countersign_contract
          means
                    create(c(bank, inv, executed_order)),
                    invested, contract_countersigned
          if
                    order_signed & !contract_countersigned &
                    proposed_RiskL.

send_contract
          means
                    contract_sent
          if
                    !contract_sent & contract_countersigned.


notify
          means
                    notified
          if
                    contract_countersigned & !notified
                   & !contract_ended.

end
          means
                    executed_order, contract_ended
          if
                    contract_sent & !contract_ended &
                    !contract_abort.


```

```
withdraw
        means
                contract_abort, release(c(bank, inv, executed_order)),
                cancel(c(inv, bank, contract_ended))
        if
                contract_sent & !contract_ended &
                !contract_abort.
```

Listing B.13: Constitutive specification of MiFID and sales protocol: Implementation.

MiFID regulative specification

Listing B.14 reports the regulative specification for MiFID and sales protocol.

```
notified ->. contract_ended.
contract_sent .-> notified.
created(c(fp,inv,invested)) .->
    investor_identified & document_supplied.
investor_classified ->. created(c(fp,inv,proposed_RiskL)).
evaluation ->. proposed_RiskL.
fi_discarded ./. proposed_RiskL.
order_verified ->. contract_countersigned.
```

Listing B.14: Regulative specification of MiFID and sales protocol: Implementation.

BIBLIOGRAPHY

[AÁNDVS10] H. Aldewereld, S. Álvarez-Napagao, F. Dignum, and J. Vázquez-Salceda. Making Norms Concrete. In *Proc. of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 807–814, Toronto, 2010. IFAAMAS. (Cited on page 23.)

[ACG$^+$08] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable Agent Interaction in Abductive Logic Programming: The SCIFF Framework. *ACM Transactions on Computational Logic*, 9(4), 2008. (Cited on page 17.)

[ADT$^+$04] M. Alberti, D. Daolio, P. Torroni, M. Gavanelli, E. Lamma, and P. Mello. Specification and Verification of Agent Interaction Protocols in a Logic-based System. In H. Haddad, A. Omicini, R. L. Wainwright, and L. M. Liebrock, editors, *Proc. of the 2004 ACM Symposium on Applied Computing (SAC 2004)*, pages 72–78, Nicosia, Cyprus, March 2004. ACM. (Cited on pages 17 and 94.)

[AHKV98] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating Refinement Relations. In Davide Sangiorgi and Robert de Simone, editors, *Concurrency Theory, 9th International Conference (CONCUR '98)*, volume 1466 of *LNCS*, pages 163–178, Nice, France, September 1998. Springer. (Cited on page 9.)

[AMBC$^+$95] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing. John Wiley and Sons, 1995. (Cited on page 9.)

[ANRAS06] J. L. Arcos, P. Noriega, J. A. Rodríguez-Aguilar, and C. Sierra. E4MAS Through Electronic Institutions. In D. Weyns, H. Van Dyke Parunak, and F. Michel, editors, *E4MAS*, volume 4389 of *LNCS*, pages 184–202. Springer, 2006. (Cited on pages 85 and 155.)

[ATA04] *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, New York, NY, USA, August 2004. IEEE Computer Society. (Cited on pages 200 and 201.)

[BA08] M. Ben-Ari. *Principles of the SPIN Model Checker*. Springer, 2008. (Cited on page 60.)

[BB12] M. Baldoni and C. Baroglio. Some Thoughts about Commitment Protocols. In M. Baldoni, L. Dennis, V. Mascardi, and W. Vasconcelos, editors, *Proc. of International Workshop on Declarative Agent Languages and Technologies, DALT 2012, held in conjuction with AAMAS 2012*, Valencia, Spain, June 2012. (Cited on pages 154, 155, and 156.)

[BBB$^+$10] M. Baldoni, C. Baroglio, F. Bergenti, A. Boccalatte, E. Marengo, M. Martelli, V. Mascardi, L. Padovani, V. Patti, A. Ricci, G. Rossi, and A. Santi. MERCURIO: An Interaction-oriented Framework for Designing, Verifying and Programming Multi-Agent Systems. In N. Fornara and G. Vouros, editors, *Proc. of the 3rd Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW'10), 11th International Workshop on Coordination, Organization, Institutions and Norms in Multi-Agent Systems (COIN@MALLOW 2010)*, volume 627, Domain Valpré in Lyon, France, August 30 - September 2 2010. CEUR Workshop Proceedings. (Cited on pages 85 and 156.)

[BBB$^+$11a] M. Baldoni, C. Baroglio, F. Bergenti, E. Marengo, V. Mascardi, V. Patti, A. Ricci, and A. Santi. An Interaction-oriented Agent Framework for Open Environments. In R. Pirrone and F. Sorbello, editors, *Proc. Artificial Intelligence Around Man and Beyond, 12th Congress of the Italian Association for Artificial Intelligence (AI*IA 2011)*, volume 6934 of *LNAI*, pages 68–79, Palermo, Italy, September 2011. Springer. (Cited on pages 85 and 156.)

[BBB$^+$11b] M. Baldoni, C. Baroglio, I. Brunkhorst, N. Henze, E. Marengo, and V. Patti. Constraint Modeling for Curriculum Planning and Validation. *International Journal of Interactive Learning Environments*, 19(1):83–123, 2011. (Cited on pages 17, 26, 37, and 83.)

[BBC$^+$09] M. Baldoni, C. Baroglio, A. K. Chopra, N. Desai, V. Patti, and M. P. Singh. Choice, Interoperability, and Conformance in Interaction Protocols and Service Choreographies. In K. Decker, J. Sichman, C. Sierra, and C. Castelfranchi, editors, *Proc. of the 8th International Conference* *on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 843–850, Budapest, Hungary, May 2009. IFAAMAS. (Cited on page 1.)

[BBC$^+$12] M. Baldoni, C. Baroglio, F. Capuzzimati, E. Marengo, and V. Patti. A Generalized Commitment Machine for 2CL Protocols and its Implementation. In M. Baldoni, L. Dennis, V. Mascardi, and W. Vasconcelos, editors, *Post-Proc. of International Workshop on Declarative Agent Languages and Technologies (DALT 2012), held in conjuction with AAMAS 2012*, volume LNAI, Valencia, Spain, June 2012. (Cited on pages 101 and 117.)

[BBM$^+$09] M. Baldoni, C. Baroglio, E. Marengo, V. Patti, and C. Schifanella. Joint Achievement of Services' Personal Goals. In M. Baldoni, C. Baroglio, J. Bentahar, and V. Mascardi, editors, *Proc. of the 2nd Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW'009), Agents, Web Services and Ontologies, Integrated Methodologies International Workshop (MALLOW-AWESOME'009)*, volume 494, pages 25–36, Turin, September 2009. CEUR Workshop Proceedings. (Cited on page 1.)

[BBM10a] M. Baldoni, C. Baroglio, and E. Marengo. Behavior-oriented Commitment-based Protocols. In H. Coelho, R. Studer, and M. Wooldridge, editors, *Proc. of 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 137–142, Lisbon, Portugal, August 2010. IOS Press. (Cited on pages 19, 20, and 21.)

[BBM10b] M. Baldoni, C. Baroglio, and E. Marengo. Constraints among Commitments: Regulative Specification of Interaction Protocols. In A. Artikis, J. Bentahar, A. K. Chopra, and F. Dignum, editors, *Proc. of International Workshop on Agent Communication (AC 2010), held in conjuction with AAMAS 2010*, pages 2–18, Toronto, Canada, May 2010. (Cited on pages 19 and 90.)

[BBM11a] M. Baldoni, C. Baroglio, and E. Marengo. Commitment-based Protocols with Behavioral Rules and Correctness Properties of MAS. In A. Omicini, S. Sardina, and W. Vasconcelos, editors, *Post-Proc. of the 8th International Workshop on Declarative Agent Languages and Technologies VIII (DALT 2010), Revised Selected and Invited Pa-*

pers, number 6619 in LNAI, pages 60–77. Springer, 2011. (Cited on pages 1, 19, and 99.)

[BBM+11b] M. Baldoni, C. Baroglio, E. Marengo, V. Patti, and F. Capuzzimati. Learn the Rules so you Know How to Break them Properly. In G. Fortino, A. Garro, L. Palopoli, W. Russo, and G. Spezzano, editors, *Proc. of WOA 2011: Dagli oggetti agli agenti, Progettazione ed analisi di sistemi complessi mediante modellazione e simulazione basate su agenti*, volume 741, pages 28–36, Cosenza, Italy, July 2011. CEUR Workshop Proceedings. (Cited on page 119.)

[BBM+11c] M. Baldoni, C. Baroglio, E. Marengo, V. Patti, and A. Ricci. Back to the Future: an Interaction-oriented Framework for Social Computing. In A. K. Chopra, F. Dalpiaz, and S. O. Lim, editors, *First International Workshop on Requirements Engineering for Social Computing (RESC 2011), in conjunction with the 19th IEEE International Requirements Engineering Conference*, Trento, Italy, August 29th 2011. IEEE. (Cited on page 156.)

[BBMP03a] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about Conversation Protocols in a Logic-Based Agent Language. In A. Cappelli and F. Turini, editors, *AI*IA*, volume 2829 of *LNCS*, pages 300–311, Pisa, Italy, September 2003. Springer. (Cited on page 10.)

[BBMP03b] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about Self and Others: Communicating Agents in a Model Action Logic. In C. Blundo and C. Laneve, editors, *ICTCS*, volume 2841 of *LNCS*, pages 228–241, Bertinoro, Italy, October 2003. Springer. (Cited on page 10.)

[BBMP11] M. Baldoni, C. Baroglio, E. Marengo, and V. Patti. Grafting Regulations into Business Protocols: Supporting the Analysis of Risks of Violation. In A. Antón, D. Baumer, T. Breaux, and D. Karagiannis, editors, *Fourth International Workshop on Requirements Engineering and Law (RELAW 2011), in conjunction with the 19th IEEE International Requirements Engineering Conference*, Trento, Italy, August 30th 2011. IEEE. (Cited on pages 85, 102, 119, 124, 128, 129, 131, and 140.)

[BBMP13] M. Baldoni, C. Baroglio, E. Marengo, and V. Patti. Constitutive and Regulative Specifications of Commitment

Protocols: a Decoupled Approach. *ACM Transactions on Intelligent Systems and Technology, Special Issue on Agent Communication*, 4(2), 2013. (Cited on pages 19, 20, 21, 54, and 90.)

[BBPM12] M. Baldoni, C. Baroglio, V. Patti, and E. Marengo. Supporting the Analysis of Risks of Violation in Business Protocols: the MiFID Case Study. In M. De Marco, D. Te'eni, V. Albano, and S. Za, editors, *Information Systems: Crossroads for Organization, Management, Accounting and Engineering*, pages 545–553. Springer, 2012. Best Track Paper Award. (Cited on pages 119, 124, 128, and 140.)

[BBvdT06] M. Baldoni, G. Boella, and L. W. N. van der Torre. Bridging Agent Theory and Object Orientation: Agent-Like Communication Among Objects. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, editors, *PROMAS*, volume 4411 of *LNCS*, pages 149–164, Hakodate, Japan, May 2006. Springer. (Cited on page 85.)

[BIP88] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence*, 4:349–355, 1988. (Cited on page 8.)

[BK08] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008. (Cited on pages 26, 60, and 77.)

[BMO01] B. Bauer, J. P. Müller, and J. Odell. Agent UML: A Formalism for Specifying Multiagent Software Systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):207–230, 2001. (Cited on pages 8, 10, and 87.)

[BMPG04] M. Baldoni, A. Martelli, V. Patti, and L. Giordano. Programming Rational Agents in a Modal Action Logic. *Annals of Mathematics and Artificial Intelligence*, 41(2-4):207–257, 2004. (Cited on page 10.)

[BMW09] J. Bentahar, J.-J. Ch. Meyer, and W. Wan. Model Checking Communicative Agent-based Systems. *Knowledge-Based Systems*, 22(3):142–159, 2009. (Cited on page 14.)

[BRVS11] S. Bandini, F. Rubagotti, G. Vizzari, and K. Shimura. An Agent Model of Pedestrian and Group Dynamics: Experiments on Group Cohesion. In R. Pirrone and F. Sorbello, editors, *Conference of the Italian Association for Artificial Intelligence (AI*IA 2011)*, volume 6934 of *LNCS*, pages 104–116, Palermo, Italy, September 2011. Springer. (Cited on page 1.)

[BZ09] M. Bravetti and G. Zavattaro. A Theory of Contracts for Strong Service Compliance. *Mathematical Structures in Computer Science*, 19(3):601–638, 2009. (Cited on page 9.)

[CAB+11] A. K. Chopra, A. Artikis, J. Bentahar, M. Colombetti, F. Dignum, N. Fornara, A. J. I. Jones, M. P. Singh, and P. Yolum. Research Directions in Agent Communication. *ACM Transactions on Intelligent Systems and Technology, Special Issue on Agent Communication*, pages 1 – 26, 2011. To appear. (Cited on page 8.)

[Cap10] F. Capuzzimati. Un Approccio ai Business Process basato su Commitment: il Caso di Studio della MiFID e la Realizzazione di uno Strumento per il Design. Master's thesis, Corso di Laurea in Sistemi per il Trattamento dell'Informazione, Master Thesis, Università degli Studi di Torino, 2009 – 2010. (Cited on pages 119 and 140.)

[Cas95] C. Castelfranchi. Commitments: From Individual Intentions to Groups and Organizations. In *Proc. of The First International Conference on Multiagent Systems*, pages 41–48. The MIT Press, 1995. (Cited on pages 2, 11, 12, 55, and 154.)

[CCD98] R. Conte, C. Castelfranchi, and F. Dignum. Autonomous Norm Acceptance. In J. P. Müller, M. P. Singh, and A. S. Rao, editors, *International Workshop on Agent Theories, Architectures, and Languages (ATAL 1998)*, volume 1555 of *LNCS*, pages 99–112, Paris, France, July 1998. Springer. (Cited on pages 14 and 154.)

[CCF+99] R. S. Cost, Y. Chen, T Finin, Y. K. Labrou, and Y. Peng. Modeling Agent Conversations with Colored Petri Nets. In *Working notes of the Autonomous Agents '99. Workshop on Specifying and Implementing Conversation Policies*, Seattle, Washington, May 1999. (Cited on pages 8 and 9.)

[CCGR00] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: A New Symbolic Model Checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000. (Cited on page 62.)

[CGP01] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2001. (Cited on page 96.)

[Che73] C. Cherry. Regulative Rules and Constitutive Rules. *The Philosophical Quarterly*, 23(93):301–315, 1973. (Cited on pages 23 and 83.)

[Cho09] A.K. Chopra. *Commitment Alignment: Semantics, Patterns, and Decision Procedures for Distributed Computing*. PhD thesis, North Carolina State University, Raleigh, NC, 2009. (Cited on pages 13, 15, 24, 81, 99, and 155.)

[CL90] P. R. Cohen and H. J. Levesque. Intention is Choice with Commitment. *Artificial Intelligence*, 42(2-3):213–261, 1990. (Cited on page 154.)

[CMMT09a] F. Chesani, P. Mello, M. Montali, and P. Torroni. Commitment Tracking via the Reactive Event Calculus. In C. Boutilier, editor, *IJCAI*, pages 91–96, Pasadena, California, USA, July 2009. (Cited on pages 95, 102, and 126.)

[CMMT09b] F. Chesani, P. Mello, M. Montali, and P. Torroni. Verifying A-Priori the Composition of Declarative Specified Services. In M. Baldoni, C. Baroglio, J. Bentahar, G. Boella, M. Cossentino, M. Dastani, B. Dunin-Keplicz, G. Fortino, M. P. Gleizes, J. Leite, V. Mascardi, J. A. Padget, J. Pavón, A. Polleres, A. El Fallah-Seghrouchni, P. Torroni, and R. Verbrugge, editors, *Proc. of the Second Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW 2009)*, volume 494 of *CEUR Workshop Proceedings*, Turin, Italy, September 2009. CEUR-WS.org. (Cited on pages 18, 94, and 98.)

[Com] Cognizant Company. MiFID White Paper. http://www.cognizant.com/InsightsWhitepapers/Cognizant_MiFID_Whitepaper.pdf. (Cited on page 151.)

[Con] Consob: Commissione Nazionale per le Società e la Borsa. http://www.consob.it/mainen/index.html?mode=gfx. (Cited on page 139.)

[CS03] A. K. Chopra and M. P. Singh. Nonmonotonic Commitment Machines. In Frank Dignum, editor, *Advances in Agent Communication, International Workshop on Agent Communication Languages (ACL 2003)*, volume 2922 of *LNCS*, pages 183–200, Melbourne, Australia, July 2003. Springer. (Cited on page 14.)

[CS06] A. K. Chopra and M. P. Singh. Contextualizing Commitment Protocol. In H. Nakashima, M. P. Wellman, G. Weiss, and P. Stone, editors, *Proc. of 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 1345–1352, Hakodate, Japan, May 2006. ACM. (Cited on pages 15, 82, and 84.)

[CS08] A. K. Chopra and M. P. Singh. Constitutive Interoperability. In *Proc. of the 7th international joint conference on Autonomous agents and multiagent systems*, volume 2, pages 797–804. International Foundation for Autonomous Agents and Multiagent Systems, 2008. (Cited on pages 15, 81, 98, and 99.)

[CS09a] A. K. Chopra and M. P. Singh. An Architecture for Multiagent Systems: An Approach Based on Commitments. In L. Braubach, J.-P. Briot, and J. Thangarajah, editors, *Proc. of the AAMAS Workshop on Programming Multiagent Systems (ProMAS 2009)*, volume 5919 of *LNAI*, pages 148–162, Heidelberg, 2009. Springer. (Cited on page 155.)

[CS09b] A. K. Chopra and M. P. Singh. Multiagent Commitment Alignment. In C. Sierra, C. Castelfranchi, K. S. Decker, and J. S. Sichman, editors, *AAMAS (2)*, volume 2, pages 937–944, Budapest, Hungary, May 2009. IFAAMAS. (Cited on page 99.)

[CS11] A. K. Chopra and M. P. Singh. Specifying and Applying Commitment-Based Business Patterns. In Sonenberg et al. [SSTY11], pages 475–482. (Cited on page 129.)

[CTS95] B. Cox, J. D. Tygar, and M. Sirbu. NetBill Security and Transaction Protocol. In *WOEC'95: Workshop on Electronic Commerce*, 1995. (Cited on pages 40 and 41.)

[CW05] C. Cheong and M. Winikoff. Hermes: Designing Goal-Oriented Agent Interactions. In J. P. Müller and F. Zambonelli, editors, *AOSE*, volume 3950 of *Lecture Notes in Computer Science*, pages 16–27. Springer, 2005. (Cited on page 88.)

[CW09] C. Cheong and M. Winikoff. Hermes: Designing Flexible and Robust Agent Interactions. In *Multi-Agent Systems - Semantics and Dynamics of Organizational Models*, pages 105–139, 2009. (Cited on page 88.)

[DCA$^+$07] N. V. Desai, A. K. Chopra, M. Arrott, B. Specht, and M. P. Singh. Engineering Foreign Exchange Processes via Commitment Protocols. In *IEEE Int. Conf. SCC 2007*, pages 514–521, 2007. (Cited on page 129.)

[DCS09] N. Desai, A. K. Chopra, and M. P. Singh. Amoeba: A Methodology for Modelling and Evolving Cross-Organizational Business Processes. *ACM Transactions on Software Engineering and Methodology*, 19(2), 2009. (Cited on pages 15, 129, and 156.)

[DDCP05] H.R. Dunn-Davies, R.J. Cunningham, and S. Paurobally. Propositional Statecharts for Agent Interaction Protocols. *Electronic Notes in Theoretical Computer Science*, 134(0):55 – 75, 2005. (Cited on page 8.)

[Des09] N. Desai. *Interorganizational Business Interactions: Contracts, Processes, Evolution.* PhD thesis, North Carolina State University, Raleigh, NC, 2009. (Cited on page 15.)

[DGG$^+$10] D. D'Aprile, L. Giordano, V. Gliozzi, A. Martelli, G. Pozzato, and D. Theseider Duprè. Verifying Business Process Compliance by Reasoning about Actions. In *Proc. of the 11th international conference on Computational logic in multi-agent systems*, pages 99–116. Springer, 2010. (Cited on page 125.)

[Dig04] F. Dignum, editor. *Advances in Agent Communication, International Workshop on Agent Communication Languages (ACL 2003)*, volume 2922 of *LNCS*, Melbourne, Australia, July 2004. Springer. (Cited on pages 200 and 205.)

[DYn] DYnamics in LOGic language. http://www.di.unito.it/~alice/dynamicslogic/index.html. (Cited on page 10.)

[EFSHM01] A. El Fallah-Seghrouchni, S. Haddad, and H Mazouzi. A Formal Study of Interactions in Multi-agent Systems. *International Journal of Computational and Applied Mathematics*, 8(1), 2001. (Cited on pages 8 and 9.)

[EMBD10] M. El-Menshawy, J. Bentahar, and R. Dssouli. Verifiable Semantic Model for Agent Interactions Using Social Commitments. In M. Dastani, A. El Fallah-Seghrouchni, J. Leite, and P. Torroni, editors, *LAnguages, methodologies and Development tools for multi-agent systemS (LADS 2009)*, volume 6039 of *LNCS*, pages 128–152, Torino, Italy, September 2010. Springer. (Cited on pages 9, 14, and 82.)

[EMBD11] M. El-Menshawy, J. Bentahar, and R. Dssouli. Symbolic Model Checking Commitment Protocols using Reduction. In A. Omicini, S. Sardina, and W. Vasconcelos, editors, *Declarative Agent Languages and Technologies VIII*, Toronto, Canada, 2011. Springer. (Cited on pages 1, 9, 14, 82, and 102.)

[Eme90] E. A. Emerson. *Temporal and Modal Logic*, volume B. Elsevier, Amsterdam, The Netherlands, 1990. (Cited on page 59.)

[ERRAA04] M. Esteva, B. Rosell, J. A. Rodríguez-Aguilar, and J. L. Arcos. AMELI: An Agent-Based Middleware for Electronic Institutions. In ATAL2004 [ATA04], pages 236–243. (Cited on pages 85 and 155.)

[FC03] N. Fornara and M. Colombetti. Defining Interaction Protocols using a Commitment-based Agent Communication Language. In Rosenschein et al. [RSWY03], pages 520–527. (Cited on page 87.)

[FC04a] N. Fornara and M. Colombetti. A Commitment-Based Approach To Agent Communication. *Applied Artificial Intelligence*, 18(9-10):853–866, 2004. (Cited on pages 11 and 87.)

[FC04b] N. Fornara and M. Colombetti. Protocol Specification Using a Commitment Based ACL. In Dignum [Dig04], pages 108–127. (Cited on pages 11 and 87.)

[FFMM94] T. W. Finin, R. Fritzson, D. P. McKay, and R. McEntire. KQML As An Agent Communication Language. In *CIKM*, pages 456–463, Gaithersburg, Maryland, November, December 1994. ACM. (Cited on page 8.)

[FHSB07] J. Fred Hübner, J. S. Sichman, and O. Boissier. Developing Organised Multiagent Systems Using the MOISE. *IJAOSE*, 1(3/4):370–395, 2007. (Cited on pages 85 and 155.)

[Fin94] T. Finin. Specification of the KQML Agent Communication Language. DARPA knowledge sharing initiative external interfaces working group., 1994. (Cited on page 8.)

[FK04a] R. A. Flores and R. C. Kremer. A Pragmatic Approach to Build Conversation Protocols Using Social Commitments. In ATAL2004 [ATA04], pages 1242–1243. (Cited on page 84.)

[FK04b] R. A. Flores and R. C. Kremer. A Principled Modular Approach to Construct Flexible Conversation Protocols. In A. Y. Tawfik and S. D. Goodwin, editors, *Canadian Conference on AI*, volume 3060 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2004. (Cited on page 84.)

[For03] N. Fornara. *Interaction and Communication among Autonomous Agents in Multiagent Systems*. PhD thesis, Università della Svizzera italiana, Facoltà di Scienze della Comunicazione, June 2003. (Cited on pages 11, 12, 87, and 98.)

[Fou02a] Foundation for Intelligent Physical Agents. *FIPA ACL Message Structure Specification Specification*, December 2002. (Cited on page 8.)

[Fou02b] Foundation for Intelligent Physical Agents. *FIPA Communicative Act Library Specification*, December 2002. (Cited on page 8.)

[Fou02c] Foundation for Intelligent Physical Agents. *FIPA Contract Net Interaction Protocol Specification*, December 2002. (Cited on pages xv, 40, 45, 46, and 88.)

[Fou03] Foundation for Intelligent Physical Agents. *FIPA Modelling: Interaction Diagrams*, July 2003. (Cited on pages 8, 10, and 87.)

[GAD07] D. Grossi, H. Aldewereld, and F. Dignum. Ubi lex, ibi poena: Designing Norm Enforcement in E-Institutions. In V. Dignum, N. Fornara, P. Noriega, G. Boella, O. Boissier, E. Matson, and J. Ýazquez-Salceda, editors, *Proc. of Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems II (COIN 2006)*, volume 4386 of *LNCS*, pages 101–114, Hakodate, Japan, 2007. Springer. (Cited on page 85.)

[Gib07] G. Gibilaro. Sentence of the Corte di Cassazione. Sentenza, Sez. SS.UU., 19/12/2007, n. 26724 e 26725. Intermediazione finanziaria, nullità del contratto e risarcimento del danno, 2007. (Cited on page 140.)

[GMS07] L. Giordano, A. Martelli, and C. Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic. *Journal of Applied Logic*, 5(2):214–234, 2007. (Cited on pages 14, 40, and 82.)

[Gov10] G. Governatori. Law, Logic and Business Processes. In *Requirements Engineering and Law (RELAW), 2010 Third International Workshop on*, pages 1–10. IEEE, 2010. (Cited on page 125.)

[Gro07] D. Grossi. *Designing Invisible Handcuffs, Formal Investigations in Institutions and Organizations for Multi-agent Systems*. PhD thesis, University of Utrecht, 2007. (Cited on page 155.)

[HBB10] J. F. Hübner, O. Boissier, and R. H. Bordini. From Organisation Specification to Normative Programming in Multi-Agent Organisations. In *Computational Logic in Multi-Agent Systems, 11th International Workshop (CLIMA XI)*, volume 6245 of *LNCS*, pages 117–134. Springer, 2010. (Cited on page 155.)

[HK03] M. P. Huget and J. L. Koning. Interaction Protocol Engineering in Multiagent Systems. Multiagent System Technologies (MATES 2003), Tutorial, 2003. (Cited on page 8.)

[Hol03] G. J. Holzmann. *The SPIN Model Checker, Primer and Reference Manual*. Addison-Wesley, 2003. (Cited on pages 62 and 68.)

[HR04] M. Huth and M. Ryan. *Logic in Computer Science. Modelling and Reasoning about Systems*. Cambridge University Press, 2nd edition, 2004. (Cited on page 60.)

[Jen93] N. R. Jennings. Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems. *The Knowledge Engineering Review*, 8:223–250, 1993. (Cited on page 154.)

[JS94] A. J. I. Jones and M. Sergot. *On the Characterization of Law and Computer Systems: the Normative Systems Perspective*, pages 275–307. John Wiley & Sons, Inc., New York, NY, USA, 1994. (Cited on pages 85, 101, and 124.)

[JS96] A. J. I. Jones and M. J. Sergot. A Formal Characterisation of Institutionalised Power. *Logic Journal of the IGPL*, 4(3):429–445, 1996. (Cited on page 155.)

[Kan04] C. W. Kann. *Creating Components*. Auerbach, 2004. (Cited on page 82.)

[KFD98] J.-L. Koning, G. Francois, and Y. Demazeau. Formalization and Pre-Validation for Interaction Protocols in a Multi Agent Systems. In *European Conference on Artificial Intelligence (ECAI 1998)*, pages 298–307, 1998. (Cited on pages 8 and 9.)

[KIO95] K. Kuwabara, T. Ishida, and N. Osato. AgenTalk: Coordination Protocol Description for Multiagent Systems. In *Proc. of the Seventh International Conference on Tools with Artificial Intelligence (TAI '95)*, pages 460–465, 1995. (Cited on pages 8 and 9.)

[KY09] Ö. Kafali and P. Yolum. Detecting Exceptions in Commitment Protocols: Discovering Hidden States. In *Proc. of the Second Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW 2009)*, volume 494 of *CEUR Workshop Proceedings*, Turin, Italy, 2009. CEUR-WS.org. (Cited on pages 14 and 84.)

[Lam78] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 1978. (Cited on page 32.)

[Lee99] R. M. Lee. Distributed Electronic Trade Scenarios: Representation, Design, Prototyping. *International Journal of Electronic Commerce*, 3(2), 1999. (Cited on pages 8 and 9.)

[LG95] V. R. Lesser and L. Gasser, editors. *Proc. of the First International Conference on Multiagent Systems*, San Francisco, California, USA, June 1995. The MIT Press. (Cited on pages 204 and 207.)

[MBB11a] E. Marengo, M. Baldoni, and C. Baroglio. Extend Commitment Protocols with Temporal Regulations: Why and How. In C. Damasio, A. Preece, and U. Straccia, editors, *Proc. of 5th International Symposium on Rules, Doctoral Consortium (RuleML 2011)*, CoRR abs/1107.2086, pages 137–142, Barcellona, Spain, July 2011. CoRR, Cornell University Library. (Cited on page 21.)

[MBB11b] E. Marengo, M. Baldoni, and C. Baroglio. On Temporal Regulations and Commitment Protocols. In T. Walsh, editor, *Proc. of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 2824–2825, Barcellona, Spain, July 2011. AAAI Press/International Joint Conferences on Artificial Intelligence. (Cited on page 21.)

[MBB+11c] E. Marengo, M. Baldoni, C. Baroglio, A. K. Chopra, V. Patti, and M. P. Singh. Commitments with Regulations: Reasoning about Safety and Control in REGULA. In Sonenberg et al. [SSTY11], pages 467–474. (Cited on pages 12, 14, 92, and 154.)

[MBF95] M. Mihai Barbuceanu and M. S. Fox. COOL: A Language for Describing Coordination in Multi Agent Systems. In Lesser and Gasser [LG95], pages 17–24. (Cited on pages 8 and 9.)

[Mifa] Directive 2004/39/EC of the European Parliament and of the Council of 21 April 2004 on markets in financial instruments. (Cited on pages 3 and 140.)

[Mifb] Markets in Financial Instruments Directive. Directive 2004/39/EC. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CONSLEG:2004L0039:20070921:EN:PDF. (Cited on page 128.)

[MM08] T. Miller and J. McGinnis. Amongst First-Class Protocols. In A. Artikis, G. M. P. O'Hare, K. Stathis, and G. A. Vouros, editors, *Proc. of the 8th International Workshop on Engineering Societies in the Agents World VIII (ESAW 2007)*, volume 4995 of *LNCS*, pages 208–223, Athens, Greece, October 2008. Springer. (Cited on page 35.)

[Mon09] M. Montali. *Specification and Verification of Declarative Open Interaction Models*. PhD thesis, Electronics, Computer Science and Telecommunications Engineering, University of Bologna, 2009. (Cited on pages 9, 18, 26, 82, 83, 91, 94, and 98.)

[MPvdA+10] M. Montali, M. Pesic, W.M. P. van der Aalst, F. Chesani, P. Mello, and S. Storari. Declarative Specification and Verification of Service Choreographies. *ACM Transactions on the Web*, 4(1):3–62, 2010. (Cited on pages 18, 94, and 98.)

[MS05] A. U. Mallya and M. P. Singh. Modeling Exceptions via Commitment Protocols. In F. Dignum, V. Dignum,

S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, editors, *AAMAS*, pages 122–129, Utrecht, The Netherlands, July 2005. ACM. (Cited on pages 12, 13, and 102.)

[MS06] A. U. Mallya and M. P. Singh. Introducing Preferences into Commitment Protocols. In F. Dignum, R. M. van Eijk, and R. A. Flores, editors, *Agent Communication II, International Workshops on Agent Communication (AC 2005 and AC 2006)*, volume 3859 of *LNCS*, pages 136–149, Utrecht, Netherlands, 2006. Springer. (Cited on pages 15, 89, 91, and 98.)

[MTC$^+$10] M. Montali, P. Torroni, F. Chesani, P. Mello, M. Alberti, and E. Lamma. Abductive Logic Programming as an Effective Technology for the Static Verification of Declarative Business Processes. *Fundamenta Informaticae*, 102(3-4):325–361, 2010. (Cited on pages 18, 94, 95, and 125.)

[MVB$^+$04] C. Masolo, L. Vieu, E. Bottazzi, C. Catenacci, R. Ferrario, A. Gangemi, and N. Guarino. Social Roles and their Descriptions. In D. Dubois, C. A. Welty, and M.-A. Williams, editors, *Principles of Knowledge Representation and Reasoning: Proc. of the Ninth International Conference (KR2004)*, pages 267–277, Whistler, Canada, June 2004. AAAI Press. (Cited on page 85.)

[MYS03] Ashok U. Mallya, Pinar Yolum, and Munindar P. Singh. Resolving Commitments among Autonomous Agents. In Dignum [Dig04], pages 166–182. (Cited on pages 12 and 14.)

[Org80] Organisation for Economic Co-operation and Development. OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data. Available online, 1980. http://www.oecd.org/. (Cited on pages 2, 128, and 130.)

[ORV08] A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems (AAMAS 2008)*, 17(3):432–456, 2008. (Cited on page 156.)

[OVDPB00] J. Odell, H. Van Dyke Parunak, and B. Bauer. Representing Agent Interaction Protocols in UML. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering, First International Workshop (AOSE 2000)*, volume 1957 of *LNCS*, pages 121–140, Limerick, Ireland, June 2000. Springer. (Cited on page 8.)

[PBvdA10] M. Pesic, D. Bosnacki, and W. M. P. van der Aalst. Enacting Declarative Languages using LTL: Avoiding Errors and Improving Performance. In J. van de Pol and M. Weber, editors, *Proc. of the 17th International SPIN Workshop on Model Checking of Software (SPIN 2010)*, volume 6349 of *LNCS*, pages 146–161, Enschede, The Netherlands, September 2010. Springer. (Cited on page 96.)

[Pes08] M. Pesic. *Constraint-Based Workflow Management Systems: Shifting Control to Users*. PhD thesis, Eindhoven University of Technology, 2008. (Cited on page 16.)

[Pir11] V. Piro. iDot2CL - Interfaccia Grafica per l'analisi di files 2CL. Master's thesis, Bachelor Degree in Computer Science, Università degli Studi di Torino, 2010 – 2011. (Cited on page 123.)

[PKSA06] J. Pitt, L. Kamara, M. J. Sergot, and A. Artikis. Voting in Multi-Agent Systems. *The Computer Journal*, 49(2):156–170, 2006. (Cited on page 21.)

[PSvdA07] M. Pesic, H. Schonenberg, and W.M.P. van der Aalst. DECLARE: Full Support for Loosely-Structured Processes. In *Proc. of the 11th IEEE International Enterprise Distributed Object Computing Conference*. IEEE Computer Society, Washington, DC, USA, 2007. (Cited on page 16.)

[PvdA06] M. Pesic and W. M. P. van der Aalst. A Declarative Approach for Flexible Business Processes Management. In J. Eder and S. Dustdar, editors, *Proc. of Business Process Management International Workshops (BPM 2006)*, volume 4103 of *LNCS*, pages 169–180, Vienna, Austria, September 2006. Springer. (Cited on pages 16, 18, 37, and 83.)

[Rai06] F. Raimondi. *Model Checking Multi-Agent Systems*. PhD thesis, University College London, 2006. (Cited on page 62.)

[RBW07] G. Regev, I. Bider, and A. Wegmann. Defining Business Process Flexibility with the help of Invariants. *Software Process: Improvement and Practice*, 12(1):65–79, 2007. (Cited on page 82.)

[REHB00] H. M. III Robert, W. J. Evans, D. H. Honemann, and T. J. Balch. *Robert's Rules of Order Newly Revised, 10th Edition*. Da Capo Press, Cambridge, Massachusetts, 2000. (Cited on pages 2 and 21.)

[RG95] A. S. Rao and M. P. Georgeff. BDI Agents: From Theory to Practice. In Lesser and Gasser [LG95], pages 312–319. (Cited on page 8.)

[RN03] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Series in Artificial Intelligence. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003. (Cited on page 1.)

[RPV11] A. Ricci, M. Piunti, and M. Viroli. Environment Programming in MAS – An Artifact-Based Perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, 2011. (Cited on page 156.)

[RR02] S. K. Rajamani and J. Rehof. Conformance Checking for Models of Asynchronous Message Passing Software. In Ed Brinksma and Kim G. Larsen, editors, *Computer Aided Verification, 14th International Conference (CAV 2002)*, volume 2404 of *LNCS*, pages 166–179, Copenhagen, Denmark, July 2002. Springer. (Cited on page 9.)

[RSWY03] J. S. Rosenschein, T. Sandholm, M. Wooldridge, and M. Yokoo, editors. *Proc. of the Second International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2003)*, Melbourne, Australia, July 2003. ACM. (Cited on pages 200 and 208.)

[SBH05] C. Sibertin-Blanc and N. Hameurlain. Participation Components for Holding Roles in Multiagent Systems Protocols. In *Engineering Societies in the Agents World*, volume 3451 of *LNCS*, pages 60–73, August 2005. (Cited on page 155.)

[SC09] M. P. Singh and A. K. Chopra. Correctness Properties for Multiagent Systems. In *Declarative Agent Languages and Technologies. DALT 2009*, 2009. (Cited on page 20.)

[SD78] R. Smith and R. Davis. Distributed Problem Solving: the Contract-Net Approach. In *Conference of Canadian Society for Computational Studies of Intelligence*, pages 217 – 236, 1978. (Cited on page 40.)

[Sea69] J. R. Searle. *Speech Acts. An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, England, 1969. (Cited on pages 22 and 83.)

[Sea95] J. R. Searle. *The construction of social reality*. Free Press, New York, 1995. (Cited on pages 22 and 23.)

[Ser08] M. Sergot. Action and agency in norm-governed multi-agent systems. In *Engineering Societies in the Agents World VIII*, pages 1–54. Springer, 2008. (Cited on page 126.)

[Sin91] M. P. Singh. Social and Psychological Commitments in Multiagent Systems. In *AAAI fall symposium on knowledge and action at social and organizational levels*, pages 104–106, 1991. (Cited on pages 12 and 154.)

[Sin92] M. P. Singh. A Critical Examination of Use Cohen-Levesque Theory of Intentions. In *ECAI*, pages 364–368, 1992. (Cited on pages 12 and 154.)

[Sin99] M. P. Singh. An Ontology for Commitments in Multiagent Systems. *Artifficial Intelligence and Law*, 7(1):97–113, 1999. (Cited on pages 2, 11, 12, 13, 15, 24, 55, and 154.)

[Sin00] M. P. Singh. A Social Semantics for Agent Communication Languages. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, volume 1916 of *LNCS*, pages 31–45, Heidelberg, 2000. Springer. (Cited on page 10.)

[Sin03a] M. P. Singh. Agent Communication Languages: Rethinking the Principles. In M.-P. Huget, editor, *Communication in Multiagent Systems*, volume 2650 of *LNCS*, pages 37–50. Springer, 2003. (Cited on pages 10 and 11.)

[Sin03b] M. P. Singh. Distributed Enactment of Multiagent Workflows: Temporal Logic for Web Service Composition. In Rosenschein et al. [RSWY03], pages 907–914. (Cited on pages 15, 89, 91, and 98.)

[Sin07] M. P. Singh. Formalizing Communication Protocols for Multiagent Systems. In M. M. Veloso, editor, *IJCAI*, pages 1519–1524, Hyderabad, India, January 2007. (Cited on pages 4, 6, 14, 35, 40, 59, 71, 72, 73, 75, 76, 77, 78, 79, 108, 117, 118, and 153.)

[Sin08] M. P. Singh. Semantical Considerations on Dialectical and Practical Commitments. In D. Fox and C. P. Gomes, editors, *AAAI*, pages 176–181, Chicago, Illinois, USA, July 2008. AAAI Press. (Cited on page 12.)

[SMR+08] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. M. P. van der Aalst. Towards a Taxonomy of Process Flexibility. In Z. Bellahsene, C. Woo, E. Hunt, X. Franch, and R. Coletta, editors, *CAiSE Forum*, volume 344 of *CEUR Workshop Proceedings*, pages 81–84, Montpellier, France, June 2008. CEUR-WS.org. (Cited on page 82.)

[SSTY11] L. Sonenberg, P. Stone, K. Tumer, and P. Yolum, editors. *10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, volume 1–3, Taipei, Taiwan, May 2011. IFAAMAS. (Cited on pages 198 and 204.)

[ST95] M. A. Sirbu and J. D. Tygar. NetBill: An Internet Commerce System Optimized for Network Delivered Services. In *IEEE Computer Society International Conference: Technologies for the Information Superhighway (COMPCON 1995)*, pages 20–25, 1995. (Cited on pages 40 and 41.)

[TCMM09] P. Torroni, F. Chesani, P. Mello, and M. Montali. Social Commitments in Time: Satisfied or Compensated. In M. Baldoni, J. Bentahar, M. B. van Riemsdijk, and J. Lloyd, editors, *DALT*, volume 5948 of *LNCS*, pages 228–243, Budapest, Hungary, May 2009. Springer. (Cited on page 95.)

[TCY+09] P. Torroni, F. Chesani, P. Yolum, M. Gavanelli, M. P. Singh, E. Lamma, M. Alberti, and P. Mello. Modelling Interactions via Commitments and Expectations. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pages 263–284, 2009. (Cited on pages 10, 11, 17, 18, 85, and 94.)

[TS10] P. R. Telang and M. P. Singh. Abstracting Business Modeling Patterns from RosettaNet. In *Service-Oriented Computing: Agents, Semantics, and Engineering*, 2010. (Cited on pages 127 and 129.)

[TT98] Y.-H. Tan and W. Thoen. Modeling Directed Obligations and Permissions in Trade Contracts. In *Annual Hawaii International Conference on System Sciences (HICSS 1998)*, pages 166–175, 1998. (Cited on page 15.)

[tup] http://www.alice.unibo.it/xwiki/bin/view/Tuprolog/. (Cited on page 101.)

[vdAPS09] W. van der Aalst, M. Pesic, and H. Schonenberg. Declarative Workflows: Balancing Between Flexibility and Support. *Computer Science - Research and Development*, 23:99–113, 2009. 10.1007/s00450-009-0057-9. (Cited on page 16.)

[vRW07] M. B. van Riemsdijk and M. Wirsing. Using Goals for Flexible Service Orchestration. In *Service-Oriented Computing: Agents, Semantics, and Engineering: SOCASE 2007*, volume 4504 of *LNCS*, pages 31–48, 2007. (Cited on page 1.)

[VS99] M. Venkatraman and M. P. Singh. Verifying Compliance with Commitment Protocols. *Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, 1999. (Cited on pages 2, 11, 13, 23, and 85.)

[Wei99] G. Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999. (Cited on pages 7, 8, and 14.)

[Win07] M. Winikoff. Implementing Commitment-Based Interactions. In E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, editors, *AAMAS*, page 128. IFAAMAS, 2007. (Cited on pages 84 and 103.)

[WLH05] M. Winikoff, W. Liu, and J. Harland. Enhancing Commitment Machines. In *DALT 2004*, volume 3476 of *LNCS*, pages 198–220. Springer, 2005. (Cited on pages 14, 40, 41, 84, 85, 98, 101, 102, 118, and 159.)

[Woo02] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, March 2002. (Cited on page 8.)

[WOO07] D. Weyns, A. Omicini, and J. Odell. Environment as a First Class Abstraction in Multiagent Systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, 2007. (Cited on page 156.)

[YS01] P. Yolum and M. P. Singh. Designing and Executing Protocols Using the Event Calculus. In *Proc. of the fifth international conference on Autonomous agents*, AGENTS '01, pages 27–28, New York, NY, USA, 2001. ACM. (Cited on pages 15, 24, and 82.)

[YS02a] P. Yolum and M. P. Singh. Commitment Machines. In J.-J. Ch. Meyer and M. Tambe, editors, *Proc. of the 8th International Workshop on Intelligent Agents VIII (ATAL 2001)*, volume 2333 of *LNCS*, pages 235–247, Seattle, WA, USA, August 2002. Springer. (Cited on pages 9, 14, 20, 40, 41, 81, and 82.)

[YS02b] P. Yolum and M. P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. In *AAMAS*, pages 527–534, Bologna, Italy, July 2002. ACM. (Cited on page 41.)

[YS04] P. Yolum and M. P. Singh. Reasoning about Commitments in the Event Calculus: An Approach for Specifying and Executing Protocols. *Annals of Mathematics and Artificial Intelligence*, 42(1-3):227–253, 2004. (Cited on page 14.)

[ZJW03] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing Multiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, 2003. (Cited on page 85.)