A Causal MultiAgent System Approach for Automating Processes in Intelligent Organizations

BY

HECTOR GIBRAN CEBALLOS CANCINO

MONTERREY, N.L. DECEMBER, 2010
The committee members hereby recommend the dissertation presented by Héctor Gibrán Ceballos Cancino to be accepted as a partial fulfillment of requirements to be admitted to the Degree of Doctor of Philosophy in Information Technologies and Communications, Major in Intelligent Systems.

Committee members:

Dr. Francisco J. Cantú Ortíz
Tecnológico de Monterrey
Advisor

Dr. Ramón F. Brenn Piñero
Tecnológico de Monterrey

Dr. Antonio Sánchez Aguilar
Texas Christian University

Dr. Pablo Noriega
III-CSIC

Dr. Leonardo Garrido
Tecnológico de Monterrey

Dr. José Luis Gordillo Moscoso
Director of the Doctoral Program in Information Technologies and Communications
Division of Mechatronics and Information Technologies Graduate Program
Declaration

I hereby declare that I composed this dissertation entirely myself and that it describes my own research.

Héctor Gibrán Ceballos Cancino
Monterrey, N.L., México
December 2010
Abstract

The current competitive environment motivated Knowledge Management (KM) theorists to propose the notion of Organizational Intelligence (OI) for enabling a rapid response of large organizations to changing conditions. KM practitioners consider that OI resides in both processes and members of the organization, and recommend implementing learning mechanisms and empowering participants with knowledge and decision making for improving organization competitiveness.

In that sense, have been provided some theoretical definitions and practical approaches (e.g. Electronic Institutions and Autonomic Computing), as well as commercial platforms (e.g. Whitestein Technologies), that implement OI to a certain extent. Some of these approaches have already taken advantage of tools and formalisms developed in Artificial Intelligence (e.g. Knowledge Representation, Data Mining, and Intelligent Agents).

In this research, I propose the use of Aristotelian Causality for modeling organizations, as well as its members, as intelligent entities through the Causal Artificial Intelligence Design (CAID) theory, and present the Causal Multi-Agent System (CMAS) framework for automating organizational processes. Bayesian Causal Networks are extended to Semantic Causal Networks (SCN) for providing an explicit representation of the goals, participants, resources and knowledge involved in these processes. The CAID principles and the SCN formalism are used for providing a probabilistic extension of the goal-driven Belief-Desire-Intention agent architecture, called Causal Agent. Lastly, the capabilities of this framework are demonstrated through the specification and automation of an information auditing process.
Acknowledgements

I would first like to express my deepest gratitude to my advisor for this intellectual and spiritual journey.

I would also thank to all revisors by their insightful comments and recommendations. In particular to Dr. Sánchez and Dr. Noriega by their hospitality and openness in discussion.

I would also thank to colleagues like Alejandro Meade and Eduardo Novelo by the fruitful discussion. In particular to Magdalena Ortíz by unveiling the mystery of Description Logics and to Carlos Aguilar for sharing with me his experience in industrial applications with agents and RDF stores.

Later but not least, I would like to thank to my wife Yolanda by her understanding and support. Furthermore, by the long talks through which I discovered the human dimension of my work.
Dedication

To my parents, Héctor and Lucy
Thank you for your confidence and for directing me in the right path.

To Yolanda and Victoria
Thank you for your support, patience and encouragement. You were my main motivation for pushing through this work.

To my little sister
Thank you for your advice and example.
## Contents

Committee Declaration iii

Declaration v

List of Figures xvii

List of Tables xx

1 Introduction 1

1.1 Organizational Intelligence and Artificial Intelligence . . . . . . . . . . . . 1
1.2 Causality in the Science of Design . . . . . . . . . . . . . . . . . . . . . . 3
1.3 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
1.4 Problem Statement and Context . . . . . . . . . . . . . . . . . . . . . . . 4
1.5 Hypothesis and Research Questions . . . . . . . . . . . . . . . . . . . . . 5
1.6 Solution Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
1.7 Main Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
1.8 Thesis Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
1.9 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

2 Background and Related Work 9

2.1 Organizational Intelligence in Knowledge Management . . . . . . . . . . . 9
2.2 Autonomic Computing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
2.3 Artificial Intelligence Design . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.4 Philosophical Foundations . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
6.3.2 Learning .................................................. 130
6.3.3 Agent instantiation ....................................... 131
6.4 Agent Types .................................................. 131
   6.4.1 The Specialized Agent ................................. 131
   6.4.2 The User Agent ......................................... 132
   6.4.3 The Process Manager Agent ........................... 132
6.5 Summary ..................................................... 133

7 The Causal Multi-Agent System ................................ 135
   7.1 The CMAS Architecture .................................. 135
      7.1.1 The Organizational Ontology ...................... 135
      7.1.2 The Knowledge Management System ................ 136
      7.1.3 The Agent repository ................................ 137
      7.1.4 The Knowledge-Information Interpreter ............ 137
      7.1.5 Organizational Assets ............................... 138
      7.1.6 System and Process Administrators ................ 138
      7.1.7 System users ......................................... 138
      7.1.8 The MAS platform .................................... 139
   7.2 The CMAS Methodology ................................... 140
      7.2.1 Defining Organizational Goals ...................... 141
      7.2.2 Specifying Organizational Processes ............... 142
      7.2.3 Introducing Specialized Agents .................... 144
      7.2.4 Introducing Human Supervision ..................... 145
      7.2.5 Incorporating Organizational Metrics ............... 146
      7.2.6 Introduce Process Managers and Monitors .......... 147
      7.2.7 Generating Process Views ........................... 148
      7.2.8 Identifying participant strategies ................ 150
      7.2.9 Generating the Communication Protocol ............. 150
      7.2.10 Building the Organizational Ontology .............. 152
      7.2.11 Validating the specification ....................... 155
7.2.12 Extending Agent Classes ........................................... 156
7.2.13 Developing Software Agents .................................... 157
7.2.14 Designing Organizational Protocols ............................... 158
7.3 CMAS Tools ............................................................. 158
  7.3.1 CMAS Log .......................................................... 159
7.4 CMAS Operations ...................................................... 159
  7.4.1 Starting up .......................................................... 159
  7.4.2 Following up ........................................................ 159
  7.4.3 Tuning ............................................................... 160
7.5 Summary ............................................................... 160

8 A Case Study on AIA .................................................... 163
  8.1 A Knowledge-based information system for managing research programs
       and value creation ..................................................... 164
  8.2 Autonomic Information Auditing .................................... 166
  8.3 Causal modeling of AIA .............................................. 167
    8.3.1 Organizational goals and metrics ............................. 167
    8.3.2 The AuditNewPub process .................................... 168
    8.3.3 The AIA Organizational Ontology ............................ 176
    8.3.4 AIA Organizational Protocols ................................ 182
  8.4 Implementing AIA through Electronic Institutions ................ 182
    8.4.1 Expressing the AuditNewPub process in EIs ................ 184
    8.4.2 Implementing Organizational Protocols ...................... 186
    8.4.3 AIA agents’ implementation in EIs ........................... 188
    8.4.4 Simulating AIA through EIs .................................. 189
  8.5 Implementing the AIA through Causal Agents ....................... 189
  8.6 Experiments .......................................................... 191
    8.6.1 Set up ............................................................ 191
    8.6.2 Learning configuration ........................................ 192
    8.6.3 Experiment 1. Self-configuration of process specification .. 194
8.6.4 Experiment 2. Optimization on conflicting goals ........................................ 195
8.6.5 Experiment 3. Self-protection of inaccurate expert knowledge .................. 196
8.7 Results ........................................................................................................... 197
  8.7.1 Results on Self-configuration of process specification ......................... 197
  8.7.2 Results for optimization on conflicting goals ........................................ 199
  8.7.3 Results on self-protection of inaccurate expert knowledge .................. 201
  8.7.4 Discussion on results .............................................................................. 202
8.8 CMAS applications and limitations ............................................................... 203
  8.8.1 CMAS in other domains .......................................................................... 203
  8.8.2 Scalability .............................................................................................. 204
  8.8.3 Limitations of the current CMAS implementation ............................... 205
8.9 Summary ....................................................................................................... 206

9 Conclusions ....................................................................................................... 209
  9.1 Organizational Intelligence through Causality ........................................... 210
  9.2 Contributions of Aristotelian Causality .................................................... 212
  9.3 Ontological modeling of domains ............................................................... 215
  9.4 A semantic and probabilistic plan representation ...................................... 216
  9.5 Causal modeling of intelligent entities ...................................................... 218
  9.6 A causal goal-driven BDI agent architecture ............................................ 219
  9.7 A methodology for automating organizational processes ....................... 220
  9.8 Applications on autonomic processes ....................................................... 221
  9.9 Future Work .............................................................................................. 222

Bibliography ........................................................................................................ 225
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Causal Model Graphical Representation (Causal Diagram)</td>
<td>33</td>
</tr>
<tr>
<td>2.2</td>
<td>Bayesian Network for a Diagnostic Task</td>
<td>39</td>
</tr>
<tr>
<td>2.3</td>
<td>MEBN Fragments for the Diagnosis Task</td>
<td>40</td>
</tr>
<tr>
<td>2.4</td>
<td>Schematic diagram of a generic BDI architecture [91]</td>
<td>42</td>
</tr>
<tr>
<td>2.5</td>
<td>Prometheus design phases [68]</td>
<td>47</td>
</tr>
<tr>
<td>2.6</td>
<td>AGR Methodological Model [28]</td>
<td>52</td>
</tr>
<tr>
<td>3.1</td>
<td>A causal taxonomy of entities</td>
<td>58</td>
</tr>
<tr>
<td>4.1</td>
<td>Graphical representation of a SCN</td>
<td>84</td>
</tr>
<tr>
<td>4.2</td>
<td>Types of workflows represented through SCNs</td>
<td>85</td>
</tr>
<tr>
<td>5.1</td>
<td>Missing conditions for the execution of an action</td>
<td>112</td>
</tr>
<tr>
<td>6.1</td>
<td>Causal Agent architecture</td>
<td>119</td>
</tr>
<tr>
<td>6.2</td>
<td>Goal and plan life cycles</td>
<td>121</td>
</tr>
<tr>
<td>7.1</td>
<td>The Causal MAS architecture</td>
<td>136</td>
</tr>
<tr>
<td>7.2</td>
<td>Ontological repositories</td>
<td>137</td>
</tr>
<tr>
<td>7.3</td>
<td>Agent Layers in a CMAS</td>
<td>139</td>
</tr>
<tr>
<td>7.4</td>
<td>Goal template</td>
<td>141</td>
</tr>
<tr>
<td>7.5</td>
<td>Process template</td>
<td>142</td>
</tr>
<tr>
<td>7.6</td>
<td>Nodes introduced by the intervention of a software agent</td>
<td>145</td>
</tr>
<tr>
<td>7.7</td>
<td>Nodes introduced by human supervision</td>
<td>146</td>
</tr>
<tr>
<td>7.8</td>
<td>Template for process organizational metrics</td>
<td>147</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>7.9</td>
<td>Template for general organizational metrics.</td>
<td>147</td>
</tr>
<tr>
<td>7.10</td>
<td>Template for process PMA and PMO.</td>
<td>148</td>
</tr>
<tr>
<td>8.1</td>
<td>Metadata common to all publication categories.</td>
<td>164</td>
</tr>
<tr>
<td>8.2</td>
<td>Publication status evolution.</td>
<td>165</td>
</tr>
<tr>
<td>8.3</td>
<td>Causal diagram of AIA.</td>
<td>166</td>
</tr>
<tr>
<td>8.4</td>
<td>The original auditing process.</td>
<td>169</td>
</tr>
<tr>
<td>8.5</td>
<td>The automated auditing process.</td>
<td>171</td>
</tr>
<tr>
<td>8.6</td>
<td>Simplified automated auditing process.</td>
<td>172</td>
</tr>
<tr>
<td>8.7</td>
<td>The AuditNewPub augmented process.</td>
<td>173</td>
</tr>
<tr>
<td>8.8</td>
<td><em>CorrectorAg</em> process view of AuditNewPub.</td>
<td>174</td>
</tr>
<tr>
<td>8.9</td>
<td>Strategies identified in AuditNewPub.</td>
<td>175</td>
</tr>
<tr>
<td>8.10</td>
<td>Strategies identified in the partial view of <em>CorrectorAg</em>.</td>
<td>176</td>
</tr>
<tr>
<td>8.11</td>
<td>The AuditNewPub OCDD.</td>
<td>177</td>
</tr>
<tr>
<td>8.12</td>
<td>The UML sequence diagram for AuditNewPub.</td>
<td>178</td>
</tr>
<tr>
<td>8.13</td>
<td>The Entity hierarchy of classes.</td>
<td>180</td>
</tr>
<tr>
<td>8.14</td>
<td>Causal diagrams for <em>AgRequest</em>.</td>
<td>183</td>
</tr>
<tr>
<td>8.15</td>
<td>Communication protocol for <em>AgRequest</em>.</td>
<td>183</td>
</tr>
<tr>
<td>8.16</td>
<td>Performative structure of AuditNewPub.</td>
<td>184</td>
</tr>
<tr>
<td>8.17</td>
<td><em>NewPub</em> scene.</td>
<td>185</td>
</tr>
<tr>
<td>8.18</td>
<td><em>Correction</em> scene.</td>
<td>186</td>
</tr>
<tr>
<td>8.19</td>
<td>Communication protocol for <em>AgRequest</em>.</td>
<td>187</td>
</tr>
<tr>
<td>8.20</td>
<td>Performative structure for organizational protocols.</td>
<td>188</td>
</tr>
<tr>
<td>8.21</td>
<td>Comparison of score functions in the AuditNewPub SCN.</td>
<td>193</td>
</tr>
<tr>
<td>8.22</td>
<td>Simulated behaviors for (a) Auditors and (b) Authors in experiment 1.</td>
<td>195</td>
</tr>
<tr>
<td>8.23</td>
<td>Simulated behaviors for (a) Auditors and (b) Authors in experiment 2.</td>
<td>196</td>
</tr>
<tr>
<td>8.24</td>
<td>Simulated behaviors for (a) Auditors and (b) Authors in experiment 3.</td>
<td>197</td>
</tr>
<tr>
<td>8.25</td>
<td>Strategy selection and overall performance in experiment 1.</td>
<td>198</td>
</tr>
<tr>
<td>8.26</td>
<td>Decision making for the PubCarrierAg.</td>
<td>199</td>
</tr>
<tr>
<td>8.27</td>
<td>Decision making for the Auditor.</td>
<td>199</td>
</tr>
</tbody>
</table>
8.28 Strategy selection and overall performance in experiment 2. . . . . . . . . 201
8.29 Strategy selection by correction rule in experiment 3. . . . . . . . . . . . 202
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Concept constructors</td>
<td>20</td>
</tr>
<tr>
<td>2.2</td>
<td>Role constructors</td>
<td>21</td>
</tr>
<tr>
<td>2.3</td>
<td>Terminological and assertional axioms</td>
<td>21</td>
</tr>
<tr>
<td>2.4</td>
<td>CCalc causal laws abbreviations (axioms).</td>
<td>30</td>
</tr>
<tr>
<td>3.1</td>
<td>Design and creation interpretations for DL concept constructors</td>
<td>60</td>
</tr>
<tr>
<td>5.1</td>
<td>Main causes representation in an intelligent entity definition.</td>
<td>96</td>
</tr>
<tr>
<td>8.1</td>
<td>AIA process goal.</td>
<td>167</td>
</tr>
<tr>
<td>8.2</td>
<td>Organizational metric: Maximize repository confidence.</td>
<td>168</td>
</tr>
<tr>
<td>8.3</td>
<td>Organizational Metric: Minimize unnecessary human revision.</td>
<td>168</td>
</tr>
<tr>
<td>8.4</td>
<td>Organizational process: Audit new publications.</td>
<td>169</td>
</tr>
<tr>
<td>8.5</td>
<td>Some equivalences between SCN annotations and ACL messages.</td>
<td>179</td>
</tr>
<tr>
<td>8.6</td>
<td>Relation <em>types</em> from the process AuditNewPub.</td>
<td>180</td>
</tr>
<tr>
<td>8.7</td>
<td>Initial organizational assets.</td>
<td>182</td>
</tr>
<tr>
<td>8.8</td>
<td>Equivalences between SCN annotations and ACL messages for <em>AgRequest</em>.</td>
<td>187</td>
</tr>
<tr>
<td>8.9</td>
<td><em>PubCarrierAg</em>'s initial strategies in <em>AuditNewPub</em></td>
<td>193</td>
</tr>
<tr>
<td>8.10</td>
<td>New arcs found with the MDL score</td>
<td>194</td>
</tr>
<tr>
<td>8.11</td>
<td>Configurations for experiment 2</td>
<td>196</td>
</tr>
<tr>
<td>8.12</td>
<td>Performance evolution on self-configuration.</td>
<td>198</td>
</tr>
<tr>
<td>8.13</td>
<td>Performance comparison with the three different configurations.</td>
<td>200</td>
</tr>
<tr>
<td>8.14</td>
<td>Strategies by correction rule in experiment 3</td>
<td>201</td>
</tr>
<tr>
<td>8.15</td>
<td>Decision making of the <em>PubCarrier</em> for each correction rule.</td>
<td>202</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The present research is focused on the problem of implementing Organizational Intelligence in large organizations. In this sense, we propose integrating AI techniques around a holistic theory that allows us modeling organizational processes and organization members. As a result I will develop a software platform to automate organization processes with the participation of organizational members.

1.1 Organizational Intelligence and Artificial Intelligence

In Knowledge Management (KM), several approaches have been proposed for addressing the necessity of improving the competitiveness of organizations. An approach that is interesting due to its treatment of organizations as intelligent entities is Organizational Intelligence (OI).

In a broad sense, Organizational Intelligence is defined as “the capability of an organization to comprehend and conclude knowledge relevant to its business purpose” [93]. In particular it can be seen as the ability to develop, share and use knowledge, reflect and learn from experience, make sense of complex situations and act effectively. Veryard [90] considers that OI is achieved through “collaborative problem-solving between people and technical artifacts within and beyond complex enterprises”. Finally, McMaster define OI as “the capacity to sense, make sense, and act in flexible, creative, adaptive ways” [62].

In this ways, some authors recommend structuring processes as goal-oriented and improving knowledge exchange among members of the organization. Other authors propose that OI is the sum of the intelligence of its members. In the end, these authors propose that the intelligence showed by an organization relies on the process, their participants and how they are empowered with knowledge and decision making for achieving clearly defined objectives.
This kind of intelligence allows the organization be quickly adapted to changes. The capability for adaptation is based on learning mechanisms that correlates external feedback with internal decisions. There are several proposals for organizational learning in KM; nevertheless, few of them are supported by a computation framework capable of processing information online [78].

In this sense, the Artificial Intelligence (AI) discipline has developed paradigms and tools that have shown to be useful for the achievement of intelligent organizations with the learning capabilities described above. Machine Learning (ML) techniques have been applied to discover hidden patterns on huge data repositories built from operational data. ML has demonstrated to be valuable on applications of Business Intelligence (BI) like fraud detection and market segmentation.

Knowledge Representation (KR) has developed several symbolic logic systems for representing rules that have been used for representing organizational norms or user preferences, for instance. Logic frameworks proposed by KR enable automatic inference, but it becomes intractable with a large amount of rules or data. Another recent contribution of KR has been the Ontology formalism, which have been used for describing kinds of objects of a particular domain and reasoning about the represented objects.

Expert Systems are another contribution of AI for capturing the expertise of one or more human experts in certain matter. Several approaches on ML, KR and Case-Based Reasoning (CBR) were proposed for representing knowledge and perform automatic reasoning. On the other hand, the Intelligent Agent paradigm allowed thinking on autonomous entities acting on behalf of human users for repetitive tasks with a high degree of autonomy. Furthermore, the social dimension of intelligent agents allowed thinking on groups of agents interacting in order to solve complex distributed problems, proceeding to the Multi-Agent Systems (MAS) paradigm.

Recently, several MAS approaches have been inspired on organizations for guiding agent design and coordination towards the achievement of organizational goals. One of these approaches is the Electronic Institution (EI) paradigm [82, 5]. EI has been proposed as a framework for regulating interactions among heterogeneous agents in open systems. Participant agents modeled through roles must interact with other agents according to a specification given in terms of illocutions, norms, and protocols. Interactions are organized as scenes that are specified as illocution-based protocols. Thanks to the standard EI illocution mechanism for information exchange human users and software agents can interact with each other.

Recently, causality has been reintroduced in the scientific discourse and has incurred in Artificial Intelligence with the purpose of simplifying inference algorithms and make them more intuitive or natural. This is the case of Bayesian Causal Networks [71] and the nonmonotonic causal logic C+ [34].
1.2 Causality in the Science of Design

In [83], Herbert Simon highlights the importance of design in Artificial Intelligence disciplines and concludes that design is a science per se, the science of design. An intelligent artifact is designed expecting that it behaves in an intelligent way. Simon proposes a curriculum to consider on artificial intelligence design: evaluation of alternatives, optimum computation, formal representation of design (specification), learning, search spaces representation and hierarchical organization.

Aristotelian Causality and its revision by scholastic philosophers provide notions that can be used for covering this agenda. More than two thousand years ago, Causality provided simple and intuitive explanations of natural phenomena. Causality explains change in terms of cause-effect relationships on which the conditions needed for a change are called causes, and the regular consequences of it are called effects. Causes are classified by Aristotle as material, formal, efficient and final causes. In order to manage change, the notion of action is used for grouping sets of causal relations.

Any system or artifact made by man, and every one of its components, is designed with a purpose, the final cause of its design, even when this is not explicitly stated. No system can converge to a solution without guidance. Even a Genetic Algorithm has a fitness function that guides the development of simulated populations until finding a suitable solution.

For causality, the specification of an intelligent artifact and its components is represented by formal causes. Potential attributes of an entity can be used for describing its capabilities. Aristotle proposed a classification of accidents that can be predicated of an entity. In this way, the specification of a system or component can be given in terms of potential accidents and constraints that the entity must observe in order to comply with its purpose. These forms or definitions are given by the designer and are inherently aligned with the purpose of the artifact.

Brentano distinguished agents (efficient causes) from objects by the capability of the formers for having intentions, calling them intentional entities. In this way, intentional agents can be characterized as intentional entities with the capacity for elaborating plans that lead to the achievement of its finality and foreseeing the consequences of its actions on it. Furthermore, learning can be expressed as the capability of an intelligent agent for improving the accuracy of the prediction of this causal effect. The last can be done using past experiences and incorporating additional criterions in such calculations.

Finalities of the components contribute with the finality of the whole. So instead of forcing these agents to follow a protocol or observing some norms, the system can be designed in such a way that participants be motivated to cooperate.

An intelligent agent aware of its capabilities can evaluate its options in function of the plans it knows and the effects of its actions on its finality. As long as the agent is aware of the direct or indirect effects of its actions, he is capable of making decisions aligned with its finality. Instead of talking about the utility of an action we could refer
CHAPTER 1. INTRODUCTION

to its contribution to agent's finality, expressed in terms of truth and goodness. This is, an action can truly achieve or not certain objectives considered good by the agent. The optimum choice an agent can do is that which most contributes to its finality.

A causal design shows a hierarchical organization in the decomposition of finalities and on the creation of agents. The finality of a component of a system is aligned (contributes) to the finality of the system insofar as the former is a mean for the latter or both finalities have something in common. In this way, the finality of the system will cause the creation of agents with aligned finalities in order to fulfill its own finality.

1.3 Motivation

Even when the aforementioned AI approaches have demonstrated their usefulness for aims pursued by organizations, they only impact certain aspects of the golden goal called Organizational Intelligence. On each case, the articulation of these technologies respond to specific needs, and the provided solutions show intelligent behaviors that not necessarily obeys to a single design approach. For example, we can find a clustering approach inspired in ant behavior, working with an optimization algorithm inspired in simulated annealing.

Additionally, the impact of AI technologies is limited by the computational complexity inherent to some of these formalisms. Large organizations demand solutions that operate on industrial scale with huge amounts of data, on real time and under multiple and changing constraints imposed by a competitive environment and the complexity of the organization.

Even when the Electronic Institutions approach, one of the most suitable tools for the purpose of this research, deals with the specification of organizational processes and norms, it does not consider organizational or individual's goals and intelligence, neither provide mechanisms for adapting an actual implementation to changes in the process specification.

1.4 Problem Statement and Context

In order to give another step towards the carrying out of Organizational Intelligence I will tackle the problem of endowing organizational processes with intelligence in large organizations through causal design. To do so, I will characterize the organization as an entity which intelligence resides on the design and the implementation of its processes and on the alignment of its goals with those of its participants.

The organizational process must have a clear goal and designate obligations to organization's participants. Its members must be able to delegate tasks and transferring knowledge to autonomous software agents.
1.5 HYPOTHESIS AND RESEARCH QUESTIONS

I consider that organizational processes are static so participants, human or software, must learn and be adapted to environmental changes. Environmental conditions of automated processes are represented by the users’ feedback or intervention, the effectiveness of external systems, and the availability of external resources.

Inside the organization, goals and performance metrics can be adjusted meanwhile norms and protocols are considered fixed organizational constraints. Such constraints must be observed by agents in order to make decisions that benefit both the agent and the organization.

1.5 Hypothesis and Research Questions

The main hypothesis of my research is that Aristotelian Causality can be used as a theoretical framework that gives coherence to the integration of AI techniques in the pursuing of Organizational Intelligence.

In this way, the main research question is formulated as follows:

- How do we endow a process in a large organization with organizational intelligence through the introduction of AI techniques through a theory of causal design?

In order to answer this question I formulated the following:

- Can an organization be modeled as an intelligent entity?
- Can goal-oriented organizational processes be specified in terms of causal relations?
- Can these causal models be used by intelligent entities for making decisions and learn through experience?
- How should be managed knowledge transference from human experts to software agents?
- Can the proposed approach be applied on an industrial scale scenario?

1.6 Solution Overview

In the first place, I use causality theory as a theoretical framework for modeling goal-driven processes in terms of the goals they pursue, their components and tasks involved in their development. This framework will allow selecting compatible AI techniques and integrate them in a common software platform. Such AI techniques must count with efficient algorithms that allow its deployment in large scale scenarios. These technologies
must support the detection of environmental changes and provide adaptability to the solution components.

The proposed software platform will be integrated in a MAS platform that allows the interaction of human users and software agents. These agents will be responsible for performing well structured tasks and learn through experience, always observing organizational goals.

It will be proposed a methodology for modeling organizational processes in terms of the proposed theoretical framework. This methodology must allow specifying mechanisms of knowledge transference and tasks delegation. The resulting specification must serve for the implementation and deployment of software agents required by the process.

Finally, I will test my approach through the modeling and implementation of a process in a large organization. My framework must allow identifying components and actors of the process, as well as modeling their interaction and dependencies. The software platform must support the implementation of the process accessing organizational resources and requesting the participation of the organization’s members. In addition to the organizational goal pursued by the process, it will be included autonomic goals for demonstrating the capabilities of our approach for endowing an organizational process with organizational intelligence (according to the AC proposal).

1.7 Main Contributions

I will propose a definition of intelligent entities from a causal perspective. In this sense I will propose a theory of artificial intelligence design on which Aristotelian notions are represented trough modern AI formalisms: an ontological framework that captures metaphysics notions and a semantic extension of Bayesian Causal Networks.

This theory will be used for defining intelligent organizations and intelligent agents. Then I will develop an intelligent agent and a MultiAgent System architecture that follows this theory and adopts these formalisms. Additionally, I will provide a methodology and set of tools for specifying and automating processes in large organizations.

Through a case study will be tested the capabilities of this framework for: learning from experience, transferring knowledge from human experts to software agents and adapting to environmental changes.

1.8 Thesis Organization

In Chapter 2 Organizational Intelligence is defined. In this chapter I present philosophical notions that will serve for proposing the integrative framework and some modern AI approaches that formalize (at some extent) these notions. Then it presents a compilation of multiagent frameworks and intelligent agent architectures compatible with
1.9. SUMMARY

In this chapter it is explained how Organizational Intelligence (OI), a notion proposed by Knowledge Management, has been benefited through the application of AI techniques. In a broad sense, OI proposes to treat the organization as an intelligent entity capable of detecting environmental changes and adapt to them. Machine Learning (ML), Knowledge Representation (KR), Expert Systems and Multi-Agent Systems (MAS), are goal-driven organizations. Finally, this chapter presents some notes about the design of artificial intelligence and a definition of Autonomic Computing.

Chapter 3 presents an ontological framework that allows modeling statically and potentially the organization and its components. This framework uses Description Logic (DL) formalisms for providing intelligent entities specifications.

In Chapter 4 I introduce two types of dynamic causal models: a rule representation and a network of causal relationships. Through these formalisms we can model organizational processes and agent interactions.

In Chapter 5 I propose a theory called Causal Artificial Intelligence Design (CAID) inspired in scholastic notions of causality, metaphysics (ontologies) and intentionality for the design and implementation of intelligent entities. I enunciate four principles that provide a causal definition of intelligent entities and extend it for specifying Intelligent Organizations, Intelligent Agents and Human Users.

In Chapter 6 I introduce the Causal Agent, an intelligent agent architecture that provides an implementation of the causal definition of Intelligent Agents. There, it is described its goal-driven BDI inference engine supported by the ontological framework and causal models; and it concludes describing three different types of agents implemented through this architecture.

Chapter 7 presents the Causal Multi-Agent System (CMAS), a multi-agent system architecture based on Causal Agents and used for implementing organizational processes with the participation of users and intelligent agents. It is also provided a methodology that guides the modeling of organizational processes and the implementation of the required Causal Agents. The chapter concludes describing the operation of a CMAS.

Chapter 8 presents a case study in which the task of information auditing is automated through the design and implementation of a CMAS. It is characterized a solution with autonomic capabilities on which human auditors transfer their expert knowledge to intelligent agents. There are documented two implementations: one through Electronic Institutions and another through a CMAS. This chapter concludes presenting some experimental results.

Finally, Chapter 9 presents a comparison of this approach with other frameworks as well as final remarks and future work.
example of AI disciplines that have contributed in this sense.

I motivated the use of causality theory as a tool for designing intelligent artifacts. Causality and other notions borrowed from scholastic philosophy are used for providing: formal representations of artifacts, learning, cooperation, evaluation and selection of optimal choices, and hierarchical organization.

My research is motivated by the need of an integrated approach that apply AI techniques in the construction of OI. The complexity of some AI formalisms and the lack of integration between them have made difficult their application in real world scenarios. Even when ELs have provided an approach that enables and regulates interactions between human and software participants, process modeling doesn’t consider organizational or individual goals nor participants’ capabilities.

It is focused on the problem of designing and implementing intelligent organizational processes. Process design must be prepared for providing organizational adapting to environmental changes, consider participants’ obligations and knowledge transference from expert humans to software agents. I consider that the capacity of the organization for adapting relies on software agents but it is driven by organizational goals.

I started this research asking how to endow processes in large organizations with organizational intelligence through the introduction of AI techniques inspired in causality. To answer this question I focused in three specific proposals for implementing OI: goal-oriented processes, adaptation to environmental changes and knowledge transference.

In order to provide an integral solution to this problem, I will use causality as a holistic theory that integrate goals, actors and knowledge in the design and implementation of organizational processes. Then I will select AI approaches suitable with this theory and integrate them in a common platform. This framework will allow implementing a MAS that support environmental changes and enable knowledge transference. Finally, I will use the developed framework for automating a process in a real organization.

The main contributions of this work are the following:

1. a definition and computational implementation of intelligent entities based on causality,

2. a novel framework for modeling goal-driven organizational processes that enable knowledge transference and task delegation from human users to software agents, and

3. an agent-based platform capable of adapting to organizational and environmental changes.
Chapter 2

Background and Related Work

In this chapter I present some approaches that answer to one or various of the objectives pursued through Organizational Intelligence. In the first place there are presented some notions and approaches proposed by Knowledge Management practitioners. Next there are presented some Autonomic Computing implementations as an industrial approach to OI on which there are given scalable solutions to complex and large-scale problems.

Then I present the curricula proposed by Simon for designing intelligent artifacts, followed by the philosophical foundations on which will rely my theory of design: Aristotelian metaphysics and causality revised by scholastic philosophers. I continue with the revision of modern AI approaches to these theories, among which I can highlight Description Logics and Bayesian Causal Networks.

Finally I present some architectures of Intelligent Agents and MultiAgent Systems with which my approach can be compared. Among the last I can highlight Withestein Technologies, a company that develops autonomic computing software based on agents.

2.1 Organizational Intelligence in Knowledge Management

Knowledge Management (KM) has proposed the notion of Organizational Intelligence (OI) as a way for improving organization competitiveness. Along with Learning Organizations, OI is focused on the generation and diffusion of information and knowledge across the organization. In this sense, Corporate Memories provide an approach for storing and delivering knowledge across the organization.

From a cognitive perspective, Mary Ann Glynn defines Organizational Intelligence as "an organization's capability to process, interpret, encode, manipulate and access information in a purposeful, goal-directed manner, so it can increase its adaptive potential in the environment in which it operates" [35]. For Glynn, organizational intelligence is a social outcome and is related to individual intelligence by mechanisms of aggregation,
cross-level transference, and distribution. For her, an organization is more intelligent if its members are more intelligent and the organization have more and better diffusion and institutionalization mechanisms of complex knowledge encoded declaratively and procedurally.

Similarly, Sydanmaanlakka proposes that an Intelligent Organization must: improve the organization’s capacity to learn, develop the right competencies needed in the organization, improve individual performance throughout the organization, achieve better business results, and manage in the ever more competitive global business environment [86].

This notion is closely related to that of Learning Organizations. Martha Gephart mentions as key elements of a Learning Organization: the generation and sharing of knowledge, the continuous analysis of organizational routines or procedures, the critical systemic thinking that enable to discover links and feedback loops, among others. M. Leann Brown classifies organizational learning on three forms: 1) changes on routines, policies, goals and paradigms; 2) ideational interaction among organizational entities; and 3) changes on leader’s ideas, beliefs and paradigms [12]. Organizational learning can be done through several methods among which we can find information acquisition, experience, research, customer feedback and by doing things. On this sense, KM have proposed several ways for capturing insights and experiences either embodied in individuals or embedded in organizational processes or practice.

Barbara Levitt proposes a process oriented definition of organizational learning as routine-based, history-dependent and target-oriented [61]. For Levitt, learning comes from direct and others experience and considers that the knowledge or information obtained must be captured organizationally. In this sense, Jay Liebowitz and other authors have proposed the design of Corporate Memories (CM) as active repositories to store and distribute information and knowledge relevant for business operation across organization’s members.

In this sense we can find a variety of works on which KM practitioners provide best practices for improving organization competitiveness. For instance, in [3] Verna Allee proposes best practices for identifying core knowledge competencies and individual expertise of organizations members in order to understand knowledge creation and learning. This best practices include guidelines, design principles, analogies, and conceptual frameworks.

In [56], Liebowitz provides a set of surveys and questionnaires for capturing organization members knowledge, auditing knowledge management practices and assessing learning strategies in the organization. Furthermore, Liebowitz and other authors developed an intelligent agent-based knowledge management system where there are distinguished three kinds of agents: User Agent, Knowledge Agent and Knowledge Manager; the first controls human-computer interaction, the second manages and indexes knowledge, and the third monitors changes and correlates information.

Continuing in the direction of knowledge distribution, Liebowitz and Beckman propose
the notion of Corporate Memory, “an explicit, disembodied, persistent representation of the knowledge and information of an organization” [55]. They classify corporate memories according to the capture and distribution mechanisms that are used to build the memory; these mechanisms can be either passive or active procedures. Passive capture and passive distribution is just an archive called a knowledge attic which is consulted when needed. Active capture and passive distribution is a repository with automated capture facilities called a knowledge sponge. Passive capture and active distribution is called a knowledge publisher which has automatic distribution facilities to interested users. Finally, active capture and active distribution is a corporate memory called a knowledge pump.

2.2 Autonomic Computing

Autonomic Computing (AC) provides a pragmatic definition of intelligent systems. Autonomic Computing is an initiative started by IBM in 2001 with the purpose of developing computer systems capable of self-management, to overcome the rapidly growing complexity of computing systems management, and to reduce the barrier that complexity poses to further growth [48].

Autonomic Computing refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes whilst hiding intrinsic complexity to operators and users. An autonomic system makes decisions on its own, using high-level policies; constantly checks and optimizes its status and automatically adapt itself to changing conditions. Autonomic computing frameworks are usually composed by Autonomic Components interacting with each other. An autonomic component can be modeled in terms of two main control loops (local and global) with sensors (for self-monitoring), effectors (for self-adjustment), knowledge and planer/adapter for exploiting policies based on self- and environment awareness.

In an Autonomic System, the human operator does not control the system directly, but defines general policies and rules that serve as an input for the self-management process. For this process, IBM defined the following four functional areas:

Self-Configuration: consists on the automatic configuration of components.

Self-Healing: includes automatic discovery, and correction of faults.

Self-Optimization: consists on automatic monitoring and control of resources to ensure the optimal functioning with respect to the defined requirements.

Self-Protection: is formulated as the proactive identification and protection from arbitrary attacks.

Autonomic approaches are mainly oriented to specific services but also exist proposals for generic architectures [41]. Additionally, adaptation strategies can be internalized or
externalized; meanwhile the former are highly dependent of the system, the latter can be used in different applications.

2.3 Artificial Intelligence Design

In his book The Science of the Artificial [83], Herbert Simon points out the importance of design in Artificial Intelligence disciplines and proposes a theory for considering it as a science per se, the *science of design*. The curriculum he proposes consists on the following topics:


2. Computational methods for actually deducing which of the satisfactory alternatives is the optimum.

3. The formal logic of design: imperative and declarative logics.

4. The exploitation of parallel, or near-parallel, factorizations of differences.

5. The allocation of resources for search to alternative, partly explored action sequences.

6. The organization of complex structures and its implication for the organization of design processes: hierarchic systems.

7. Alternative representations for design problems.

According to Simon, "design is concerned with how things ought to be, with devising artifacts to attain goals". In this sense, modal and declarative logics provide a base for expressing design alternatives. These alternatives must be evaluated for choosing an optimal or at least a good solution that satisfy a set of design requirements (constraints). Means-ends analysis is proposed as a mechanism for searching for the optimal solution in a space of possibilities built from these requirements.

As well, Simon proposes that complex systems must be constructed in a hierarchy of levels. Components perform particular subfunctions that contribute to the overall function. He discusses the importance of representation of design: solving a problem is equivalent to finding a representation that makes the solution transparent. In order to have a better understanding of design problems, Simon points out the necessity of a taxonomy in which the identification of problems similarities and differences facilitate their solution.
2.4 Philosophical Foundations

This work is inspired in the Aristotelian-Thomist philosophy. Some fundamental notions are presented next.

2.4.1 Metaphysics

Created by Aristotle, and revised by Aquino, Metaphysics provides a general conceptualization of reality[4]. It conceives reality constituted by entities that have an essence that humans can recognize. Entity essence is identified by its characteristics or accidents and is captured through an abstraction process.

Aristotle classify accidents as intrinsic, extrinsic and mixed. Intrinsic includes quantitative (age, size, etc.), qualitative (color, shape, etc.) and relational (fatherhood, nationality, etc.) accidents, that is, what internally identify an entity. Extrinsic are relative to time (birth date, duration, etc.), location (position), possession and disposition. Mixed accidents explain interaction between entities: action is present in an entity when originates movement in another, meanwhile passion is present in entities that receive passively action from another.

Aristotle conceived the No Contradiction Principle that states that nothing can to be and not to be at the same time and on the same sense. Derived from this principle, it is the Third Excluded Principle that states that there is no middle between being and not being.

Aristotle treated the problem of movement or change too. He considered change as a transition on an individual from one state to another, always that individual can be able of reach the final state. He defined potency as the entity capacity to present certain accident. Act, opposite to potency, is the actual presence of the accident on the entity. Having certain accident in potency doesn’t imply that entity presents it actually, but denotes possibility.

Additionally, Aquino recognize certain characteristics that transcend entities, i.e. are present on every entity. Transcendentals are: unity (entity exist on itself, it exists independently of accidents), truth (something is true as long as is an entity), goodness (an entity is good as long as it acts according to its essence) and beauty (things’ truth and goodness delights whom contemplate them).

Trascendentals recognize in Scholastic tradition entities proclivity to perfection. Perfection is concordance between entities definition and its reality.

2.4.2 Causality

Causality[4] refers to the set of all particular “causal” or “cause-and-effect” relations. Most generally, causation is a relationship that holds between events, properties, vari-
ables, or states of affairs. Causality implies at least some relationship of dependency between the cause and the effect. For example, deeming something a cause may imply that, all other things being equal, if the cause occurs the effect does as well, or at least that the probability of the effect occurring increases. Cause chronologically precedes the effect.

In natural languages, causal relationships can be expressed by the following causative expressions: i) a set of causative verbs [cause, make, create, do, effect, produce, occasion, perform, determine, influence; construct, compose, constitute; provoke, motivate, force, facilitate, induce, get, stimulate; begin, commence, initiate, institute, originate, start; prevent, keep, restrain, preclude, forbid, stop, cease]; ii) a set of causative names [actor, agent, author, creator, designer, former, originator; antecedent, causality, causation, condition, fountain, occasion, origin, power, precedent, reason, source, spring; reason, grounds, motive, need, impulse]; iii) a set of effective names [consequence, creation, development, effect, end, event, fruit, impact, influence, issue, outcome, outgrowth, product, result, upshot].

In Metaphysics and Posterior Analytics, Aristotle stated: “All causes of things are beginnings; that we have scientific knowledge when we know the cause; that to know a thing’s existence is to know the reason why it is”. With this, he set the guidelines for all the subsequent causal theories by specifying the number, nature, principles, elements, varieties, order of causes as well as the modes of causation. Aristotle’s account of the causes of things may be qualified as the most comprehensive model up to now.

According to Aristotle’s theory, all the possible causes fall into several wide groups, the total number of which amounts to the ways the question “why” may be answered; namely, by reference to the matter or the substratum; to the essence, the pattern, the form, or the structure; to the primary moving change or the agent and its action; and to the goal, the plan, the end, or the good. As a result, the major kinds of causes come under the following divisions:

- The Material Cause is that from which a thing comes into existence as from its parts, constituents, substratum or materials. This reduces the explanation of causes to the parts (factors, elements, constituents, ingredients) forming the whole (system, structure, compound, complex, composite, or combination) (the part-whole causation).

- The Formal Cause tells us what a thing is, that any thing is determined by the definition, pattern, essence, whole, synthesis, or archetype. It embraces the account of causes in terms of fundamental principles or general laws, as the whole (macrostructure) is the cause of its parts (the whole-part causation).

- The Efficient Cause is that from which the change or the ending of the change first starts. It identifies “what makes of what is made and what causes change of what is changed” and so suggests all sorts of agents, nonliving or living, acting as the sources of change or movement or rest. Representing the current understanding
of causality as the relation of cause and effect, this covers the modern definitions of "cause" as either the agent, agency, particular events, or states of affairs.

- The Final Cause is that for the sake of which a thing exists, or is done - including both purposeful and instrumental actions. The final cause, or telos, is the purpose, or end, that something is supposed to serve; or it is that from which, and that to which, the change is. This also covers modern ideas of mental causation involving such psychological causes as volition, need, motivation, or motives; rational, irrational, ethical - all that gives purpose to behavior.

Additionally, things can be causes of one another. Causing each other reciprocally, as hard work causes fitness, and vice versa - although not in the same way or function: the one is as the beginning of change, the other as the goal. (Thus Aristotle first suggested a reciprocal or circular causality - as a relation of mutual dependence, action, or influence of cause and effect.) Also, Aristotle indicated that the same thing can be the cause of contrary effects - as its presence and absence may result in different outcomes.

Aristotle marked two modes of causation: proper (prior) causation and accidental (chance) causation. All causes, proper and incidental, can be spoken as potential or as actual, particular or generic. The same language refers to the effects of causes; so that generic effects assigned to generic causes, particular effects to particular causes, and operating causes to actual effects. It is also essential that ontological causality does not suggest the temporal relation of before and after - between the cause and the effect; that spontaneity (in nature) and chance (in the sphere of moral actions) are among the causes of effects belonging to the efficient causation, and that no incidental, spontaneous, or chance cause can be prior to a proper, real, or underlying cause per se.

2.5 Modern Approaches to Metaphysics

A recent knowledge representation formalism called Ontologies has captured attention of AI researchers. Descendant of Frame systems and Semantic Networks, Ontologies have provided a mechanism for describing objects of an specific application domains. These descriptions or definitions allows expressing common sense information that can be further used for inferring implicit relations during problem solving. Besides it has become a standard mechanism in Multi-Agent Systems for facilitating communications and enabling common understanding.

Diversity of ontologies for representing the same objects in the same or in different domains motivated the necessity of mapping concepts. One solution in this sense was the proposal of foundational ontologies as intermediary schemas. Some of these foundational ontologies found in Metaphysics a solid framework for representing any kind of object. Nevertheless, they were not widely used because: 1) definitions were too complex and detailed, and 2) it was not possible to develop automatic mapping mechanisms. Up to now, ontology mapping (or matching) is still an open issue.
Description Logics (DL) have provided a formal representation for ontologies with sound inference algorithms. Ontological definitions are expressed through logical constructors expressed in terms of concepts and roles, meanwhile objects are expressed through concept and role assertions that describe their characteristics.

A basic DL inference known as concept subsumption allows to check the consistency of a set of definitions and a set of objects w.r.t. a set of definitions. It also enables identifying objects matching a given description (query). Between the type of queries that can be computed in DL representations, conjunctive queries allow computing query containment which is used to determine subsumption and disjointness among simple DL expressions.

This section concludes with a set of standards proposed by the World Wide Web Consortium (W3C) for representing and querying ontologies. These standards are closely related to the formalisms proposed by the Description Logics community.

2.5.1 Ontologies

Metaphysics, as well known as ontologies, are recovered from classical philosophy for expressing the universe of discourse on computer systems. In [37], Gruber defines an ontology as “an explicit specification of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence.” Gruber clarifies that for AI systems, what “exists” is that which can be represented, i.e. the Universe of Discourse.

In an ontology, definitions associate the names of entities in the universe of discourse (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. Formally, an ontology is the statement of a logical theory.

Common ontologies are used to describe ontological commitments for a set of agents so that they can communicate about a domain of discourse without necessarily operating on a globally shared theory. An agent commits to an ontology if its observable actions are consistent with the definitions in the ontology. A commitment to a common ontology is a guarantee of consistency, but not completeness, with respect to queries and assertions using the vocabulary defined in the ontology.

Gruber proposes five criterions for designing ontologies: clarity (objective and preferably complete definitions), coherence (concepts and axioms should be logically consistent), extendability (anticipate the uses of the shared vocabulary), minimal encoding bias (minimize dependence on particular symbol-level encodings), and minimal ontological commitment (which allows the parties committed to the ontology freedom to specialize and instantiate the ontology as needed).

Some languages like Knowledge Interchange Format (KIF) [32], Simple HTML Ontology Extension (SHOE) [43], the DARPA Agent Markup Language (DAML) [8] and the Ontology Interchange Language (OIL) [27] were the first formal languages for expressing
computational ontologies. They were used primarily in multiagent systems for encoding messages exchanged between agents.

2.5.2 Foundational Ontologies

Foundational ontologies were proposed as an alternative for mapping ontologies. Also known as upper-level ontologies, these contain concepts and properties that represent very general concepts to which every domain-dependent definition would be mapped to. Some of them borrow notions from Aristotelian metaphysics. Some of these projects were criticized due to the incapability for making an automatic mapping between domain-specific definitions and those proposed by these works. Others like CyC, reached high levels of complexity making their use impractical.

DOLCE

Welty and Guarino describe in [92] a methodology for analyzing universal properties of a taxonomy. They define a classification of meta-properties on which the most important notions are identity, essence, unity and dependency. Using these meta-properties,

DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) [30] is a foundational ontology of particulars or concepts. In DOLCE there are distinguished endurants (continuants) from perdurants (occurrences). The former, invariant in time, the latter, occurring in one or more points in time and on different ways. The upper level elements, after Entity, are Abstraction, Endurant and Perdurant. From the first there are derived concepts like Set, Region, SpatialRegion, TemporalRegion, etc. From Abstraction there are derived qualities and substances (physical and non physical). In Endurant there are classified Events, States, Processes, using spatial and temporal regions.

DOLCE make a distinction between qualities and regions of qualities. The former are characteristics shown by a specific individual, meanwhile the latter are the possible values qualities can hold. The values a quality adopt along time are called quales. The notion of quality region is used for matching similar quales. Space and time are considered qualities, as well as color, weight, etc.

DOLCE authors consider substances as stable aggregates of qualities: they are endurants that may have qualities. The condition for being (exist) of a physical substance is the existence of a direct spatial quality. Physical objects with intentionality are called Agents, those which do not posses it are called Non-Agent. Non-physical objects are divided in mental (owned by an agent) and social (accepted by a group of agents).

Using DOLCE it was possible identify problems between WordNet definitions like confusion between concepts and individuals, confusion between the object level and meta levels, heterogenous levels of generality, among others. Nevertheless, expressing WordNet definitions in DOLCE was not automated.
CHAPTER 2. BACKGROUND AND RELATED WORK

BFO

The Basic Formal Ontology (BFO) [36], is another upper-level ontology inspired in philosophy that incorporates theories of continuants and occurrents, mereology, mereotopology, dependence and location. BFO has been used for representing medical terms [49] in order to improve their internal coherence and serve as translation hub. BFO also includes biological classes and granular partitions. LinKBase and Mapping Databases onto Knowledge Systems (MADBooks) are tools that use BFO for mapping external databases.

HowNet

En [89], Veale proposes that re-categorization may be used for generating analogies using the HowNet ontology. HowNet is a lexical resource similar to WordNet that provides explicit semantic definitions for each one of the defined terms. In HowNet, definitions are enriched with causal relations which are used for generating automatically a taxonomy for representing metaphors and analogies.

GOL

Degen, Heller, Herre y Smith present in [23] the General Ontological Language (GOL), based on notions borrowed from Aristotle and Brentano. In this language, are distinguished sets from individuals, and universals from their extensions. Individuals are categorized in moments, substances, cronoids (time), topoids (space) and situods (situations = individuals + moments). Situoids are used for defining phenomena and dynamical entities like change, processes, events and states.

Additionally, authors propose some operators for reasoning on these ontologies; membership (to sets), instantiation, inherence (predicated of), part-of, reflexive part-of, framing, foundation, contained in, occupy, component-of, and assignment. They also include a ternary relationship part-hole specific for the domain and dependent of a universal. With these operators they can provide axioms of order and existence, instantiation, sets, moments-substances-inherence, part-of, cronoids, topoids and situoids.

CyC

The Cyc Knowledge Base (KB) ¹, expressed through the language CycL, is a formalized representation of a vast quantity of fundamental human knowledge: facts, rules of thumb, and heuristics for reasoning about the objects and events of everyday life. The KB consists of terms—which constitute the vocabulary of CycL—and assertions which relate those terms.

The Cyc KB is divided into thousands of "microtheories", each of which is essentially a bundle of assertions that share a common set of assumptions; some microtheories are focused on a particular domain of knowledge, a particular level of detail, a particular interval in time, etc. The microtheory mechanism allows Cyc to independently maintain assertions which are prima facie contradictory, and enhances the performance of the Cyc system by focusing the inferencing process.

At the present time, the Cyc KB contains nearly five hundred thousand terms, including about fifteen thousand types of relations, and about five million facts (assertions) relating these terms. New assertions are continually added to the KB through a combination of automated and manual means. The main application of CyC has been on natural language processing, but has also being used for mapping ontologies [77].

2.5.3 Description Logics

Description Logics (DL) are a family of knowledge representation languages developed in the 1980s as an extension of frames and semantic networks complemented with a formal logic-based semantics. Currently is used to represent terminological knowledge of an application domain in a structured and formally well-understood way[7].

Syntax and Semantics

DL are used to define concepts and relations between them. The basic step for the terminology construction is provided by two disjoint alphabets of symbols that are used to denote atomic concepts, designated by unary predicate symbols, and atomic roles, designated by unary predicate symbols; the latter are used to express relationships between concepts. These are elementary descriptions.

Complex descriptions can be built from them inductively with concept constructors and role constructors. Concept constructors in the $\mathcal{AL}$ sublanguage are formed according to the following syntax rule$^2$:

$$C, D \rightarrow \begin{cases} A | & \text{(atomic concept)} \\ \top | & \text{(universal concept, top concept)} \\ \bot | & \text{(bottom concept)} \\ \neg A | & \text{(atomic negation)} \\ C \cap D | & \text{(intersection)} \\ \forall R.C | & \text{(value restriction)} \\ \exists R.\top & \text{(limited existencial quantification)} \end{cases} \quad (2.1)$$

An interpretation $\mathcal{I}$ consists of a non-empty set $\Delta^\mathcal{I}$ (the domain of the interpretation) and an interpretation function, which assigns to every atomic concept $A$ a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$.

$^2$ A is an atomic concept, R is an atomic role, and C and D denote concept descriptions.
and to every atomic role $R$ a binary relation $R^I \subseteq \Delta^T \times \Delta^T$. The interpretation function is extended to concept descriptions by the following inductive definitions:

$$
\begin{align*}
\top^I &= \Delta^T \\
\bot^I &= \emptyset \\
A^I &= \Delta^T \setminus A^T \\
(C \cap D)^I &= C^I \cap D^I \\
(\forall R.C)^I &= \{ a \in \Delta^T | \forall b. (a, b) \in R^T \rightarrow b \in C^T \} \\
(\exists R.T)^I &= \{ a \in \Delta^T | \exists b. (a, b) \in R^T \}
\end{align*}
$$

(2.2)

There are several possibilities for extending $\mathcal{AL}$ in order to obtain a more expressive DL. The three most prominent are adding additional concept constructors, adding role constructors, and formulating restrictions on role interpretations. Basically each extension is assigned a letter or symbol. For concept constructors, the letters/symbols are written after the starting $\mathcal{AL}$, for role constructors, the letters/symbols are superscripts, and for restrictions on the interpretation of roles as subscripts.

Restrictions on role interpretation can be: functional roles ($f$) and transitive roles ($R^+$). In functional roles, an interpretation must map features $f$ to functional binary relations $f^I \subseteq \Delta^T \times \Delta^T$; can be written as $f^T(a) = b$. An interpretation must map transitive roles $R \in N_{R^+}$ to transitive binary relations $R^I \subseteq \Delta^T \times \Delta^T$.

Additional concept constructors are shown in Table 2.1, including the symbol that represents it.

Role constructors take role and/or concept descriptions and transform them into more complex role descriptions. Table 2.2 shows some of them.

A DL knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ consists of a set of terminological axioms (called a $\mathcal{T}$Box $\mathcal{T}$) and a set of assertional axioms or assertions (called an $\mathcal{A}$Box $\mathcal{A}$). The syntax and semantics of these axioms are shown in Table 2.3. An interpretation $\mathcal{I}$ is called a model of an axiom if it satisfies the statement in the last column of the table.

An equality whose left-hand side is an atomic concept or role is called a concept or role definition. A finite set of definitions is called a terminology or TBox if the definitions are
2.5. MODERN APPROACHES TO METAPHYSICS

<table>
<thead>
<tr>
<th>Name</th>
<th>Syntax</th>
<th>Semantics</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal role</td>
<td>(U)</td>
<td>(\Delta I \times \Delta I)</td>
<td>(U)</td>
</tr>
<tr>
<td>Intersection</td>
<td>(R \cap S)</td>
<td>(R^I \cap S^I)</td>
<td>(\cap)</td>
</tr>
<tr>
<td>Union</td>
<td>(R \cup S)</td>
<td>(R^I \cup S^I)</td>
<td>(\cup)</td>
</tr>
<tr>
<td>Complement</td>
<td>(\neg R)</td>
<td>(\Delta I \times \Delta I \setminus R^I)</td>
<td>(\neg)</td>
</tr>
<tr>
<td>Inverse</td>
<td>(R^-)</td>
<td>\{(b, a) \in \Delta I \times \Delta I</td>
<td>(a, b) \in R^I}}</td>
</tr>
<tr>
<td>Composition</td>
<td>(R \circ S)</td>
<td>(R^I \circ S^I)</td>
<td>(\circ)</td>
</tr>
<tr>
<td>Transitive closure</td>
<td>(R^+)</td>
<td>(\bigcup_{n \geq 1} (R^I)^n)</td>
<td>(+)</td>
</tr>
<tr>
<td>Reflexive-transitive closure</td>
<td>(R^*)</td>
<td>(\bigcup_{n \geq 0} (R^I)^n)</td>
<td>(*)</td>
</tr>
</tbody>
</table>

Table 2.2: Role constructors

<table>
<thead>
<tr>
<th>Name</th>
<th>Syntax</th>
<th>Semantics</th>
<th>Paragraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept inclusion</td>
<td>(C \subseteq D)</td>
<td>(C^I \subseteq D^I)</td>
<td>unambiguous, i.e. no atomic concept occurs more than once as left-hand side. Axioms of the form (C \subseteq D) for a complex description (C) are often called general inclusion axioms. A set of axioms of the form (R \subseteq S) where both (R) and (S) are atomic are called a role hierarchy. The fact that the knowledge base may contain a role hierarchy is indicated by appending a (\mathcal{H}) to the name of the DL.</td>
</tr>
<tr>
<td>Role inclusion</td>
<td>(R \subseteq S)</td>
<td>(R^I \subseteq S^I)</td>
<td></td>
</tr>
<tr>
<td>Concept equality</td>
<td>(C \equiv D)</td>
<td>(C^I = D^I)</td>
<td></td>
</tr>
<tr>
<td>Role equality</td>
<td>(R \equiv S)</td>
<td>(R^I = S^I)</td>
<td></td>
</tr>
<tr>
<td>Concept assertion</td>
<td>(C(a))</td>
<td>(a^I \in C^I)</td>
<td></td>
</tr>
<tr>
<td>Role assertion</td>
<td>(R(a, b))</td>
<td>((a^I, b^I) \in R^I)</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3: Terminological and assertional axioms

Inferences

A knowledge representation system based on DL is able to perform specific kinds of reasoning defined as logical inferences. Inferences can be performed for concepts, for TBoxes and ABoxes, and for TBoxes and ABoxes together. There is one main inference problem, namely the consistency check for ABoxes, to which all other inferences can be reduced.

The basic inference on concept expressions in DL is subsumption \((C \subseteq D)\). Determining subsumption is the problem of checking whether the concept denoted by \(D\) (the subsumer) is considered more general than the one denoted by \(C\) (the subsumee), i.e. checks whether the first concept always denotes a subset of the set denoted by the second one.

Another typical inference on concept expressions is concept satisfiability, which is the problem of checking whether a concept expression does not necessarily denote the empty concept. It is a special case of subsumption, with the subsumer being the empty concept.
An ABox $A$ is consistent with respect to a TBox $T$, if there is an interpretation that is a model of both $A$ and $T$. $A$ is consistent if it is consistent with respect to the empty TBox. The empty TBox is that on which definitions are described only by atomic concepts, i.e. no terminological axioms are included on it. An empty TBox is generated through the expansion of every concept $C$ with respect to $T$. The concept $C'$ is obtained from $C$ by replacing each occurrence of a name symbol $A$ in $C$ by the concept $D$, where $A \equiv D$ is the definition of $A$ in $T$.

Over an ABox $A$, one can pose queries about relationships between concepts, roles and individuals. The typical ABox inference on which such queries are based is instance checking, or the check whether an assertion is entailed by an ABox. We say that an assertion $\alpha$ is entailed by $A$ and we write $A \models \alpha$, if every interpretation that satisfies $A$ also satisfies $\alpha$.

The retrieval problem is, given an ABox $A$ and a concept $C$, to find all individuals $a$ such that $A \models C(a)$. A non-optimized algorithm for a retrieval query can be realized by testing for each individual occurring in the ABox whether it is an instance of the query concept $C$.

The dual inference to retrieval is the realization problem: given an individual $a$ and a set of concepts, find the most specific concept $C$ from the set such that $A \models C(a)$. Here the most specific concepts are those that are minimal with respect to the subsumption ordering $\subseteq$.

Reasoning Algorithms

For DL without negation, subsumption of concepts can usually be computed by so-called structural subsumption algorithms, i.e., algorithms that compare the syntactic structure of (possibly normalized) concept descriptions. While they are usually very efficient, they are only complete for rather simple languages with little expressivity.

In particular, DL with full negation and disjunction cannot be handled by structural subsumption algorithms. For such languages, so-called tableau-based algorithms have turned out to be very useful. In the area of DL, the first tableau-based algorithm was presented by Schmidt-Schauss and Smolka in 1991 for satisfiability of $\mathcal{ALC}$-concepts. Since then, this approach has been employed to obtain sound and complete satisfiability (and thus also subsumption) algorithms for a great variety of DL extending $\mathcal{ALC}$ (number restrictions, transitive closure of roles and transitive roles. In addition, it has been extended to the consistency problem for ABoxes and to TBoxes allowing general sets of inclusion axioms and more. Satisfiability of $\mathcal{ALCN}$-concept descriptions is PSpace-complete.

A tableau-based satisfiability algorithm can easily be extended to an algorithm that decides consistency of $\mathcal{ALCN}$-ABoxes, being its complexity PSpace-complete. If the algorithm additionally considers general inclusion axioms its complexity becomes ExpTime-
Decision procedures with lower complexity can be obtained by using the connection between DL and propositional modal logics. Schild was the first to observe that the language $\mathcal{ALC}$ is a syntactic variant of the propositional multi-modal logic $\mathcal{K}$, and the extension of $\mathcal{ALC}$ by transitive closure of roles corresponds to Propositional Dynamic Logic (PDL).

Conjunctive Queries

Let $N_V$ be a finite set of variables and $N_C$ be a finite set of constants. A conjunctive query (CQ) over a KB $K = (T, \mathcal{A})$ is a finite set of atoms of the form $A(v)$ or $r(v,v')$, where $v, v' \in (N_V \cup N_C)$, $A$ is a concept name and $r$ is a role, both occurring in $K$. For a CQ $q$ over $K$, let $\text{Var}(q)$ denote the variables occurring in $q$ and $\text{Dis}(q) \subseteq \text{Var}(q)$ denote the variables distinguished in $q$. A union of conjunctive queries (UCQ) $q$ is a formula of the form $q(x) = \bigvee_{i=1}^{n} \exists y_i. \text{conj}_i(x, y_i)$ being $\text{conj}_i(x, y_i)$ an atom like the one defined for CQs.

A match for $q$ in an interpretation $I$ is a mapping $\pi : \text{Var}(q) \rightarrow \Delta^I$ such that (i) $\pi(v) \in A^I$ for each $A(v) \in q$, and (ii) $(\pi(v), \pi(v')) \in r^I$ for each $r(v, v') \in q$. We write $I \models q$ if there is at least one match for $q$ in $I$. $I \not\models q$ denotes that there is no match for $q$ in $I$. If $I \models q$ for every model $I$ of $K$, then $K$ entails $q$, written $K \models q$. The query entailment problem is to decide, given $K$ and $q$, whether $K \models q$.

Query containment under constraints is the problem of determining whether matches of one query are contained in the matches of another query given a set of definitions contained in a TBox $T$. A query $q_1$ is contained in a query $q_2$ w.r.t. a TBox $T$ (written $T \models q_1 \subseteq q_2$), iff, for every model $I$ of $T$, $q_1(I) \subseteq q_2(I)$. Two queries are called equivalent w.r.t. $T$ iff $T \models q_1 \subseteq q_2$ and $T \models q_2 \subseteq q_1$, denoted $q_1 \equiv q_2$. Some approaches have being proposed in this sense and their complexity depends on the richness of the language[14, 44].

Query containment is transitive, i.e. if $q_1 \subseteq q_2$ and $q_2 \subseteq q_3$ then $q_1 \subseteq q_3$. Besides, a conjunctive query $q_1$ can be narrowed or specialized into another query $q_2$ by 1) replacing a variable by a constant, or 2) adding a constraint (another statement); in both cases, $q_2 \subseteq q_1$.

For instance, suppose that $q_1 = \{ ?x \text{ isA Animal} \}$ and that we specialize the query by doing $q_2 = \{ q_1 \cup ?x \text{ isA Dog} \}$. Whereas evaluating $q_1$ retrieves all the possible animals compiled in a knowledge base, $q_2$ will only find the dogs subset, i.e. $q_2 \subseteq q_1$.

Query answering

Query answering over a DL knowledge bases $K = (T, \mathcal{A})$ consists on retrieving information from $\mathcal{A}$ according to a conjunctive query $q$ interpreting the definitions given in $T$. This task can be done in several ways.
An approach consists on generating an ABox $\mathcal{A}'$ on which definitions of $\mathcal{T}$ are expanded through entailment rules; this is applied in the Jena Toolkit\(^3\). Hybrid approaches like \cite{hybrid} uses a rule engine to apply a domain-specific version of ABox-related entailments and uses a DL reasoner to reason over the TBox $\mathcal{T}$.

Another approach consists on rewriting a CQ or a UCQ $q$ according to $\mathcal{T}$ producing another UCQ $q'$ that can be evaluated directly over $\mathcal{A}$. In \cite{rosati}, Rosati defines a query-rewriting-based technique for answering unions of conjunctive queries in $\mathcal{EL}$ and $\mathcal{ELH}$. This technique is PTIME-complete with respect to both data complexity (i.e., with respect to the size of the ABox) and knowledge base complexity (i.e., with respect to the size of the knowledge base) and is NP-complete with respect to combined complexity (i.e., with respect to the size of both the knowledge base and the query).

In the other hand, Botoeva and colleges proposes in \cite{botoeva} an algorithm for rewriting UCQs posed to $\mathcal{DL-Lite}^{Horn}$ knowledge bases; $\mathcal{DL-Lite}^{Horn}$ is a logic from the extended DL-Lite family that contains horn concept inclusions and number restriction. The complexity of this algorithm is exponential in the size of the TBox.

### 2.5.4 Standardization

The World Wide Web Consortium (W3C), an organism founded by Tim Berners Lee, has promoted several initiatives for transforming the Web into a web of information. In this project, known as Semantic Web or Web 3.0, have being proposed standard languages for representing ontologies and for querying information represented through these schemas. These standards and their implementations are supported by current research on DL. Even when the expressive richness provided by these standards is not entirely supported by current implementations, choosing a suitable sub-logic is responsibility of the technology adopter.

#### Semantic Web

The Semantic Web was originally conceptualized by Tim Berners Lee\cite{berners} as the evolution of a Web that consisted largely of documents for humans to read to one that included data and information for computers to manipulate. The Semantic Web is a Web of actionable information (information derived from data through a semantic theory for interpreting the symbols).

The objective of Semantic Web is to provide a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. It is based on the Resource Description Framework (RDF).

The Semantic Web is about common formats for interchange of data and about language

\(^3\) Jena A Semantic Web Framework for Java. \url{http://jena.sourceforge.net/}
for recording how the data relates to real world objects. That allows a person, or a machine, to start off in one database, and then move through an unending set of databases which are connected not by wires but by being about the same thing.

This project is leaded by organizations like the World Wide Web Consortium (W3C) and the Internet Engineering Task Force. Between the contributions of this groups and from the scientific community around this project it can be mentioned: standards like the Universal Resource Identifiers (URI) as global naming convention and Resource Description Framework (RDF) and RDF Schema (RDFS) to express structured vocabulary, repositories of RDF content (triple stores), protocols to extract RDF from XML and XHTML documents (through GRDDL and XSLT), a Web Ontology Language (OWL) that provides greater expressivity in their objects and relation descriptions supporting inference on subsumption and classification, and a Rule Interchange Format (RIF) that supports and interoperates across a variety of rule-based formats[81].

**Ontology Web Language (OWL)**

Ontology Web Language (OWL) is a W3C recommendation [85] for ontologies definition built over the widespread de facto standards XML and RDF. Inspired on the Object Oriented paradigm, OWL has as primitive elements: classes, properties, Instances of classes and relationships between instances.

OWL provides three increasingly expressive sublanguages designed for being used by specific communities of implementers and users: OWL Lite, supporting classification hierarchies and simple constraint features; OWL DL, for maximum expressiveness without losing computational completeness and decidability of reasoning systems; and OWL Full, which provides the maximum expressiveness without computational guarantees. The OWL-DL and OWL-Lite sub-languages of the W3C-endorsed Web Ontology Language (OWL) are based on a description logic.

Classes identifies types of individuals and have certain properties associated to them. Inherence mechanism applies to classes and properties. Individuals are represented as instances of a class and inherence properties associated to the class. Any element in the ontology is identified by an URL, which permits reference other ontologies definitions.

Properties are of two types: datatyped and objects. First uses the XMLSchema data types and second points to instances of certain class. Properties have a range (possible values) and domain (possible classes to be attained to). Properties characteristics that can be expressed are: transitivity, symmetry, functionality and inverse. Some local restrictions can be defined in the class specification such as: cardinality and restriction of values to certain class. The hasValue restriction allows to specify classes based on the existence of particular property values.

Two mechanisms are provided to map classes: equivalence (all instances of class A are instances of class B) and sameness (class A is identical in every sense to class B). Equivalent classes implies a necessary and sufficient condition to classify an instance
as member of a class, meanwhile subclassing only denotes necessary conditions. Two individuals can be declared to be identical or to be different.

Complex classes, called class expressions, can be constructed from the intersection, union or complement of other classes. Additionally classes can be specified via a direct enumeration of its members. And it is possible to assert that class extensions must be disjoint.

SPARQL Query Language

SPARQL[75] is a query language for getting information from RDF graphs. It provides facilities to: extract information in the form of URIs (blank nodes and literals), extract RDF subgraphs, and construct new RDF graphs based on information in the queried graphs.

An SPARQL query consists of two parts: the SELECT clause and the WHERE clause. The SELECT clause identifies the variables to appear in the query results, and the WHERE clause has one triple pattern. Triple Patterns are written as a list of subject, predicate, object.

Variables are preceded by a question mark and serve to establish bindings with RDF terms in a query. Queries represented in Turtle format lets specify a prefix for each namespace. A namespace groups a set of definitions.

Values can be constrained in a SPARQL query using a FILTER condition in the WHERE clause. Variable bindings can be restricted to strings matching a regular expression by using the regex operator. Filter enable to apply relational operators in numerical variables.

Query variables in SPARQL queries have global scope; use of a given variable name anywhere in a query identifies the same variable. Variables are indicated by “?”; the “?” does not form part of the variable name. “$” is an alternative to “?”.

Formally, a SPARQL query contains four components: the graph pattern (GP), the dataset being queried (DS), a set of solution modifiers (SM), and the result form (R). The graph pattern of a query is called the query pattern.

SPARQL has four query result forms. These result forms use the solutions from pattern matching to form result sets or RDF graphs. The query result forms are: SELECT (that returns the variables bound in a query pattern match), CONSTRUCT (that returns a RDF graph constructed by substituting variables in a set of triple templates), DESCRIBE (that returns an RDF graph that describes the resources found), and ASK (that returns a boolean indicating whether a query pattern matches or not).
2.6. MODERN APPROACHES TO CAUSALITY

SPARQL-Update Language

SPARQL-Update is a language for updating RDF graphs that reuses the syntax of the SPARQL query language. Currently it is a W3C member submission\(^4\). It provides the following facilities:

- Insert new triples to an RDF graph
- Delete triples from an RDF graph
- Perform a group of update operations as a single action
- Create a new RDF Graph to a Graph Store
- Delete an RDF graph from a Graph Store

Rule languages for OWL

Semantic Web Rule Language (SWRL)[65] proposes an abstract syntax for Horn-like rules in both the OWL DL and OWL Lite sublanguages of OWL. It counts with an XML syntax based on RuleML\(^5\) and the OWL XML Presentation Syntax as well as an RDF concrete syntax based on the OWL RDF/XML exchange syntax.

Like any production rule, a SWRL rule has an antecedent and a consequent, both of them expressed using concept and role assertions that may contain constants or variables. Besides actual implementations of these language, like Bossam, Hoolet and Pellet, incorporate mathematical, string and logical functions. A rule set is just a set of SWRL rules evaluated non-hierarchically.

Jena\(^6\) provides an implementation of a rule language compatible with SWRL but with an own syntax convention. It also counts with operators that cover the must common functions.

2.6 Modern Approaches to Causality

Causality, banished from the scientific discourse for long time, is now returning to the scene as a useful tool for explaining unobservable events and for explaining processes in a natural way. In this section we present two recent formalism that make use of causality.

CHAPTER 2. BACKGROUND AND RELATED WORK

In one hand we have the nonmonotonic causal logic C+ proposed by Giunchiglia [34], that is the first logic (as far as I know) that compares two world states contiguous in time and uses temporal precedence as a necessary condition for triggering rules. Causal rules expresses causal relations in fact. These formalism is a valuable tool for modeling dynamic processes as long as facilitate us to represent sequences of actions and exogenous events. An extension of C+ called MAD[57] allows to represent action descriptions as extensible modules described in terms of causal rules.

In the other hand we have Bayesian Causal Networks proposed by Judea Pearl [71], which enables representing observable and unobservable events related by causal relations arranged in networks. The introduction of Do calculus let us calculate the causal effect of intervening a variable in the model. Besides, Do Calculus allows to determine the identifiability of sequences of interventions (plans) even in the presence of unobservable variables. Bayesian learning algorithms can be used for training these kind of models.

2.6.1 Nonmonotonic Causal Logic

The Causal Calculator (CCalc) is a system for representing commonsense knowledge about action and change. It implements a fragment of the causal logic C+ [Giunchiglia et al, 2004]. The system is being maintained by the Texas Action Group at Austin\footnote{Texas Action Group at Austin. \url{http://www.cs.utexas.edu/users/tag}}. The semantics of the language of CCalc is related to default logic and logic programming. Computationally, CCalc uses ideas of satisfiability planning. CCalc runs over Prolog and connects to a SAT solver for generating the possible plans.

CCalc Logic

CCalc uses the nonmonotonic causal logic C+ which through formulas and axioms expresses causal rules. This logic advantages to other action description languages by distinguishing between variant and invariant symbols and assuming that nothing changes unless there is a cause for it.

A causal theory in CCalc is constituted by sorts, variables, constants and axioms (causal laws). CCalc allows to express sorts of things identified by a single name. A mechanism of simple inheritance is represented with the operator \( i. \). For example, Car \( i. \) Vehicle. This is the form of expressing that every car is a vehicle. Sorts instances are expressed as objects. For example, myCar :: Car. Variables are typed with the defined sorts and are used in the construction of axioms (rules) to indicate the possible values that can be used in a given predicate.

CCalc define constants instead of predicates. Constants are typed too, allowing not only associating Boolean values to them but objects (instances of sorts) too, which allows expressing the current value of an object property in a given time. Constants
2.6. MODERN APPROACHES TO CAUSALITY

Fluent constants receive any kind of value and can be inertial or rigid, depending on its value is allowed to change or not in time. Nevertheless, the value of fluent constants doesn't change unless there is an axiom indicating so. A fluent formula is a formula such that all constants occurring in it are fluent constants; meanwhile an action formula is a formula that contains at least one action constant and no fluent constants.

CCalc has a single construct for expressing rules or axioms, called causal laws. Causal laws can express static and dynamic laws which relate events occurring in the same time or in consecutive time frames, respectively. Causal laws have the general form caused F if G, where F, G and H are formulas. Through this formalism CCalc can express static laws (if F and G are fluent formulas), action dynamic laws (if F is a fluent formula and G is an action formula), and fluent dynamic laws (if it is added after H to the causal law, being H any formula).

Time in CCalc is expressed explicitly at two levels: in dynamic causal laws (through the after clause) and on the declaration of facts or queries (indicating the time slice, which ranges from 0 to the maxstep variable). CCalc represents a state with a set of instantiated fluent formulas and transitions with events that result of the instantiation of (action or fluent) dynamic causal laws.

CCalc Modeling Capabilities

CCalc provides a set of abbreviations over its general form that synthetically expresses causal axioms. See Table 2.4 for details.

CCalc was originally intended to represent action descriptions, which are expressed through a set of causal laws. MAD\(^8\), described in \cite{LifschitzRen2006}, is an extension of CCalc that allows writing modules for encapsulating actions. MAD modules can be imported in other models and their causal laws are adjusted to the current model during the importation.

In CCalc, an action is described through exogenous, nonexecutable and causes axioms. Exogenous actions denote events coming from exterior, i.e. actions not controlled by the system. Nonexecutable axioms express constraints for executing an action given the current conditions (the opposite of preconditions). Causes axioms describe the effects of the action. Actions effects can be optionally expressed conditionally (if clause) or defeasible (unless clause). Concurrence of actions A and B can be constrained with the nonexecutable axiom. The noconcurrency axiom expresses that only one action can occur at one time.

Fluents are used to express objects properties or attributes, as well as general conditions of the system. Fluents are defined as inertial (if they don't change unless there is a reason for it), or rigid (if they cannot change in time). Default axioms allow expressing assumptions in the absence of causal laws that indicate the current value of a fluent.

\(^8\) Modular Action Description Language (MAD). http://www.cs.utexas.edu/~tag/mad/
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>General form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>caused $F$</td>
<td>$F$ if $T$</td>
<td>$F$ is caused unconditionally.</td>
</tr>
<tr>
<td>caused $F$ after $H$</td>
<td>$F$ if $T$ after $H$</td>
<td>$F$ is caused unconditionally after $H$.</td>
</tr>
<tr>
<td>constraint $F$</td>
<td>$\bot$ if $\neg F$</td>
<td>$F$ cannot occur.</td>
</tr>
<tr>
<td>constraint $F$ after $G$</td>
<td>$\bot$ if $\neg F$ after $G$</td>
<td>$F$ cannot occur after $G$ occurs.</td>
</tr>
<tr>
<td>rigid $c$</td>
<td>$c = v$ if $c \neq v$</td>
<td>The fluent constant $c$ doesn't change of value along time.</td>
</tr>
<tr>
<td>always $F$</td>
<td>$\bot$ after $\neg F$</td>
<td>$F$ must always be true.</td>
</tr>
<tr>
<td>nonexecutable $F$ if $G$</td>
<td>$\bot$ after $F \land G$</td>
<td>The action formula $F$ cannot be executed if $G$ is true.</td>
</tr>
</tbody>
</table>

$F$ causes $G$ if $H$ caused $G$ after $F \land H$  

If $G$ is a fluent formula: Action $F$ causes $G$ when $H$ is true (conditional effect).  

$F$ causes $G$ if $H$ caused $G$ after $F \land H$  

If $G$ is an action formula: Action $F$ causes another action $G$ to occur when $H$ is true (conditional action).  

default $F$ caused $F$ if $F$  

$F$ is true unless the opposite is indicated.  

default $F$ if $G$ caused $F$ if $F \land G$  

$F$ is true conditioned to $G$, unless other value for $F$ is indicated.  

default $F$ if $G$ after $H$ caused $F$ after $F \land G$  

$F$ is true after $H$ conditioned to $G$, unless other value for $F$ is indicated.  

exogenous $c$ caused $c = v$  

Being $c$ an action constant, the action $c = v$ can occur independently of a cause for it.  

or default $c = v$  

default $c = v$ if $G$  

Being $c$ an action constant, the action $c = v$ can occur independently of a cause for it, whenever $G$ is true.  

exogenous $c$ if $G$ caused $c = v$ if $c \land G$  

Being $c$ an action constant, the action $c = v$ can occur independently of a cause for it, whenever $G$ is true.  

or default $c = v$ if $G$  

exogenous $c$ if $G$ caused $c = v$ if $c \land G$  

The effect $G$ (fluent) of action $F$ depends on $H$ being true (nondeterministic actions).  

or default $c = v$ if $G$  

inertial $c$ caused $c = v$ if $c \land G$  

The value of $c$ doesn't change unless there is a cause for it.  

c = $v$  

or default $c = v$ after $c = v \land G$  

inertial $c$ if $G$ caused $c = v$ if $c \land G$  

The value of $c$ doesn't change when $G$ is true, unless there is a cause for it.  

or default $c = v$ after $c = v \land G$  

caused $F$ if $G$ unless $c$ caused $F$ if $G \land \neg c$  

$G$ causes $F$ unless condition $c$ is true.  

and default $\neg c$  

caused $F$ if $G$ after $H \land \neg c$  

and default $\neg c$  

$F$ is caused by action $H$ if $G$ unless action $c$ had occurred.  

Table 2.4: CCalc causal laws abbreviations (axioms).
(given the current or previous conditions). Invariant properties are expressed with the always clause; meanwhile static and dynamic constraints can be expressed with the respective axioms.

In a causal theory can be expressed conditions or actions occurring in a given time step. This is indicated preceding an instantiated formula with the time step followed by colon (:).

CCalc Inference

CCalc inference is performed through queries. Queries definitions consists on a scope (number or range of steps for finding a solution), conditions on the initial state (0:), conditions in the final state (maxstep:) and any action or condition occurring in the middle (N:).

Queries allow performing three types of inference: prediction, postdiction and planning. A prediction query provides an initial state and a sequence of actions, CCalc calculates the conditions after the execution of each action and hence the final state. A postdiction query expresses an incomplete initial state, some actions and the final state, CCalc determines additional initial conditions required for achieving the final state. In a planning query the initial state and the final state are given, CCalc determines a valid sequence of actions for achieving the final state given the initial conditions.

If some condition is not satisfied, i.e. the atom ⊥ is produced by some rule, the theory is not satisfied. Otherwise, CCalc shows the sequence of states and actions occurring in the solution. States are described by instantiated fluent constants with their current values; affirmative Boolean constants are listed meanwhile negated ones don’t. If an action occurs between two states then this is showed between the listing of both states.

### 2.6.2 Bayesian Causal Networks

Pearl defines a causal model as a model that “... encode the truth value of sentences that deal with causal relationships; these include action sentences (e.g., A will be true if we do B), counterfactuals (e.g., A would have been different were it not for B), and plain causal utterances (e.g., A may cause B, or, B occurred because of A). Such sentences cannot be interpreted in standard propositional logic or probability calculus because they deal with changes that occur in the external world rather than with changes in our beliefs about a static world. Causal models encode and distinguish information about external changes through an explicit representation of the mechanism that are altered in such changes.”[71]
CHAPTER 2. BACKGROUND AND RELATED WORK

Functional Causal Models

Judea Pearl proposes a functional specification to express causal models. He considers a causal model constituted by three elements:

$$ M = (U, V, F) $$

(2.3)

where,

- $U$ is a set of background or exogenous variables that are determined by factors outside the model,
- $V$ is a set $\{V_1, V_2, ..., V_n\}$ of endogenous variables that are determined by variables in the model $(V \cup U)$,
- $F$ is a set of functions $\{f_1, f_2, ..., f_n\}$ such that each $f_i$ is mapping from $U \cup (V \setminus V_i)$ to $V_i$ and such that the entire set $F$ forms a mapping from $U$ to $V$.

Then causal model consists of a set of equations of the form:

$$ x_i = f_i(pa_i, u_i), \quad i = 1, \ldots, n, $$

(2.4)

where $pa_i$ (connoting parents) stands for the set of variables judged to be immediate causes of $X_i$ and where the $U_i$ represent errors (or "disturbances") due to omitted factors. Equation (2.4) is a nonlinear, nonparametric generalization of the linear structural equation models (SEMs):

$$ x_i = \sum_{k \neq i} \alpha_{ik} x_k + u_i, \quad i = 1, \ldots, n, $$

(2.5)

A set of equations in the form of (2.4) and in which each equation represents an autonomous mechanism is called a structural model; if each mechanism determines the value of just one distinct variable (called the dependent variable), then the model is called a structural causal model.

A typical specification of a functional causal model is exemplified by the Boolean model below:

$$
\begin{align*}
x_2 &= [(X_1 = \text{winter}) \lor (X_1 = \text{fall}) \lor u_2] \land \neg u'_2, \\
x_3 &= [(X_1 = \text{summer}) \lor (X_1 = \text{spring}) \lor u_3] \land \neg u'_3, \\
x_4 &= (x_2 \lor x_3 \lor u_4) \land \neg u'_4, \\
x_5 &= (x_4 \lor u_5) \land \neg u'_5,
\end{align*}
$$

where $x_i$ stands for $X_i = \text{true}$ and where $u_i$ and $u'_i$ stand for triggering and inhibiting abnormalities, respectively.

---

9 Uppercase letters represent variables or sets of variables, meanwhile lowercase letters represent their values (instantiations).
Graphical Representation

A functional causal model $M$ can be associated with a directed graph, $G(M)$, in which each node corresponds to a variable and the directed edges point from members of $PA_i$ and $U_t$ toward $V_t$. Pearl’s calls it the causal diagram associated to $M$. $G$ constitutes an acyclic directed graph. Figure 2.1(a) shows an example of a causal graph with five endogenous variables drawn with filled circles and two exogenous variables drawn with empty circles. Exogenous variables that affect exactly two endogenous variables can be drawn as bidirected dashed arrows connecting endogenous variables as can be shown in Figure 2.1(b). Can be deduced from Figure 2.1 that $f_4 = (v_2 \lor u_1)$.

![figure 2.1(a)](image)

![figure 2.1(b)](image)

Figure 2.1: Causal Model Graphical Representation (Causal Diagram)

Probabilistic Causal Models

Following Pearl’s definition of causal models, we can use a semi-Markovian model to represent a probabilistic causal model, i.e. a model where some variables are observed and others are not. Probabilistic causal model can be expressed by:

$$M = (V, U, G_{VU}, P(v_i|pa_i, u_i))$$

where $V$ is the set of observed variables, $U$ is the set of unobserved variables, $G_{VU}$ is a causal graph consisting of variables in $V \times U$ and $P(v_i|pa_i, u_i)$ is the probabilistic function of $V_i$ which value depends on the value of its parents ($PA_i$) in the graph and the value of unobserved variables ($U_i$) affecting it. A Markovian causal model is a special case of probabilistic causal models where it doesn’t exist unobserved variables, i.e. $U = \emptyset$.

Graph interpretation has two components, probabilistic and causal. The probabilistic interpretation views the arrows as representing probabilistic dependencies among the corresponding variables, and the missing arrows as representing conditional independence assertions: Each variables is independent of all its non descendants given its
direct parents in the graph. This way we can represent the joint probability function
\[ P(v) = P(v_1, ..., v_n, u_1, ..., u_n) \]
as a combination of products
\[ P(v) = \sum_u \prod_i P(v_i|pa_i, u_i)P(u) \tag{2.7} \]
where \( pa_i \) and \( u_i \) stand for the sets of the observed and unobserved parents of \( V_i \), and
the summation ranges over all the \( U \) variables.

Causal interpretation views the arrows as representing causal influences between linked
variables. In this interpretation, the factorization of (2.7) still holds, but the factors
are further assumed to represent autonomous data-generation processes, that is, each
conditional probability \( P(v_i|pa_i, u_i) \) represents a stochastic process by which the values
of \( V_i \) are chosen in response to the values \( pa_i \) (previously chosen for \( V_i \)'s parents) and a
bias compensation represented by \( u_i \) and encoded in \( P(u) \). The stochastic variation of
this assignment is assumed independent of the variations in all other assignments.

Prediction

The simplest operation on causal models is prediction, which consists on calculate the
\textit{a priori} probability of a set of variables \( Y \), i.e. \( P(y) \).

In a functional specification, values of \( X \) variables will be uniquely determined by those
of the \( U \) values; i.e. the joint distribution \( P(x_1, ..., x_n) \) is determined by the distribution
\( P(U) \) of the error variables.

Intervention

Intervention operation consists on setting the value of a variable or set of variables to a
given value and calculate the probability of the rest of the variables in the new model.
This is made modifying the corresponding equations in the model and using the modified
model to compute a new probability function.

Atomic interventions are performed over a single variable and is equivalent to lifting
\( X_i \) from the influence of the old mechanism \( x_i = f(pa_i, u_i) \) and placing it under the
influence of a new mechanism that sets the value \( x_i \) while keeping all other mechanisms
unperturbed. Pearl represents atomic intervention like \( do(X_i = x_i), do(x_i) \) or \( \hat{x}_i \).
Tian and Maes represent it like \( P_x(y)[87] \).

Causal Effects

A model modified by an intervention \( do(x_i) \) can be solved for the distribution of other
variable \( X_j \), yielding to the notion of causal effect of \( X_i \) on \( X_j \), which is denoted \( P(x_j|\hat{x}_i) \).
When an intervention forces a subset \( X \) of variables to attain fixed values \( x \), then a subset
of equations is pruned from the function causal model, one for each member of \( X \), thus
2.6. MODERN APPROACHES TO CAUSALITY

defining a new distribution over the remaining variables that completely characterizes
the effect of the intervention.

Formally we can define the *causal effect* of $X$ on $Y$, where $X$ and $Y$ are sets of variables,
as a function from $X$ to the space of probability distributions on $Y$. For each realization
$x$ of $X$, $P(y|\hat{x})$ gives the probability of $Y = y$ induced by deleting from the structural
model 2.4 all equations corresponding to variables in $X$ and substituting $X = x$ in the
remaining equations. In the graphical representation this is equivalent to pruning all
arrows entering $X$.

Intervention can be represented as a *modification* of an existing model or as *conditional-
ization* in an augmented model. In both cases its result is a well-defined transformation
between the preintervention and postintervention distributions. In the case of an atomic
intervention $do(x'_i)$, where $x'_i$ represents post-intervention value of $x_i$, this transformation
can be expressed in a simple truncated factorization formula:

$$P(x_1, \ldots, x_n|\hat{x}'_i) = \begin{cases} 
\prod_{j \neq i} P(x_j|pa_i) & \text{if } x_i = x'_i, \\
0 & \text{if } x_i \neq x'_i.
\end{cases} \quad (2.8)$$

Extending intervention to a set of variables $S$ in a semi-Markovian model, the post-
intervention distribution is given by a combination of truncated products:

$$P(x_1, \ldots, x_n|\hat{s}) = \sum_{\{X_i \notin s\}} \prod_{i} P(v_i|pa_i, u_i) P(u) \quad \text{for } x_1, \ldots, x_n \text{ consistent with } s \nonumber$$

otherwise \quad (2.9)

**Causal Effect Identifiability**

The question of identifiability is whether a given causal effect of a given set of variables $X$
on a disjoint set of variables $Y$, $P(y|\hat{x})$, can be determined uniquely from the distribution
$P(v)$ of the observed variables, and is thus independent of the unknown quantities, $P(u)$
and $P(v_i|pa_i, u_i)$, that involve elements of $U$.

The identifiability of $P(y|\hat{x})$ ensures that it is possible to infer the effect of action
$do(X = x)$ on $Y$ from two sources of information:

- passive observations, as summarized by the probability function $P(v)$; and
- the causal graph $G$, which specifies (qualitatively) which variables make up the
  stable mechanisms in the domain or, alternatively, which variables participate in
  the determination of each variable in the domain.

Pearl establishes that given a causal diagram $G$ of any Markovian model in which a
subset $V$ of variables are measured, the causal effect $P(y|\hat{x})$ is identifiable whenever $X$,
$Y$ and all parents of variables in $X$ are measured, that is, whenever $\{X \cup Y \cup PA_X\} \subseteq V$. 
**CHAPTER 2. BACKGROUND AND RELATED WORK**

**Plan Identification**

Pearl characterizes plan identification as the probability of a variable $Y$ given a set of control variables $X$, a set of observed variables $Z$ (often called *covariates*), and a set of unobserved variables $U$. Control variables are ordered $(X = X_1, X_2, ..., X_n)$ such that every $X_k$ is a non-descendant of $X_{k+j}$ ($j > 0$) in $G$ and $Y$ is descendant of $X_n$. $N_k$ is the set of observed nodes that are non-descendants of any element in the set of control variables, i.e. the ancestors of $X$.

A *plan* is an ordered sequence $(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n)$ of value assignments to control variables, where $\hat{x}_k$ means “$X_k$ is set to $x_k$”. A *conditional plan* is an ordered sequence $(g_1(z_1), g_2(z_2), ..., g_n(z_n))$, where each $g_k$ is a function from a set $Z_k$ to $X_k$ and where $g_k(z_k)$ stands for the statement “set $X_k$ to $g_k(z_k)$ whenever $Z_k$ attains the value $z_k$”. The support $Z_k$ of each $g_k(z_k)$ function must not contain any variables that are descendant of $X_k$ in $G$.

Pearl and Robins provide a general criterion for plan identification: The probability $P(y|x_1, x_2, ..., x_n)$ is identifiable if, for every $1 \leq k \leq n$, there exists a set $Z_k$ of covariates satisfying

$$Z_k \subseteq N_k$$  \hspace{1cm} (2.10)

and

$$(Y \perp X_k | X_1, ..., X_{k-1}, Z_1, Z_2, ..., Z_k)_{G_{X_k, X_{k+1}, ..., X_n}}^{10}$$  \hspace{1cm} (2.11)

that is, covariates are ancestors of the control variables and $Y$ is conditionally independent of $X_k$ given the previous actions and their respective covariates.

When these conditions are satisfied, the effect of the plan is given by

$$P(y|\hat{x}_1, \hat{x}_2, ..., \hat{x}_n) = \sum_{z_1, ..., z_n} P(y|z_1, ..., z_n, x_1, ..., x_n) \times \prod_{k=1}^{n} P(z_k|z_1, ..., z_{k-1}, x_1, ..., x_{k-1})$$  \hspace{1cm} (2.12)

Beyond defining the necessity of conditioning on $Z_k$ sets, Pearl and Robins [72] establish an easy procedure for identifying such sets. $W_k$ sets are defined as those covariates in $G$ that are both non-descendants of $X_k$ and have either $Y$ or $X_k$ as descendant. In this way, the condition for *G-identifiability* is reformulated as

$$(Y \perp X_k | X_1, ..., X_{k-1}, W_1, W_2, ..., W_k)_{G_{X_k, X_{k+1}, ..., X_n}}$$  \hspace{1cm} (2.13)

for every $1 \leq k \leq n$. In consequence the plan evaluates to

$$P(y|\hat{x}_1, \hat{x}_2, ..., \hat{x}_n) = \sum_{w_1, ..., w_n} P(y|w_1, ..., w_n, x_1, ..., x_n) \times \prod_{k=1}^{n} P(w_k|w_1, ..., w_{k-1}, x_1, ..., x_{k-1})$$  \hspace{1cm} (2.14)

Pearl and Robins also prove that in their procedure for plan identifiability, $X_k$ not necessarily must be ancestor of $X_{k+j}$.

---

$^{10}$ $G_X$ denotes the graph obtained by deleting from $G$ all arrows emerging from nodes in $X$, $G_X$ denotes the graph obtained by deleting from $G$ all arrows pointing to nodes in $X$. 

---
2.7 Modern approaches to Metaphysics and Causality

Description Logics and Bayesian networks have met in some modern approaches. Next we mention some of them.

2.7.1 Probabilistic Description Logics

Probabilistic Description Logics have been developed mainly as variations of classical Description Logics by introducing a probabilistic component to represent uncertainty on individuals classification.

P-Classic


\[ P-Classic \] is a probabilistic version of the description logic \textit{Classic}. In order to express uncertainty it uses a bayesian network whose random variables are the basic properties of an individual, the number of fillers for its roles, and the properties of these fillers.

A \textit{P-Classic} knowledge base includes a number of different \textit{p-classes} or \textit{probabilistic classes}, each of which is a bayesian network over basic properties, the number of R-fillers, and the p-classes from which the role fillers are chosen. A p-class is interpreted as a probability distribution over the elements in the domain.

\textit{P-Classic} authors provide an inference procedure for probabilistic subsumption that consists on computing the probability that a random individual in class \textit{C} is also in class \textit{D}. This algorithm implements a form of lifted inference - reasoning at the level of variables rather than at the level of ground terms. If bayesian networks used to express p-classes are restricted to polynomial time (e.g. polytrees), the complexity of this algorithm is also polynomial time.

P-SHOQ(D)

\[ P-SHOQ(D) \] is a probabilistic extension of \textit{SHOQ(D)} designed to deal with probabilistic ontologies in the Semantic Web. \textit{P-SHOQ(D)} is based on the notion of probabilistic lexicographic entailment for probabilistic default reasoning. \textit{P-SHOQ(D)} allows to express default knowledge as a special case of generic probabilistic knowledge.

Every probabilistic terminology consists of a generic part, which expresses generic classical and probabilistic knowledge about concepts, and an assertional part, which represents classical and probabilistic knowledge about a set of individuals. For example, individuals are partitioned into \textit{classical individuals} and \textit{probabilistic individuals}. \textit{Conditional constraints} are expressed in the form \((D|C)[l, u]\) indicating that the concept
$D$ subsumes the concept $C$ with a probability in $[l, u]$. On a similar way is expressed probabilistic knowledge that indicates the relation of a concept and an individual given a role, the probability of an individual to belong to a class, and the relation between a probabilistic individual and a classical individual given a role.

Authors complements a classical interpretation with a probability function w.r.t. the set of concrete datatypes $D$. They present sound and complete techniques for probabilistic reasoning in $P$-SHOQ($D$), based on reductions to classical reasoning in SHOQ($D$) and to linear programming.

**Labeled Bayesian Networks**

Yelland proposes in [96] a framework that combines bayesian networks and description logics in order to classify individuals on a given class. For this purpose he uses a Tiny Description Logic (TDL) that resembles $\mathcal{FL}$ and a Labeled Bayesian Network (LBN).

The LBN comprises: a DAG with nodes $1, ..., n$, an indexed collection of disjoints labels sets $L_1, ..., L_n$ such that $L_i$ is a subset of atomic TDL expressions, and a conditional probability $p(L|L_1, ..., L_m)$ for each node $i$.

The nodes of a labeled Bayesian network represent collections of disjoint sets of objects denoted by atomic expressions in TDL. The conditional probabilities in such a network assert the probability that an object selected at random will be a member of a given set, given that it is a member of certain other sets.

The conditional probability of the (general) concept $C_1$ given (general) $C_2$ is denote by $\lambda(C_1|C_2)$ and is defined as the ratio of the joint probability of $C_1$ and $C_2$ to the probability of $C_2$.

**2.7.2 Multi-Entity Bayesian Networks**

In [53], Kathryn Laskey proposes a first-order language for specifying probabilistic knowledge bases as parameterized fragments of Bayesian networks. MEBN fragments (MFrags) can be instantiated and combined to form arbitrarily complex graphical probability models. An MFragment represents probabilistic relationships among a conceptually meaningful group of uncertain hypotheses.

The semantics of MEBN assigns a probability distribution over interpretations of an associated classical first-order theory on a finite or countably infinite domain. MEBN exploits the capabilities of Bayesian inference for combining prior knowledge with observations, and for refining a model as evidence accrues. MEBN can be used for representing a probability distribution on interpretations of any finitely axiomatizable first-order theory.

The MEBN language treats the world as being comprised of entities that have attributes and are related to other entities. Constant and variable symbols are used to refer to
2.8. INTELLIGENT AGENTS

entities.

An MFrag $F = (C, I, R, G, D)$ consists of a finite set $C$ of context value assignment terms; a finite set $I$ of input random variable terms; a finite set $R$ of resident random variable terms; a fragment graph $G$; and a set $D$ of local distributions, one for each member of $R$. The sets $C$, $I$, and $R$ are pairwise disjoint. The fragment graph $G$ is an acyclic directed graph whose nodes are in one-to-one correspondence with the random variables in $I \cup R$, such that random variables in $I$ correspond to root nodes in $G$. Local distributions specify conditional probability distributions for the resident random variables.

MEBN allows to express the meaning of a Bayesian network like the one shown in Figure 2.2. The meaning of this Bayesian network is expressed using MEBN fragments as illustrated in Figure 2.3.

![Figure 2.2: Bayesian Network for a Diagnostic Task](image)

2.8 Intelligent Agents

From a broad point of view, an agent can be defined as something that can perceive its environment through sensors and that responds and acts in such environment through effectors [80]. Another definition is "... a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives" [91].

In addition to this definition, an intelligent agent must satisfy some extra requisites, such as reactivity (agent should respond to stimuli on time giving an effective response),
proactivity (agent presents goal driven behavior and takes actions to reach its objectives), and sociability (intelligent agents are capable of interact with other agents to satisfy its objectives).

Several architectures have been proposed for formalizing the notion of intelligent agent. For their relevance to our work we selected some of them. In the first place, we present SOAR, the Newell's computational implementation of those notions introduced by himself and Simon[83]. Then we describe the Belief-Desire-Intention (BDI) architecture proposed by Rao and Georgeff [76] and inspired in the Bratman’s theory of intention. Finally we present two implementations of a goal-driven BDI architecture that incorporates goals and plans in the BDI architecture.

2.8.1 SOAR

SOAR[66] is a Unified Theory of Cognition (UTC) proposed by Newell in 1990. SOAR is based on the Means-Ends Analysis technique. Newell proposes a framework for intelligent agents capable of supporting multiple applications with a single agent architecture. This architecture is composed by:

**States and operators.** States contains information of the current state, including perceptions, goals and problem spaces. Operators are the mean for moving along the problem space.
2.8. INTELLIGENT AGENTS

**Long Term Memory (LTM).** Repository for domain content stored in three representation forms: procedural, semantic and episodic.

**Working Memory (WM).** Contains perceptions and the hierarchy of states and their associated operators; its content trigger retrieval from LTM and motor actions.

**Perception/Motor interfaces.** Allows gathering information from the environment and intervene on it.

**Decision cycle.** It's the core of the operator selection and application machine. It has three phases: elaboration (elaborate state, suggest new operators and evaluate them), quiescence (end of elaboration and beginning of decision), and decision (select an operator or detect an impasse).

**Impasses.** Situations produced by lack of knowledge that avoids the agent to make an operator selection, solved by gathering knowledge from the LTM.

**Learning mechanisms.** Chunking (learning from conditions present in an impasse), reinforcement learning, episodic learning and semantic learning.

Basically the agent construct the state with current beliefs and perceptions in the working memory (WM). WM contains rules that may request retrieval of more rules from the LTM. With the rules in WM and current world state a decision tree is built with all possible operators. Goal states are marked in the tree. Each node denotes a possible world represented by features and values. The problem is reduced to a search in the built problem space. If there is an impasse or there is no knowledge enough to select an operator, the LTM is searched for help. The architecture makes hierarchical inference by mapping a node to a new decision tree; the inference mechanism determines that it must solve first inner problems and then the main problem. The last is known as the automatic generation of subgoals in SOAR.

### 2.8.2 The BDI Architecture

*Belief-Desire-Intention* (BDI) agent architectures are based on practical reasoning, i.e. the process of deciding, moment by moment, which action to perform in the furtherance of our goals. Figure 2.4 illustrates the process of decision making in this architecture[91].

Beliefs represent information the agent has about its current environment, desires are the options available to the agent, and intentions represent agent’s current focus - those states of affairs that it has committed to trying to bring about. This three elements represent the internal state of an agent and guides him in the decision making process. Next we’ll summarize this mechanism.

First, agent receive perceptions that updates its current set of beliefs. According to the intentions the agent have, a set of options are generated from the beliefs. This options must be consistent with agent’s current beliefs and current intentions, and opportunistic recognizing the environment changes to offer the agent new ways of achieving intentions.
The filter function updates the agent’s intentions on the basis of its previously-held intentions and current beliefs and desires and represents the agent’s deliberation process. In this function intentions no longer achievable or very expensive are dropped and new intentions that help to achieve existing intentions or exploit new opportunities are adopted.

Finally the agent chooses which intention to achieve and transform this intention into an action.

### 2.8.3 Goal-driven BDI Implementations

Goal-driven BDI architectures like JAM[47] and JADEX[10] incorporate the notion of goal in the the Belief-Desire-Intention inference model. These architectures use a library of plans designed for achieving specific goals and use current beliefs and goals to generate the set of possible options in an event-based schedule.

#### JAM

A JAM agent[47] may have a possibly large number of alternative plans for accomplishing any single goal and the JAM interpreter reasons about all of the alternatives combinations of plans, goals, and variable bindings (based on goal arguments and be-
2.8. INTELLIGENT AGENTS

liefs) before selecting the best alternative given the particular situation.

The agent checks all the plans that can be applied to a goal to make sure they are relevant to the current situation. Those plans that are applicable are collected into what is called the Applicable Plan List (or APL). A utility value is determined for each instantiated plan in the APL and, if no metalevel plans are available to select between the APL elements, the JAM interpreter selects the highest utility instantiated plan (called an intention) and intends it to the goal.

If the goal with the new intention has the highest utility among all goals with intentions, then the new goals plan is executed. otherwise, a previous intention still has a higher utility and the interpreter executes that intentions plan. If generation of an APL results in one or more entries, the agent possibly enters into metalevel reasoning. That is, it reasons about how to decide which of the APL elements to intend to its intention structure.

A JAM agent is specified with a list of top level goals. This list of goals can be augmented during execution through communication with other agents, generated from internal reasoning on the part of the agent, or by many other means. Top level goals are persistent. That is, they are pursued until they are satisfied by successful plan execution or opportunistically by some other means, such as another agent, or are removed explicitly within a plan. Subgoals are not persistent by default. If a plan fails for a subgoal, the interpreter considers the subgoaling action to have failed (just as if it were any other type of plan action).

JAM supports a number of different types of goals, ACHIEVE, PERFORM, and MAINTAIN each with distinct semantics. An ACHIEVE goal specifies that the agent desires to achieve a goal state and is the goal type typically associated with BDI architectures and generative planning systems. JAM agents continually monitors for goal achievement. Typically, the plan selected for the subgoal will be the means by which the subgoal is accomplished. However, if the agent detects opportunistic accomplishment of the goal, perhaps by another agent, it will consider the subgoal action successful and discontinue execution of the plan. Finally, a world model entry indicating that the goal has been achieved is asserted if the plan selected for accomplishment of the ACHIEVE subgoal completes successfully. The world model entry that is asserted is the goal specification for the goal just achieved.

A PERFORM goal specifies the agent desires to perform some behavior. The agent does not check to see whether the goal has already been accomplished before selecting plans to perform the behavior. The agent does not monitor for goal achievement during plan execution and will execute the intended plan until the plan succeeds or fails. Finally, an assertion to the world model entry that the goal has been achieved is only performed if the plan that was executed has an ACHIEVE goal specification and it completes successfully.

A MAINTAIN goal indicates that the specified goal must be reattained if it ever becomes unsatisfied. A MAINTAIN goal is similar to an ACHIEVE goal except that a
CHAPTER 2. BACKGROUND AND RELATED WORK

MAINTAIN goal is never removed from the agents goal list automatically (i.e., it is a homeostatic goal). A MAINTAIN goal can be removed from the agents intention structure only by the agent explicitly removing it.

A JAM plan defines a procedural specification for accomplishing a goal, reacting to an event, or performing behavior. JAM agents are therefore capable of both goal-driven and data-driven behavior. A plans applicability is limited to either a particular goal or a data-driven conclusion. Each plan may be further constrained to a particular precondition, conditions that must hold before starting execution of the plan, and context, conditions that must hold both before and during execution of the plan.

The procedure to use to accomplish the goal is given in the plans procedural body, which can contain simple actions (e.g., execute a user-defined primitive function) and complex structured constructs (e.g., iteration and equivalents to if-then-else). Each plan may include an explicitly or implicitly defined utility calculation, which is used to influence selection of certain procedures over others through the default utility-based metalevel reasoning mechanism of JAM.

Another optional component is the effects field, which is a procedure that the JAM interpreter executes when the plan completes successfully. An agent programmer can use the effects field to perform World Model updating. A procedural specification of what the agent should do when a plan fails can be represented in a plans optional failure section.

JADEX

Jadex[73] is a Java based, FIPA compliant agent environment, and allows to develop goal oriented agents following the BDI model. Jadex provides a framework and a set of development tools to simplify the creation and testing of agents.

Jadex introduces beliefs, goals and plans as first class objects, that can be created an manipulated inside a JADE agent. In Jadex, agents have beliefs, which can be any kind of Java object and are stored in a belief base.

Goals are implicit or explicit descriptions of states to be achieved. To achieve its goals the agent executes plans. In Jadex, goals are concrete, momentary desires of an agent. Jadex does not assume that all adopted goals need to be consistent to each other. To distinguish between just adopted (i.e. desired) goals and actively pursued goals, a goal lifecycle is introduced which consists of the goal states option, active, and suspended. When the context of a goal is invalid it will be suspended until the context is valid again.

Jadex supports four types of goals: perform, achieve, query and maintain. The first three types are similar to those implemented in JAM. A query goal is an achieve goal which outcome is not defined as a state of the world, but as some information the agent wants to know about.

Plans are plain Java classes, extending a specific abstract class, which provides useful
methods e.g. for sending messages, dispatching sub goals or waiting for events. Plans are able to read and alter the beliefs of the agent using the API of the belief base. In addition to directly retrieving stored facts, an Object Query Language (OQL) allows to formulate arbitrary complex expressions using the objects contained in the belief base.

The developer provides an XML based Agent Definition File (ADF), which specifies the initial beliefs, goals, and plans of an agent. The Jadex runtime engine reads this file to instantiate an agent model, and executes the agent by keeping track of its goals while continuously selecting and executing plan steps.

2.9 Multiagent Systems

A Multi-Agent System (MAS) is a system composed of multiple interacting intelligent agents. Multi-agent systems can be used to solve problems which are difficult or impossible for an individual agent or monolithic system to solve. Even when agents situated in a multi-agent system must cooperate in order to achieve a global goal, they maintain at least a partial autonomy and make decisions based on their local view of the world [94]. Multi-agent systems can manifest self-organization and complex behaviors even when the individual strategies of all their agents are simple [69].

2.9.1 Multi-Agent Systems Methodologies

Development and maintenance of a Multi-Agent System is a complex task. Next we present two methodologies that have been proposed in this sense: the Gaia and the Prometheus methodologies. These methodologies generate different models that guide the specification of the problem and its decomposition in functionality manageable by agents.

The GAIA Methodology

Gaia[95] is a methodology for agent-oriented analysis and design. This methodology deals with both the macro-level (societal) and the micro-level (agent) aspects of systems. Gaia is founded on the view of a multi-agent system as a computational organization consisting of various interacting roles.

The Gaia process starts with the analysis phase, whose aim is to collect and organize the specification which is the basis for the design of the computational organization. This includes the identification of:

- The goals of the organizations that constitute the overall system and their expected global behavior.
CHAPTER 2. BACKGROUND AND RELATED WORK

- The *environmental model*; intended as an abstract, computational representation of the environment in which the MAS will be situated.

- The *preliminary roles model*; identifying the basic skills required by the organization.

- The *preliminary interaction model*; identifying the basic interactions required to accomplish the preliminary roles.

- The *rules* that the organization should respect and enforce in its global behavior.

The output of the analysis phase is exploited by the design phase, which can be logically decomposed into an architectural design phase and a detailed design phase. The *architectural design phase* includes:

- The definition of the systems *organizational structure* in terms of its topology and control regime.

- The completion of the preliminary *role and interaction models*. This is based upon the adopted organizational structure and involves separating the organizational-independent aspects and the organizational-dependent ones. This demarcation promotes a design-for-change perspective by separating the structure of the system (derived from a contingent choice) from its goals (derived from a general characterization).

The *detailed design phase* covers:

- The definition of the *agent model*. This identifies the agent classes that will make up the system and the agent instances that will be instantiated from these classes. There may be a one-to-one correspondence between roles and agent types, although a number of closely related roles can be mapped into the same agent class for the purposes of convenience or efficiency. Inheritance is not considered in Gaias agent models.

- The definition of the *services model*. This identifies the main services that are required to realize the agents roles, and their properties.

The Gaia methodology doesn't commit to specific techniques for modeling nor deals directly with implementation issues. Models generated by this methodology can be implemented using any agent platform that considers the mentioned elements.

**The Prometheus Methodology**

The Prometheus methodology [68] focuses on the development of intelligent agents rather than black boxes, supports software engineering activities from requirements
2.9. MULTIAGENT SYSTEMS

specification through to detailed design and implementation, provides detailed processes (not just artifacts and notations), has evolved out of practical industrial and pedagogical experience, has been used by people other than the developers of the methodology, and supports (automatable) cross checking, hierarchical structuring mechanisms and an iterative process.

The Prometheus methodology consists of three phases, shown schematically in Figure 2.5:

- **System Specification**: where the system is specified using goals and scenarios; the systems interface to its environment is described in terms of actions, percepts and external data; and functionalities are defined.

- **Architectural design**: where agent types are identified; the systems overall structure is captured in a system overview diagram; and scenarios are developed into interaction protocols.

- **Detailed design**: where the details of each agents internals are developed and defined in terms of capabilities, data, events and plans; process diagrams are used as a stepping stone between interaction protocols and plans.

Each of these phases includes models that focus on the dynamics of the system, graphical models that focus on the structure of the system or its components, and textual descriptor forms that provide the details for individual entities.
System specification begins with a rough idea of the system, which defines the requirements of the system in terms of goals, case use scenarios, functionalities, and the interface with the environment (actions and percepts). Goals, given as high-level descriptions, are decomposed into subgoals and illustrated in diagrams showing their dependencies. Then goals are grouped in functionalities. A functionality encompasses a number of related goals, percepts that are relevant to it, actions that it performs, and data that it uses.

Use case scenarios are a detailed description of one particular example sequence of events associated with achieving a particular goal, or with responding to a particular event. Scenarios are described using a name, description, and a triggering event. The core of the scenario is a sequence of steps. Each step consists of the functionality that performs that step, the name of the step, its type (one of ACTION, PERCEPT, GOAL, SCENARIO or OTHER) and, optionally, the information used and produced by that step. In addition, scenarios often briefly indicate variations.

Finally, the environment within which the agent system will be situated is defined. This is done by describing the percepts available to the system, the actions that it will be able to perform, as well as any external data that is available and any external bodies of code.

In the architectural design phase the focus is on:

- Deciding on the agent types in the system: where agent types are identified by grouping functionalities based on considerations of coupling; and these are explored using a coupling diagram and an agent acquaintance diagram. Once a grouping is chosen the resulting agents are described using agent descriptors.

- Describing the interactions between agents using interaction diagrams and interaction protocols: where interaction diagrams are derived from use case scenarios; and these are then revised and generalized to produce interaction protocols (Agent-UML [9]).

- Designing the overall system structure: where the overall structure of the agent system is defined and documented using a system overview diagram. This diagram captures the agent types in the system, the boundaries of the system and its interfaces in terms of actions and percepts, but also in terms of data and code that is external to the system.

A useful tool for suggesting groupings of functionalities is the data coupling diagram. This depicts each functionality (as a rectangle) and each data repository (as a data symbol) showing where functionalities read and write data.

Another technique that is useful in comparing the coupling of different alternatives is the use of agent acquaintance diagrams. An agent acquaintance diagram shows the agent types and the communication pathways between them. Agent acquaintance diagrams provide a convenient visualization of the coupling between the agent types: the higher the link density, the higher the coupling.
2.9. MULTIAGENT SYSTEMS

The *system overview diagram* shows agents, percepts, actions, messages, and external data as nodes. Each of these node types has its own distinct visual depiction. Directed arrows between nodes indicate messages being sent and received by agents, actions being performed by agents, percepts being received by agents, and data being read and written by agents.

**Detailed design** consists of:

- Developing the internals of agents, in terms of capabilities (and, in some cases directly in terms of events, plans and data). This is done using agent overview diagrams and capability descriptors.

- Develop process diagrams from interaction protocols.

- Develop the details of capabilities in terms of other capabilities as well as events, plans and data. This is done using capability overview diagrams and various descriptors. A key focus is developing plan sets to achieve goals and ensuring appropriate coverage.

*Capabilities* are a structuring mechanism similar to modules. A capability can contain plans, data, and events. It can also contain other capabilities allowing for a hierarchical structure.

The structure of each agent is depicted by an *agent overview diagram*. This is similar to the system overview diagram except that it does not contain agent or protocol nodes. However, the agent overview diagram does usually contain capability nodes and sometimes plan nodes.

The design of each agent is, usually, in terms of capabilities and is finally expressed in terms of plans, events and data. Prometheus assume that the agents are implemented using a platform that supports plans which are triggered by goals.

Prometheus counts with a Design Tool (PDT) which allows users to create and modify Prometheus designs. It ensures that certain inconsistencies cannot be introduced and provides cross checking that detects other forms of inconsistency. The tool can also export individual design diagrams as well as generate a report that contains the complete design.

Another tool that supports the Prometheus methodology is the JACK Development Environment (JDE) which provides a design tool that allows Prometheus-style overview diagrams to be drawn. The JDE can then generate skeleton code from these diagrams.

### 2.9.2 Multi-Agent Systems Frameworks

Multiple MAS frameworks have been developed with research purposes, and some of them have reached such a degree of maturity that has allowed implementing industrial scale solutions. Here we present two of them: Electronic Institutions and Madkit.
We conclude presenting a commercial platform that takes advantage of several MAS approaches demonstrating the feasibility of MAS approaches for developing complex and industrial-scale applications.

**Electronic Institutions**

The Electronic Institutions framework [5] developed in the Artificial Intelligence Institute of the Spanish National Scientific Research Council, (IIIA-CSIC), is a means to design and implement regulated open multiagent systems. The EI framework may be described in terms of a conceptual model, a computational model and a software platform, EIDE, to specify and run electronic institutions.

The **conceptual model** for electronic institutions assumes that the electronic institution determines a virtual space where agents interact subject to explicit conventions, so that institutional interactions and their effects count as facts in the real world. Because of this virtuality, it is assumed that all interactions within the electronic institution are *speech acts* expressed as illocutionary formulae. The electronic institution defines an *open MAS* in the sense that (i) it makes no assumption about the architecture and goals of participating agents (who may be human or software entities); and (ii) agents may enter and leave the institution at will, as long as the regimented conventions of the institution are met. Participating agents are subject to *role-based regulations* whose specification is given in terms of illocutions, norms and protocols. There are two classes of agents, internal and external. Internal agents act on behalf of the institution itself who is responsible for their behavior. External agents act on their own behalf and their internal state is inaccessible to the institution. Interactions are organized as repetitive activities called *scenes*. *Scenes* establish interaction protocols describing agent group meetings as transition diagrams whose arcs are labeled by valid illocutions. The *performative structure* captures the relationships among scenes describing those transitions agents playing certain role can make. Finally, *normative rules* describe the obligations an agent contracts while it participates in the institution. Agents may move from one scene to another, they may be active in more than one scene at a given time and they may perform different roles in different scenes.

The **computational model** for EIs defines a social (institutional) software layer between an agent communication platform (e.g. JADE) and participating agents. All *institutional communications* among agents are mediated by this platform. That institutional middleware is composed by three types of *infrastructure "agents"*: (i) An *institution manager* who centralizes valid communications and keeps track of the *state* of the institution, which is a data structure that contains the current values of all variables involved in the enactment of the institution. (ii) There are *scene* and *transition managers* for each, scene and transition, who handle the activation and persistence of scenes and transitions, and give access and exit to participants according to the local conventions; these managers mediate between the institution manager and the agent governors and keep track of the state of the institution as it applies to their particular context. (iii) One *governor* is attached to each agent and filters all communications.
between that agent and the institution; in particular, it directs valid illocutions to the corresponding scene managers and the institutional manager. The governor keeps a copy of the evolving state of the institution in order to apply regimented conventions on all speech acts its agent utters, and communicates to other infrastructure agents only those speech acts that comply with those conventions; thus the governor enables a change of the institutional state if and only if it admits a valid illocution from its agent.

The Electronic Institutions Development Environment, EIDE [25], consists of a graphical specification language, ISLANDER, whose output is an XML specification of an institution; a middleware AMELI [26], that takes an XML specification and enacts a runtime version of the institution with agents who run on a FIPA-compatible agent communication platform; a debugging and monitoring tool, SIMDEI, that registers all communications to and fro AMELI and displays and traces the evolution of the institutional state; finally an agent-shell builder, ABuilder, that from the XML specification produces an agent “skeleton” for each agent role. The skeleton satisfies all the (uninstantiated) navigation and communication requirements of the specification thus leaving the agent programmer to deal only with the implementation of the agent’s decision-making logic at communication points.

MadKit/AGR

MadKit\footnote{MADKIT. http://www.madkit.org/} is a modular and scalable multiagent platform written in Java and built upon the AGR (Agent/Group/Role) organizational model [28]: agents are situated in groups and play roles.

MadKit allows high heterogeneity in agent architectures and communication languages, and various customizations. MadKit communication is based on a peer to peer mechanism, and allows developers to quickly develop distributed applications using multiagent principles. Agents in MadKit may be programmed in Java, Scheme (Kawa), Jess (rule based engine) or BeanShell. MadKit comes with a full set of facilities and agents for launching, displaying, developing and monitoring agents and organisations.

AGR [28] is based on three core concepts: agent, group and role. An agent is only specified as an active communicating entity which plays roles within groups. Groups are atomic sets of agent aggregation. Each agent is part of one or more (possibly overlapped) groups.

A group can be founded by any agent, and an agent must request its admission to an existing group. Groups might be distributed among several machines. The role is an abstract representation of an agent function, service or identification within a group. Each agent can handle several roles, and each role handled by an agent is local to a group. As with group admission, handling a role in a group must be requested by the candidate agent, and is not necessarily awarded.

A special role in a group is the group manager role, which is automatically awarded to
the group creator. It has responsibility for handling requests for group admission or role requests. It can also revoke roles or group membership.

The organizational model is specified in AGR through a set of additional concepts illustrated in Figure 2.6.

The group structure is an abstract description of a group. It identifies all the roles and interactions that can appear within a group and is represented by a finite set of roles identifiers \((R)\), a graph with valid interaction between roles \((G)\), and an interaction language \((L)\) used for labeling interactions.

The organizational structure is a set of group structures expressing the design of a multi-agent organization scheme. It can be seen as the overall specification of the initial problem. It is represented by a set of valid structures \((S)\) and a representative graph \((Rep)\) that indicates which roles can be played simultaneously in different groups.

**Whitestein Technologies**

Whitestein Technologies\(^{12}\) is one of the first companies specialized in software development based on agents. Its advantage in the software development arena is its offer of providing self-adaptability on its products, which they translated into the optimization of processes and infrastructures in real-time.

Whitestein developed the Agent Modeling Language (AML)\(^{13}\), is based in UML 2 and augmented with several modeling concepts for multi-agent systems. It is designed to

---


support business modeling, requirements specification analysis, and design of software systems based on software agent concepts and principles. AML models are characterized by:

- consisting of a number of autonomous, concurrent and/or asynchronous (possibly proactive) entities,

- comprising entities that are able to observe and/or interact with their environment,

- making use of complex interactions and aggregated services,

- employing social structures, and

- capturing mental characteristics of systems and/or their parts.

AML considers as architectural elements: agents, resources, the environment, social aspects, deployment configurations, and ontologies. Agents are considered objects endowed with autonomy and the ability to interact. A resource is a physical or an informational entity, with which the main concern is its availability and usage. Environmental objects are logical or physical surroundings of entities which provide conditions under which the entities exist and function; these objects can be internal or external to the system. Social aspects include social properties, associations and roles.

In AML, Behaviors can be basic or composite, and refers to: communicative interactions, services, observations and effecting interactions, and mobility. Communicative interactions include messages and protocol definitions. In observations and effecting interactions there are defined types of perceptors and effectors, as well as the actions and acts of perceiving (sensing) and effecting (acting).

Mental states are also modeled in AML, considering: beliefs, goals, plans and mental relationships. Beliefs are used to model a state of affairs, proposition or other information relevant to the system and its mental model. It is possible to specify attributes and/or operations for a Belief, like the reliability or confidence on a it. Goals are modeled as conditions or states of affairs, with which the main concern is their achievement or maintenance. The type of goals in AML are: maintain, achieve, provide, or avoid. Plans may represent predefined plans to achieve goals, or fragments of behavior from which plans can be composed.

Besides, AML counts with several diagrams that enables a visual modeling of the application. UML class diagrams are extended for representing the ontology, mental states, the society, behavior decomposition, and goal-based requirements. UML structure diagrams are used for representing entities and services. UML sequence and communication diagrams are used for describing protocols.
2.10 Summary

I presented several definitions that Knowledge Management (KM) has coined for Organizational Intelligence (OI) [35, 86]. Other related definitions presented are Learning Organizations [12, 61] and Corporate Memories [55].

Then I introduced some philosophical foundations [4] on which my theory is inspired. Metaphysics let us generalize every real or mental being in the notion of entity, and characterize the type of properties (accidents) they can show. Causality, in the other hand, let us talk about change and the type of causes that intervene on it.

Next I presented some modern computational approaches that brings about such notions described in the aforementioned philosophical concepts. Ontologies [37] are modern versions of metaphysical definitions, but instead of representing general categories of beings [30, 36, 23, 77], these are used for representing particular application domains. Besides, ontologies are supported by the family of Description Logics (DL) that provide very rich constructors for expressing complex concepts and definitions [7].

The nonmonotonic causal logic C+ and Bayesian Causal Networks use the causality theory for making assumptions that simplifies the representation of rules and the inference on probabilistic models. C+ [34] compares two world states contiguous in time and uses temporal precedence as a necessary condition for triggering rules. Bayesian Causal networks [71] represent observable and unobservable events related by causal relations. The Do Calculus introduced by Pearl allows to determine the identifiability of sequences of interventions (plans) even in the presence of unobservable variables.

Other hybrid approaches are presented. In one way, probabilistic Description Logics have been developed mainly as variations of classical Descriptions Logics by introducing a probabilistic component to represent uncertainty on individuals classification [50, 33]. In the other, Multi-entity Bayesian Networks (MEBN) [53] proposes a first-order language for specifying probabilistic knowledge bases as parameterized fragments of Bayesian networks. MEBN fragments are instantiated and combined to form arbitrarily complex graphical probability models.

Then I commented some computational implementations of intelligent agents. In the first place we describe the main elements of SOAR, inspired in a unified theory of cognition [66]. Then I presented the Belief-Desire-Intention architecture based in the Bratman’s theory of intention [76]. Finally I described some implementation details of two Goal-driven BDI architectures, JAM [47] and JADEX [73].

Next I presented some Multi-Agent Systems methodologies and frameworks. The Gaia [95] and the Prometheus methodologies [68] provide models at macro and micro level for the analysis, design and implementation of systems based on agents. The Electronic Institution [5] and the Madkit [28] frameworks provide the means for designing and implementing regulated open multiagent systems. Finally, I presented the company Whitestein Technologies which capitalizes agent technologies for providing optimization of processes and infrastructures in real-time.
I conclude this chapter presenting some comments of Simon on the design of artificial intelligence [83] and presented the Autonomic Computing approach for developing intelligent systems [48].
Chapter 3

A Causal Ontological Framework

The current application of ontologies is centered in the modeling of domains and classification of entities. In order to design and create intelligent entities we reinterpret DL constructors for expressing entity capabilities and the minimal conditions they require for acting according to their ontological definition. In this way, entity definitions can be used by other entities for creating or building artifacts compliant with such definitions.

Between these artifacts we can find intelligent entities which behavior is given by their capabilities, the goals they pursue and the internal representation of the world they have, as well as the interfaces they have with the world. I recur to the notion of causal relations for expressing changes in the internal world representation owned by an intelligent entity derived from its own actions, exogenous events or logical inferences. In the same way, I use the notion of causal chains to express the intentionality in an action for pursuing a given goal.

3.1 A Causal Classification of Entities

I propose a basic classification of entities that reflects the main cause categories proposed by Aristotle. Figure 3.1 shows the taxonomy used for organizing the entities defined in a specific domain. In order to formalize this taxonomy through DL, we built an OWL ontology where the given categories were represented by classes and were arranged through subsumption.

These categories can be used for defining entities of a specific domain using the DL subsumption mechanism between classes. Making every taxonomy class disjoint with each one of its siblings we can check through a DL reasoner that a domain class belongs to a single category.

The class Agent represents efficient causes, the class Goal represents final causes, the class Material represents material causes, the class Form represents formal causes and the class Action represents types of actions.
The class Information is used for representing mental entities, i.e. patterns among entities and literals representing agent beliefs. On the other hand, the class Knowledge is used for abstracting complex structures (e.g. rules) or procedures which details are not relevant for the decision process of agents that use the ontology.

As long as entities are classified as causes of actions performed in the domain, their definition is made with the purpose of representing events and actions occurring in a domain. Actions defined in the ontology will be arranged in plans intended for goals also defined in the ontology. In this sense, we can say that the resulting ontology is domain dependent and purpose oriented.

### 3.2 Essential Forms

*Designing* an artifact has for output an specification that is used for the construction of the artifact itself. In metaphysics, this specification is considered a form that certain individual (agent) can issue on certain materials in order to create an entity with the characteristics described in the specification. Realist philosophers would say that the new entity have this form for essence. This position is valid in the context of creation of artifacts.

Other philosophical currents have points of view that are valid in other contexts. For example, nominalists argue that an object is given the name of the category that best matches its description and in case that it doesn’t match any of the known categories a new category can be created. This position is valid in tasks like classification on which we only know category descriptions and objects of unknown origin. The nominalist perspective can be used by an artifact when tries to figure out which kind of object is dealing with in order to treat it according to the categories it knows.

In both cases, there is a name for the essence or category and a description of the objects issuing the form or classified on the category. In the creation perspective the description
tell us the characteristics the created entity must have; whereas on the classification perspective the description is used to decide if the object satisfy the constraints given for the category. In other words, the order of these notions for design purposes is essence \(\rightarrow\) description \(\rightarrow\) entity, whereas for classification purposes the order of precedence is object \(\rightarrow\) description \(\rightarrow\) category. Likewise, both approaches uses taxonomies, one for specializing essences and the other for grouping categories, respectively. In both creation and classification it is necessary that the individual knows the essences or categories before performing the task of creation or classification.

Modern ontologies formalisms and reasoners developed so far on Description Logics (DL) have been driven by classification purposes mainly. Nevertheless, formalisms like Frame systems and semantic networks, from which Description Logics evolved, were originally intended for knowledge representation purposes. The objective was to create the categories abstracting relevant aspects from the observed objects.

In DL representations, categories are called classes and their descriptions are given in terms of constraints over the properties or relations that objects might show. These constraints are expressed in terms of concept and role constructors indicating the properties and relations that individuals should show and the values they should have in order to be classified on a given category or class.

In order to use DL formalisms for design and initialization purposes we selected a subset of concept and role constructors and assigned an appropriate interpretation in both perspectives. Table 3.1 shows the interpretations given to DL concept constructors on the design and initialization of individuals of a given type (essence). Classes (or types) are denoted by \(C\), properties or relations by \(R\), and individuals or constants by \(a\).

Role constructors are used for inferring implicit relations or properties from explicit relations or properties asserted on individuals. During design such constructors are used to represent common sense information; in plain words, for saying the same thing from several ways. Inferred relations or properties are used on initialization to validate that the new individual satisfy the given definition. For instance, having an inverse role constructor for role \(R\) on \(S\), denoted \(R^{-1}S\), would cause that a constraint like \(\exists R.C\) being equivalent to \(\exists S^{-1}.C\).

The DL family formed by the selected concept constructors is named \(SHOTQ\). Its upper bound on complexity for checking concept satisfiability and ABox consistency is \(N\text{ExpTime}\)-complete. These concept constructors serve to define a kind of concept with the double purpose of design and initialization on creation.

**Definition 1 (Essential Form).** An essential form is a DL concept expressed through subsumption relations \((C \sqsubseteq D)\) that uses exclusively those constructors shown in Table 3.1 and includes at least one relation \(C \sqsubseteq K\) where \(K\) is a class representing a category of the causal taxonomy (Fig. 3.1). The symbol used for identifying the new concept \((C)\) is referred as the name of the essential form.

DL roles are used for representing: entity properties, relations between entities, and the execution of actions. Whereas entity properties can be represented by object or
### Table 3.1: Design and creation interpretations for DL concept constructors

<table>
<thead>
<tr>
<th>DL Constructor</th>
<th>DL Syntax</th>
<th>Design interpretation</th>
<th>Creation interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value restriction</td>
<td>$\forall R.C$</td>
<td>Restrict associating only individuals or constants of type $C$ through $R$.</td>
<td>--</td>
</tr>
<tr>
<td>Limited existential quantification</td>
<td>$\exists R.T$</td>
<td>It must have some relation or property $R$ of any type.</td>
<td>Set at least one constant or entity through $R$ having compatible types with $R$.</td>
</tr>
<tr>
<td>Existential quantification</td>
<td>$\exists R.C$</td>
<td>It must have some relation or property $R$ with individuals of type $C$.</td>
<td>Set at least one constant or entity of type $C$ through $R$.</td>
</tr>
<tr>
<td>Intersection</td>
<td>$C \cap D$</td>
<td>It must inherit the common characteristics of $C$ and $D$.</td>
<td>Set the common initializations of $C$ and $D$.</td>
</tr>
<tr>
<td>Union</td>
<td>$C \cup D$</td>
<td>It must inherit all the characteristics of $C$ and $D$.</td>
<td>Set all the initializations of $C$ and $D$.</td>
</tr>
<tr>
<td>Unqualified number restriction</td>
<td>$\geq nR.T$</td>
<td>It must have (at least $n$ / at most $n$ / exactly $n$) $R$ properties or relations.</td>
<td>Set (at least $n$ / at most $n$ / exactly $n$) constants or entities through $R$ having compatible types with $R$.</td>
</tr>
<tr>
<td></td>
<td>$\leq nR.T$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$= nR.T$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qualified number restriction</td>
<td>$\geq nR.C$</td>
<td>It must have (at least $n$ / at most $n$ / exactly $n$) $R$ properties or relations.</td>
<td>Set (at least $n$ / at most $n$ / exactly $n$) constants or entities of type $C$ through $R$.</td>
</tr>
<tr>
<td></td>
<td>$\leq nR.C$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$= nR.C$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nominal</td>
<td>$R = {a}$</td>
<td>It is assumed that $R = a$ by default.</td>
<td>Set $R = a$.</td>
</tr>
<tr>
<td>One of</td>
<td>$R = {a_1, ..., a_n}$</td>
<td>Property or relation $R$ must be set to one value of ${a_1, ..., a_n}$.</td>
<td>Set $R = a_1 \lor ... \lor R = a_n$.</td>
</tr>
</tbody>
</table>
3.3 Entities

Datatyped roles, relations are represented by object roles only. In the case of actions, it is defined a special role do with range on the class Action, i.e. do.Action.

Potential properties, relations and actions are represented in essential forms through (un)qualified at-least restrictions with \( n = 0 \), i.e. \( \geq OR.T \) or \( \geq OR.C \). This indicates that an individual issuing this form can show the accident \( R \). The RDFS language has a more adequate representation through the domain relation between a role and a concept. Nevertheless, this relation is not used in DL formalisms. Value restrictions and (un)qualified number restrictions with \( n \neq 0 \) can be used too for expressing potential properties.

Finally we can mention two advantages of using DL concepts for representing entities. The first is the possibility of extending the specification in both directions, top-down or bottom-up, by specializing or generalizing concepts, respectively; in terms of DL, it consists on declaring subclasses of certain concept or declaring a superclass for a set of concepts. These mechanisms allow controlling the level of detail in the specification.

The other advantage is that we can make abstract or ambiguous definitions of entities, similarly to Java interfaces. These abstract definitions produce multiple interpretations of individuals of the class. Dealing with abstract definitions can be done on one of two ways: subsuming the concept and specifying additional constraints that produce a single interpretation, or letting to choose among the multiple configurations produced for every possible interpretation.

### 3.3 Entities

An individual must be created indicating a single essential form as its type. Additionally the essential form might produce multiple configurations from which one must be selected. This configuration will produce a set of initializations that will be performed during the instantiation or creation process. Initializations might be of the form: i) set \( R = a \), or ii) create \( I_2 \) of type \( C \) and set \( R = I_2 \). In this way, the creation operation is formalized as follows.

**Definition 2** (Creation action). The creation action receives as input parameters an essential form and a configuration containing a (possible empty) list of initializations. Its output is an entity description describing the created object. It is denoted as \( \text{create}(\text{form}, \text{config}) : \text{entity} \).

The actual description of an entity is made through the notion of DL individuals, which uses an URI for identifying the individual and a set of role assertions describing its actual relations and properties. The actual execution of an action \( A \) by the agent \( Ag \) is represented by the assertions \{do(Ag, \alpha), A(\alpha)\}, where \( \alpha \) represents the actual execution of the action \( A \).

The entity description can be incorporated in an assertional box (ABox) for checking its compliance with the essential form definition.
Definition 3 (Entity representation). In an ABox $\mathcal{A}$, an entity is represented by a DL individual $e$ that has an essential form $E$ associated through the role $\text{rdf:type}$ and that is consistent with the definition of $E$, i.e. $\mathcal{A} \models_E e$.

### 3.4 World Representation

Metaphysics propose that a representation of the world can be given in terms of entity descriptions. The world representation is owned by some entity with capabilities for sensing the environment and for storing such representation. Given the limitations of this entity, the world representation can be inaccurate and approximated relaying on the precision of its sensors and the information sources it can access.

According to previous definitions, we can say that an assertional box (ABox) can be used for representing a static state of the world at certain point in time. Given the capabilities of DL formalizations for inferring implicit information we can differentiate between two world state representations.

**Definition 4** (World State). A world state is represented by an ABox containing the descriptions of a set of entities at certain point in time. It is denoted $W^t$ where $t$ represents a point in time.

**Definition 5** (Extended World State). An extended world state is the ABox resulting of expanding a world representation using a given set of definitions (TBox). It is denoted $W^*$ where $t$ represents a point in time.

$W^*$ might contain multiple types for an individual representing an entity, given its actual accidents and the set of definitions specified in the domain TBox. Nevertheless, the essential form of the entity is the one explicitly given in $W^t$.

### 3.5 Change and Causal Relations

A world state denotes a static view of a system. In order to consider dynamism we introduce the notion of change.

**Definition 6** (Change). Change is the difference between two world states, $W^{t_1}$ and $W^{t_2}$, where the first precedes temporarily the second, denoted $t_1 < t_2$. A change is denoted by $C = (\Delta^+, \Delta^-)$, where $\Delta^+ = W^{t_2} \setminus W^{t_1}$ and $\Delta^- = W^{t_1} \setminus W^{t_2}$.

$\Delta^+$ identifies the set of statements added on $W^{t_2}$ whereas $\Delta^-$ represents the set of statements removed. Change can be calculated between any two world states as long as one state precedes the another, i.e. change identification doesn’t impose a restriction on temporal contiguity.
3.5. CHANGE AND CAUSAL RELATIONS

Statements added or removed can be interpreted as modifications, deletion or creation of entities. An entity is created during this change if there is no reference to the entity on $W^{t_1}$ and $\Delta^+$ contains the definition of such entity. On the opposite way, an entity is deleted (or consumed) during change if its description is contained on $W^{t_1}$ but is not contained in $W^{t_2}$, i.e. $\Delta^-$ contains the definition of such entity. If the entity description is contained on both $W^{t_1}$ and $W^{t_2}$ but the set of accidents representing the entity in both world states differ, we say that the entity was modified during change. If $W^{t_1}$ and $W^{t_2}$ are exactly the same, i.e. $\Delta^+ = \Delta^- = \emptyset$, we say there is no change at all.

When a change occurs between $W^{t_1}$ and $W^{t_2}$, its causes are present on $W^{t_1}$ and its effects are subsets of $\Delta^+$ and $\Delta^-$. Causes are represented by entities classified on one of the categories given in section 3.1. The existence of causes on $W^{t_1}$ and partial descriptions of them can be identified as necessary conditions for the change to occur. Causal entities can be consumed or modified during change. Similarly, new entities can be created as consequence of it. Created and modified entities are considered to be recipient of the effect of this change, and are called caused entities and modified entities, respectively.

The hard task of isolating the causes and their partial description that enable a change is not addressed in this research. Instead we limit to provide a representation for associating causes and effects on the form of causal relations. In first term we selected DL conjunctive queries for representing conditions in our framework.

**Definition 7 (Condition).** A condition is represented by $Q = (V, q)$ where $q$ represents the set of tuples of a conjunctive query and $V = \{v_1, ..., v_n\}$ is a list of distinguished variables. If $\text{Vars}(q)$ identifies the set of variables in $q$, then $V \subseteq \text{Vars}(q)$.

Given a condition $Q = (V, q)$ and a world state $W$, the evaluation of $q$ on $W$ returns a (possibly empty) set of tuples $T = \{T_1, ..., T_m\}$ such that $T_i = (V_1 = v_1, ..., V_n = v_n)$ and $V_i \in V$. If $T = \emptyset$ it is said that $W \not\models q$; otherwise, we say that $W \models q$. Each tuple $T_i$ is considered an occurrence of $Q$ on $W$. The respective replacement of variables on $T_i$ produces a submodel of $W$, a set of tuples, describing the isolated event represented by $Q$.

Next we adapt the definition of causal relation to our framework.

**Definition 8 (Causal Relation).** A causal relation is represented by $R = (\Lambda, \delta^+, \delta^-, \nu)$ where $\Lambda = (V, q)$ is a condition considered necessary and sufficient (causes) on $W^{t_1}$ for producing the insertion and deletion of statements (effects), denoted $\delta^+$ and $\delta^-$ respectively, on a $W^{t_2}$ such that $0 \leq \nu \leq (t_2 - t_1)$.

In strict sense, $W^{t_1}$ should precede temporarily to $W^{t_2}$. Nevertheless, all the changes occurred in the time interval passed during the updating of the world state are represented simultaneously on $W^{t_1}$, i.e. $\nu \approx 0$.

A causal relation $R = (\Lambda, \delta^+, \delta^-, \nu)$ explains entirely a change $C = (\Delta^+, \Delta^-)$ if $\Delta^+ = \delta^+$ and $\Delta^- = \delta^-$. Similarly, we say that $R$ explains partially $C$ if $\Delta^+ \subseteq \delta^+$ and $\Delta^- \subseteq \delta^-$. The regularity of $R$ is proved if whenever $W^{t_1} \models \Lambda$, $\delta^+$ is added and $\delta^-$ is removed to/from a $W^{t_2}$ such that $t_2 - t_1 \leq \nu$. We say that $R$ was brought about on $W^{t_1}$. 
Let us use $effect(R)$ to represent the absolute observable effect produced by a causal relation $R$. $effect(R)$ is represented by a condition $Q = (V_e, q_e)$ such that $q_e = q_\Lambda \cup \delta^+ \setminus \delta^-$ and $V_e = V_\Lambda \cup \text{Vars}(\delta^+)$. 

The observable effect of $R$ over a world state $W$ is denoted $effectOn(R, W)$. Its calculation starts by evaluating $\Lambda$ on $W$. The set of occurrences $T = \{T_1, \ldots, T_m\}$ obtained from this evaluation are used for applying the transformations denoted by $\delta^+$ and $\delta^-$ on $R$. Each $T_i$ is replaced on the transformations obtaining $\delta^+_i$ and $\delta^-_i$ respectively, and then is made the transformation $W_{i+1} = W_i \cup \delta^+_i \setminus \delta^-_i$. $W_1$ represents the initial world $W$ and $W_{m+1}$ is the world state that shows the accumulated consequences of the occurrences of $R$ on $W$.

### 3.6 Actions

A change explained by a causal relation involve the participation of one or more entities considered causes of it. These entities are identified on $\Lambda$ by concept assertions like $C(I)^1$ where $C$ is a subclass of some class of the taxonomy illustrated on Figure 3.1.

In some causal relations it is possible to identify the entity initiating actively the change. Even further, it can be identified the activity or action it initiates the change. The entity initiating the change, considered the efficient cause on the causal relation, can be an agent or a material entity. The action that initiates the change is identified by an action role assertion $\{do(E, \alpha), \alphaType(\alpha)\} \in \Lambda$ where $E$ is the identifier of the efficient cause, $\alphaType$ is the type of action and $\alpha$ represents the act. We call action causal relation to those causal relations containing an action role assertion of this kind.

Roles with domain on action types are used for identifying entities considered causes or effects of the given action. Effects of the action are identified through the role causes, which is constrained to indicate the potential type and number of entities created as consequence of the action execution. Similarly, the role efficientCause is used to identify the efficient cause of the action, i.e. the type of agent that performs or initiates the action. Additionally, custom roles are defined and constrained for representing other action causes.

We can illustrate the definition of actions and the attribution of such action to agents with an example. A sell action on which an agent sells some goods to another by an amount of money can be defined as follows:

$$
\begin{align*}
SellAction & \subseteq Action \sqcap = 1 \text{ seller.Agent} \sqcap = 1 \text{ buyer.Agent} \\
SellAction & \sqsupseteq 1 \text{ soldGood.T} \sqcap = 1 \text{ soldFor.Float} \sqcap = 1 \text{ causes.Shipping}
\end{align*}
$$

Properties seller and buyer are used to identify the role that an agent plays in the sale. The property soldGood identifies the entities sold by the seller whereas the datatyped

\footnote{In the following, when we express that some concept or role assertion is contained on $\Lambda$ we mean that such assertion is contained in the set of statements denoted by $q$.}
property \textit{soldFor} is used for representing (through a float literal) the money given by the buyer in exchange. The action description specifies as well that a shipping will be generated as consequence of the sale. The class \textit{Shipping} represents a mental entity containing information of the good’s delivery. As we will show later, the conditions on which the action can be performed and changes produced by the action are represented through one or more rules.

Finally, for indicating that the agent type \textit{Ag} is capable of performing the \textit{sell} action we assert the following:

\[ \textit{Ag} \sqsubseteq \geq 0 \text{do.SellAction} \]

The effects of an action \( \alpha \) can be described by multiple causal relations. The set of causal relations describing the effects of an action are identified by the presence of the action role assertion on \( \Lambda \). Each one of these relations can describe variations of the initial conditions or variations on the produced effects. Variations on the initial conditions are represented by statements on \( \Lambda \), whereas variations on the effects are represented by statements on \( \delta^+ \) and \( \delta^- \). In this way we could, for example, represent counterfactual conditions (\textit{unless} ...) that determine a different outcome on the presence of an inhibitor. We could even represent the effects of performing two actions simultaneously by including on \( \Lambda \) the respective action role assertions representing them.

In other cases the efficient cause or its action is not observable and it is enough to register the regularity of the change to foresee its consequences. Let us call \textit{natural causal relation} to the causal relation that describe this kind of change. An example of this kind of changes is the decomposition of an apple produced by interactions at cellular level not observable at plain sight. Nevertheless we can notice the difference on the characteristics of that apple across time. Another type of unobservable actions are the mental operations that we perform unintentionally (e.g. classifying an observed object) or those corporal movements made instinctively (e.g. breathing). Even though those actions are performed by us, or by some part of us, we ourself or someone else can perceive their effects.

### 3.6.1 Action Types

Finally we classify actions in terms of the time they last as \textit{instantaneous} or \textit{long-term}. Instantaneous actions, also known as \textit{atomic}, require all the attention of an entity and are so short that the effect of its execution is considered immediate. Long-term actions, in the other hand, take place for a period of time that cover a set of instants perceived by an agent. Each instant is represented by a world state in which the action role assertion appears. A long-term action involves a set of operations which can be perceived by the entity itself but not by other entities. For this reason, we can refer to them as \textit{composite actions}. Likewise, a composite action can be perceived by two or more entities involved on the action execution; this is the case of communication protocols between two agents. For external observers, the action role and the causal relation associated to the composite action describe only its observable effects.
3.7 Causal Chains and Indirect effects

In our framework, a composite action is described by a chain of causal relations, i.e. a sequence of causal relations where the effects of one sets, completely or partially, the causes for the next. Chained causal relations can be action or natural causal relations. A composite action may contain another composite action on its description.

A causal relation $R_1$ is said to set entirely the conditions for $R_2$ if $\Lambda_2 \subseteq \text{effect}(R_1)$, i.e. $\Lambda_2$ is contained in $\text{effect}(R_1)$. Similarly, $R_1$ is said capable of partially setting the conditions of $R_2$ if $\Lambda_2 \cap \text{effect}(R_1) \neq \emptyset$, i.e. $\Lambda_2$ and $\text{effect}(R_1)$ intersects. If a $R_1$ partially sets the conditions for $R_2$ it will be necessary that the missing conditions, $\Lambda_2 \setminus \text{effect}(R_1)$, being set. A missing condition might represent, for example, a specific time that must be waited for continuing with $R_2$. Missing conditions can be met in a dynamic world by exogenous events or by actions initiated by some of the participants on $R_2$.

**Definition 9 (Causal Chain).** A causal chain is constituted by a sequence of causal relations $R_i, R_n$ such that $\Lambda_{i+1} \subseteq \text{effect}(R_i)$ or $\Lambda_{i+1} \cap \text{effect}(R_i) \neq \emptyset$, for $1 < i < n$. A causal chain is denoted $\omega = \{R_1, ..., R_n\}$.

Meanwhile a causal relation $R$ identify the direct effect over a world state $W$ through $\text{effectOn}(R, W)$, identifying indirect effects requires the notion of causal chain. In a causal chain $\omega$, $R_i$ enables, totally or partially, to $R_{i+1}$, in turn $R_{i+1}$ enables $R_{i+2}$ and so on. In this way we can say that $R_i$ enables indirectly $R_{i+j}$ for $j > 1$. If $R_i$ is controlled by the execution of an action (represented by the insertion of an action role assertion), we can say that $\text{effect}(R_{i+j})$ is an indirect effect of the action $\alpha \in R_i$.

Causal chains are used for representing plans. For the causal chain $\omega = \{R_1, ..., R_n\}$, $\Lambda_1$ represents the necessary conditions for starting the execution of the plan and $\text{effect}(R_n)$ describes the condition that is tried to be achieved through the execution of the plan.

**Definition 10 (Plan).** A plan is represented by $P = (\omega, A, \Omega)$ where $\omega = \{R_1, ..., R_n\}$, $A = \Lambda_1$ and $\Omega = \text{effect}(R_n)$.

For indicating that the action $\alphaType$ is a composite action described by a plan $P = (\omega, A, \Omega)$ we assert $\alphaType \subseteq \text{describedBy} = P$.

3.8 Final causes and intentionality

In a broad sense, the final cause proposed by Aristotle identifies not only a future condition desired by the agent, but the intention for changing the current state of affairs. In current approaches, goals are the must close notion to intentions (in the Aristotelian sense). In our framework, rather than using a single condition for expressing goals we'll use two conditions: one for identifying aspects (entities and accidents) of interest in the
current state of the world, and another for expressing the desired state of such entities in a future world state.

**Definition 11 (Goal).** A goal is represented by \( G = \langle A, \Omega \rangle \) where \( A \) is a condition that expresses a current or past state of affairs and \( \Omega \) denotes the condition to achieve in a future world state. The class Goal is the ontological representation of the \( G \) tuple.

Nevertheless, we can find useful to express a goal that is not conditioned to having an initial state.

**Definition 12 (Unconditional Goal).** An unconditional goal is a goal \( G = \langle A, \Omega \rangle \) where \( A = T \), i.e. it’s a goal where the desired state is not conditioned to an initial condition.

As can be seen, conditions used for describing the identification and achievement of goals are expressed as DL conjunctive queries. In consequence we can use variables on its construction for representing undetermined entities or values. Goals containing variables, distinguished or not, are considered abstract.

An abstract goal can produce one or more goals through a process called goal instantiation, which consists on identifying a set of individuals/values that matches the precon­dition pattern in the current world state \( W \). The resulting goal is called goal instance or actual goal.

**Definition 13 (Goal Instance).** A goal \( G_2 = \langle A_2, \Omega_2 \rangle \) is considered instance of goal \( G = \langle A, \Omega \rangle \) if \( A = A_2, \Omega = \Omega_2 \) and there is a set of realizations \( p = \{(V_i, v_i) | V_i \in V_A \text{ for } 0 < i \leq n\} \) such that \( W \models p, A \). The goal instance is denoted \( G_2 = \langle G, p \rangle \) for short.

Instantiation of the abstract goal \( G \) over different world states might produce similar goal instances \( G_1 = \langle G, p_1 \rangle \) and \( G_2 = \langle G, p_2 \rangle \), over \( W_1 \) and \( W_2 \) respectively. We say that \( G_1 \sqsubseteq G \) and \( G_2 \sqsubseteq G \).

A goal represents the desire of an intentional entity for changing its world, the world it can perceive. We’ll use the property \( \text{pursues}(\textit{Goal}) \) to denote the intention of an entity for achieving certain goal. An agent \( A \) pursuing the goal \( G \) at world state \( W^t \) is denoted by \( W^t \models \text{pursues}(A, G) \).

A plan \( P = \langle \omega, A_P, \Omega_P \rangle \) is said to achieve goal \( G = \langle A_G, \Omega_G \rangle \) if \( \Omega_P \sqsubseteq \Omega_G \). If the intentional entity \( Ag \) pursues \( G \) and \( W^t \models A_P, \text{Ag} \) can choose to initiate \( P \)'s execution in order to achieve \( G \). Any action \( \alpha \in R_i \) will intend to \( G \) as long as an indirect effect of \( R_i \) on \( \Omega \) is identified through \( \omega \). We can say that the final cause of \( \alpha \) is \( G \).

The intention of agent \( Ag \) for pursuing the goal \( G \) can vary along time. For this reason, the representation \( W^t \models \text{pursues}(A, G) \) seems to be convenient for representing intentionality. Asserting \( \text{pursues}(A, G) \) will denote the acquisition of the intention and respectively removing this assertion will indicate the cease of such intention.
3.9 Summary

I presented an ontological framework that uses Description Logic (DL) formalisms for providing intelligent entities specifications. A DL sub-language is used for expressing the capabilities of an entity and the minimal conditions they require for acting according to their ontological definition. Entities are classified according to the four main types of causes that according to Aristotle intervene in any change. This classification aligns ontological definitions with the purpose of the domain as long as entities are modeled according to their participation in actions or events that lead to the achievement of a global goal.

Assertional boxes (ABoxes) are used for providing a static representation of the world, whereas change is expressed in terms of causal relations that distinguish differences between two world states. Actions are defined statically in terms of the type and number of entities representing their causes and effects. In the other hand, preconditions and consequences of actions are modeled dynamically through causal relations. Chains of causal relations allows to identify the intention of intelligent entities in the actions they perform.
Chapter 4

Semantic Causal Models

I adapt current causal approaches like the nonmonotonic causal logic C+ [34] and Bayesian Causal Networks [71] to our static world representation (ABoxes). The former is used for representing single causal relations whereas the latter allows to represent multiple causal relations chained towards a goal.

Semantic causal rules are used to express changes in the world state derived from entities’ actions, exogenous events and logical inferences. Semantic causal networks are used to identify and isolate the causes and actions that intervene in a process aligned towards the achievement of a goal. Occurrence of events and actions identified in a causal network are grouped in cases which are used for learning through observation.

4.1 Causal Rules

I use the definition 8 for proposing a rule representation inspired in a DL action formalism [6] and compatible with rule languages for RDF. The original representation $R = (\Lambda, \delta^+, \delta^-, \nu)$ is simplified assuming that $\nu \geq 0$ and removing $\nu$ from the representation.

Definition 14 (Causal Rule). A causal rule is represented by $R = (\Lambda, \delta^+, \delta^-)$, where $\Lambda$ denotes the necessary and sufficient conditions that must hold in world state $W_1$ for executing the rule, where $\delta^+$ and $\delta^-$ indicates the statements asserted and removed, respectively, to/from the world state $W_1$ and that can be observed on world state $W_2$.

Causal rules describing the effects of an action are considered as an special case.

Definition 15 (Action Rule). An action rule describing the effects of an action $\alphaType$ is represented by a causal rule $R_\alpha = (\Lambda, \alpha, \delta^+, \delta^-)$ such that $\Lambda$ contains the statements $\{do(E, \alpha), \alphaType(\alpha)\}$ and the effects of the action are observed on a posterior world state ($\nu > 0$).
The essential form \( e \) of the action's efficient cause \( E \) is represented by the concept assertion \( e(E) \) included in or inferred from \( \Lambda \). If the efficient cause is an intentional entity, \( \Lambda \) represents necessary, but not sufficient, conditions, i.e. the agent can choose to perform \( \alpha \) or not. The internal conditions that motivate \( E \) to act (pursued goals) are not expressed on the action rule. On the other hand, \( \Lambda \) represents necessary and sufficient conditions for \( \alpha \) when the efficient cause is a resource or material object; i.e. the object acts as a reactive artifact.

The same formalism can be used for representing inertial changes associated to natural causal relations. Such rules are used for expressing changes in the beliefs of an agent.

**Definition 16 (Immediate Rule).** An *immediate rule*, represented by \( R_I = (\Lambda, \delta^+, \delta^-) \), describes immediate changes occurring in world state \( W_1 \) such that \( W_1 \models \Lambda \) and \( \delta^+ \) and \( \delta^- \) are applied on \( W_1 \) \((\nu = 0)\).

Finally, the causal rule is used to represent changes produced by the occurrence of exogenous events perceived by the entity maintaining the world state. An agent can use this kind of rules for incorporating in its beliefs information coming from perceptions.

**Definition 17 (Exogenous Rule).** An *exogenous rule* is represented by \( R_E = (\Lambda, \Lambda', \delta^+, \delta^-) \) where \( \Lambda' \) represents necessary conditions occurring on an external ABox \( \mathcal{A} \) that are incorporated immediately \((\nu = 0)\) on \( W_1 \) whenever \( W_1 \models \Lambda \) and \( \mathcal{A} \models \Lambda' \). \( \delta^+ \) and \( \delta^- \) contain variables referenced in both \( \Lambda \) and \( \Lambda' \).

Given that the change is produced outside, the exogenous condition \( \mathcal{A} \models \Lambda' \) is evaluated first. If any occurrence of the exogenous event is detected, the values obtained from each occurrence \( T_i \) are replaced on \( \Lambda \) on the correspondent variable names; we denote the modified \( \Lambda \) using \( \Lambda_i \). Each \( \Lambda_i \) is evaluated on \( W \) and each one of its occurrence is combined with the original occurrence \( T_i \) producing a \( T'_i \) that is used for asserting or removing predicates to/from \( W \).

### 4.1.1 Representation in SWRL

Causal rules proposed in this section can be implemented using SPARQL-Update (see Section 2.5.4). The causal rule \( R = (\Lambda, \delta^+, \delta^-) \) is represented by:

\[
\text{DELETE} \{ \delta^- \} \quad \text{INSERT} \{ \delta^+ \} \quad \text{WHERE} \{ \Lambda \} \quad (4.1)
\]

An exogenous rule \( R_E = (\Lambda, \Lambda', \delta^+, \delta^-) \) is evaluated in two steps. In the first place, a SPARQL SELECT query is used for checking \( \mathcal{A} \models_B \Lambda' \):

\[
\text{SELECT} \ast \quad \text{WHERE} \{ \Lambda' \} \quad (4.2)
\]

For every binding set \( B_i \in B \) it is executed the SPARQL-Update query 4.1 replacing the values of \( B_i \) where it corresponds. The Jena API provides a method for using such \( B_i \) in any SPARQL query.
4.2. THE SEMANTIC BAYESIAN CAUSAL MODEL

The variable \textit{self} is used for representing the agent executing the rule. This variable can be expressed explicitly in A or can only be present in $\delta^+$ or $\delta^-$. For this reason, the binding \textit{self} = \textit{agentUri} is included on each SPARQL query execution.

4.2 The Semantic Bayesian Causal Model

The Semantic Bayesian Causal Model, SBCM for short, integrates a semantic layer to a Bayesian Causal Network, allowing to express semantically the meaning of random variables [17]. Recurring to definitions 2.6 and 2.12 we characterize the kind of Bayesian causal models used.

**Definition 18** (MCBCM). A Markovian Controllable Bayesian Causal Model (MCBCM) is represented by:

$$m = (V, G_V, P(v_i|pa_i), X, Z)$$  \hspace{1cm} (4.3)

where $V$ is the set of endogenous variables, $G_V$ is a directed acyclic causal graph consisting of variables in $V \times V$, $P(v_i|pa_i)$ is the conditional probabilistic distribution of variables, $X \subseteq V$ are boolean variables representing endogenous variables that can be manipulated (control variables), and $Z \subseteq V$ is the subset of endogenous variables that cannot be manipulated (covariates), such that $X \cap Z = \emptyset$.

Each variable $V_i$ in the model, and the values that can hold the variable ($v_{ij} \in dom(V_i)$), have an implicit meaning encoded in the process that generates the data used to build the model. In order to make explicit this meaning we use the notion of DL knowledge bases and conjunctive queries (see Section 2.5.3).

I will consider the realization of a variable ($V_i = v_{ij}$) as the minimum meaningful unit of a causal model. In order to express the meaning of a realization with respect to semantic definitions expressed in a TBox $T$, it is formalized the notion of \textit{semantic descriptor}.

**Definition 19** (Semantic Descriptor). The meaning of the realization $V_i = v_{ij}$, where $V_i \in V$ is a variable of the MCBCM $m = (V, G_V, P(v_i|pa_i), X, Z)$ and $v_{ij} \in dom(V_i)$, is represented by a conjunctive query $Q_{ij}$ expressed using roles and concepts defined in a TBox $T$, denoted $D_T(V_i, v_{ij}, Q_{ij})$.

The set of descriptors that describe the possible values of the variable $V_i$ constitute the description of a variable in a model, denoted $D_T(V_i)$. This set of descriptors must satisfy a set of constraints to be considered consistent.

**Definition 20** (Random Variable’s Description). A finite and ordered list of semantic descriptors $D_T(V_i) = \{D_T(V_i, v_{ij}, Q_{ij})|v_{ij} \in Dom(V_i), 0 < l < n\}$ constitute a well-formed description of the variable $V_i$ if and only if:

1. exists at least one $D_T(V_i, v_{ij}, Q_{ij})$ for each $v_{ij} \in Dom(V_i)$ where $Q_{ij} \neq \bot$, 


2. given any two descriptors $D^1_T(V_i, v_{i1}, Q_{ij1})$ and $D^2_T(V_i, v_{i2}, Q_{ij2})$ in $D_T(V_i)$ such that $v_{i1} \neq v_{i2}$, then $Q_{ij1}$ and $Q_{ij2}$ are mutually exclusive, i.e. $Q_{ij1} \not\subseteq Q_{ij2}$ and $Q_{ij2} \not\subseteq Q_{ij1}$, and

3. given any two descriptors $D^1_T(V_i, v_i, Q_{ij1})$ and $D^2_T(V_i, v_i, Q_{ij2})$ in $D_T(V_i)$ such that $Q_{ij1} \subseteq Q_{ij2}$, $D^1_T$ must be listed before $D^2_T$ in $D_T(V_i)$.

The first condition guaranties that a variable can be set to any value of its domain from facts stored in an ABox. The second satisfy the constraint of Bayesian networks which establishes that a random variable $V_i \in V$ represents a single event for which a realization $V_i = v_{ij}$ is mutually exclusive with respect to each other realization of the same variable. Finally, the third condition warrants that the most specific semantic descriptor is evaluated first. The set of descriptors that completely describes all the variables $V_i \in V$ is denoted $D_{TV}$.

Now it is formally defined the SBCM.

**Definition 21 (SBCM).** A Semantic Bayesian Causal Model, represented

$$\mathcal{M} = (m, T, D_{TV}, C, F), \quad (4.4)$$

is constituted by a MCBCM $m = (V, G_V, P(v_i, p_a), X, Z)$, a TBox $T$, a set of semantic descriptors $D_{TV}$ describing all the variables in $V$ w.r.t. $T$, $C$ is a non-empty set of realizations in $Z$ that identifies the context on which the model occurs, and $F$ is a non-empty set of realizations in $Z$ that identifies desired final states.

Let use $\text{Var}(Q)$ to denote the set of variables used in a conjunctive query $Q$, to which we will refer to as semantic variables. Semantic variables used in semantic descriptors of variables, i.e. $\text{Var}(Q_{ij})$, have the model for scope. This means that a semantic variable $s$ used in any $Q_{ij}$ is referring to the same concept, role or individual. The set of semantic variables used in $D_{TV}$ is denoted $\text{Var}(\mathcal{M})$.

### 4.2.1 SBCM expressiveness

Let us introduce some notions that enrich the expressiveness of SBCM. In the first place we comment temporal aspects of SBCMs. Additionally, semantic descriptors are used to express negation as failure and for introducing semantic values in the probabilistic model. Also we can semantically identify the contextual conditions on which the modeled phenomenon can start, as well as obtaining a semantic description of the final states of the SBCM.

**Temporality**

Given two nodes $V_i$ and $V_j$ in a SBCM, such that exists a causal relation $V_i \rightarrow V_j$, we say that the event represented by $V_i$ may occur before or simultaneously to $V_j$. 
4.2. THE SEMANTIC BAYESIAN CAUSAL MODEL

Negation as failure

The truth value, denoted T, is used in a random variable description to express negation as failure. It can only exist one semantic descriptor $D_K(V_i, v_{ij}, T) \in D_T(V_i)$ and will be listed at the end of $D_T(V_i)$. In this way, the variable $V_i$ will be set to $v_{ij}$ only if the rest of the semantic descriptors in $D_K(V_i)$ were evaluated negatively. In other words, it is kind of default value the variable $V_i$ will take when it’s evaluated.

Semantically valued random variables

Additionally we define a special case of semantic descriptor used to observe the probabilistic behavior of a value identified in the semantic layer.

Random variable’s domain is normally defined by a finite set of values. In consequence, the conditional probabilistic distribution of the model considers this set of values. Both of them are given a priori during the construction of the model. In the other hand, the meaning of a random variable, given by a set of semantic descriptors, maps domain values to DL expressions (in the form of conjunctive queries).

Nevertheless, constants in the semantic layer (individuals or literals) are not considered in the probabilistic model. This avoids quantifying the probability of an event on which certain individual of the real world participates.

In order to overcome this situation we need a mechanism for incorporating individuals or literals in the probabilistic model. Given that we don’t know a priori how many of them exist or how will they behave, we can only define a default probabilistic distribution that will be improved through Bayesian learning, i.e. observing the participation of such individuals in the phenomenon modeled in the Bayesian model.

Distinguished semantic variables of conjunctive queries contained in semantic descriptors will be the vehicle. This kind of variables are preceded by a question mark (?) to distinguish them from random variables.

**Definition 22** (Mixed Semantic Descriptor). A mixed semantic descriptor is given by a $D_T(V_i, ?s, Q_{iji})$ where $?s \in Dis(Q_{iji})$.

In this way, we have random variables in the Bayesian causal network with a mixed description.

**Definition 23** (Random Variable’s Mixed Description). A random variable $V_i$ has a mixed description if its domain contains at least one semantic variable $?s_i$, i.e. $?s_i \in Dom(V_i)$, and its description contains at least one mixed semantic descriptor for each $?s_i$, i.e. $D_T(V_i, ?s_i, Q_{iji}) \in D_T(V_i)$.

In the following sections will be explained the insertion of individuals in the probabilistic model.
Contextual and final conditions

$C$, given by a set of realizations $Z_i = z_i$, identifies the contextual conditions on which the observation of the probabilistic model starts. These contextual conditions are given by the semantic descriptors that describe such realizations.

**Definition 24 (SBCM Contextual Condition).** A contextual condition of a SBCM $\mathcal{M} = (M, T, D_{TV}, C, F)$ is given by the conjunctive query $context(\mathcal{M}) = \bigwedge_k Q_k$ such that exists a $D_T(Z_i, z_i, Q_k) \in D_{TV}$ for each $(Z_i = z_i) \in C$.

If exists more than one semantic descriptor for a $(Z_i = z_i) \in C$ it will be generated multiple contextual conditions for the SBCM, denoted $context_i(\mathcal{M})$, or $C_i$ for short. If exists one and only one semantic descriptor for each $(Z_i = z_i)$ in $C$, the contextual condition will be unique.

On the other hand, $F$, given by a set of realizations $Z_i = z_i$, identifies the possible final conditions of a SBCM.

**Definition 25 (SBCM Final Condition).** A final condition of a SBCM $\mathcal{M} = (M, T, D_{TV}, C, F)$ is given by the conjunctive query $final(\mathcal{M}) = Q_k$ such that exists a $D_T(Z_i, z_i, Q_k) \in D_{TV}$ for some $(Z_i = z_i) \in F$.

In this way, a SBCM will have as many final conditions as realizations $(Z_i = z_i) \in F$ and semantic descriptors for them exist in the model. Each one of them will be denoted by $final_i(\mathcal{M})$, or $F_i$ for short.

4.2.2 The SBCM Instance

I have described how a SBCM can describe a phenomenon in terms of causal relations between events, whose are represented by semantic descriptions made around random variables. Now I propose a representation of the actual occurrence of the modeled phenomenon. I call it SBCM instance.

**Definition 26 (SBCM Instance).** An instance of a SBCM $\mathcal{M} = (M, T, D_{TV}, C, F)$ is represented by a tuple

$$I_M = \langle Y, R \rangle,$$

where $Y$ stores a set of bindings of semantic variables with constants contained in an ABox $\mathcal{A}$ meanwhile $R$ stores a set of realizations of random variables in $V$. Formally,

$$Y = \{(Y_i, y_i)\mid Y_i \in Dis(Q_{ij}), D_T(V_i, v_{ij}, Q_{ij}) \in D_{TV}, y_i \in \mathcal{A}, 0 \leq i \leq n\}, \quad (4.6)$$

and

$$R = \{(V_i, v_{ij})\mid V_i \in V, v_{ij} \in Dom(V_i), 0 \leq i \leq n\}. \quad (4.7)$$
4.2. THE SEMANTIC BAYESIAN CAUSAL MODEL

A SBCM instance is well formed if for any two semantic bindings \((Y_1, y_1) \in Y\) and \((Y_2, y_2) \in Y\), \(Y_1 \neq Y_2\) holds. Likewise for any two probabilistic realizations \((V_1, v_1)\) and \((V_2, v_2)\) in \(R\).

\(R\) represents the occurrence of events represented by random variables \(V_i\), meanwhile \(Y\) keeps track of individuals or constants participating in such events. In this way, we can represent the actual occurrence of an event modeled in a SBCM.

Definition 27 (SBCM Event). The actual occurrence of an event modeled in a SBCM \(\mathcal{M}\) through the semantic descriptor \(D_T(V_i, v_{ij}, Q_{ij}) \in D_{TV}\) is represented by the probabilistic realization \((V_i = v_{ij})\) and by a set of bindings \(Y = \{(Y_k, y_k) | Y_k \in Dis(Q_{ij}), y_k \in A, 0 \leq k \leq n\}\) where \(A\) is an ABox representing a world state. The event is represented by \(E = (V_i, v_{ij}, Q_{ij}, Y)\).

The scope of semantic variables \(Y_i\) is the model. This is, given a binding \((Y_i, y_i) \in Y\) for an instance of a SBCM \(\mathcal{M}\), if \(D_T(V_1, v_1, Q_1) \in D_{TV}\), \(D_T(V_2, v_2, Q_2) \in D_{TV}\), \(V_1 \neq V_2\), \(Y_i \in Dis(Q_1)\) and \(Y_i \in Dis(Q_2)\), implies that the SBCM instance is supported by the occurrence of two events \(E_1 = (V_1, v_1, Q_1, Y_1)\) and \(E_2 = (V_2, v_2, Q_2, Y_2)\) such that \((Y_i, y_i) \in Y_1\) and \((Y_i, y_i) \in Y_2\).

According to this definition, an individual \(y_i\) participates in both events \(E_1\) and \(E_2\). Nevertheless, both events not necessarily occur simultaneously.

A SBCM instance \(I_\mathcal{M} = (Y, R)\) is said to be complete if \(R\) contains a realization \(V_i = v_i\) for each variable \(V_i \in V\) defined in \(\mathcal{M}\).

4.2.3 SBCM Primitives

A SBCM has two dimensions: (i) one probabilistic and (ii) one semantic. The former allows to perform the processes and inferences described in Section 2.6.2. The latter provides an interface for incorporating information from a semantic representation of the world state (an ABox).

In this point we introduce the notion of DL knowledge base \(K = (T, A)\) where the TBox \(T\) is used for expressing semantic descriptors in a SBCM \(\mathcal{M}\), meanwhile the ABox \(A\) represents the world state \(W^t\). \(A\) must be consistent with respect to \(T\), i.e. facts must be expressed using concepts and roles defined in \(T\) as well as satisfy constraints defined in \(T\).

Facts expressed in \(A\) are used for creating instances of \(\mathcal{M}\) and for updating them.

Instantiation

Contextual conditions \(C_i\) provide a fix point for start monitoring a process or following a plan modeled through the SBCM \(\mathcal{M}\).
Definition 28 (SBCM Instantiation). Given a SBCM $M$ and a knowledge base $K = <T, A>$, a SBCM instance $I_M = <Y, C>$ is created if there exists a set of bindings $Y$ such that $K \models_Y C_i$ for some contextual condition $C_i$ of $M$.

Every different binding $Y$ produced by the evaluation of $C_i$ on $K$ will produce a SBCM instance. In this way, we can represent the instantiation operation by $\text{instantiate}(M, K, Y) : I_M^i$ for $0 \leq i \leq n$, where $Y$ is an optional set of semantic bindings that constrains the instantiation of SBCM instances. This function is described in Algorithm 1.

The function $\text{evalQuery}(K, Q, Y)$ evaluates a conjunctive query $Q$ in a knowledge base $K$ constraining that matches of $Q$ in turn match a set of given bindings $Y$, represented $K \models_Y Q$. This means that if $Y$ contains a binding $S_i = s_i$ and $S_i \in \text{Vars}(Q)$, then $S_i$ is replace by $s_i$ in $Q$ before its evaluation. $\text{evalQuery}()$ returns: 1) $\perp$ if $Q$ is evaluated negatively, 2) $T$ if $Q$ has no distinguished variables and is evaluated positively, and 3) an array of sets of bindings $\overline{Y} = \{\overline{Y}[i] | 0 < i \leq n\}$ such that $\overline{Y}[i] = \{(S_j = s_j) | S_j \in \text{Vars}(Q), 0 < j \leq n\}$ and each $\overline{Y}[i]$ contains all bindings of the original $Y$, if $Q$ has distinguished variables and is evaluated positively. If $Q = T$, $\text{evalQuery}()$ returns $T$.

For incorporating semantic information into the probabilistic model we use two functions (lines 20 – 24). $\text{isSemVar}()$ determines if $v_i$ contains a semantic value. If so, the function $\text{getValueFor}()$ is used for extracting the semantic value of $v_i$ in a given bindings set. New values $?s_i$ are incorporated in the joint probabilistic distribution during off line learning.

Evidence Feeding

A SBCM instance $I_M$ can be updated with information contained in a knowledge base $K = <T, A>$. Semantic descriptors describing observable variables (covariates) in $M$ are evaluated in $K$ to determine if the described events occurs in $A$.

Algorithm 2 describes the procedure for setting evidence observed in $K$ into a SBCM instance $I_M$. In order to do so, this procedure traverse the DAG of the model in causal order revising if semantic descriptors of unrealized variables produce any matches.

Query matches are used for adding bindings to $Y$ and probabilistic realizations to $R$. Given that query matching produced by the evaluation of a semantic descriptor may produce multiple binding sets, this function might returns multiple SBCM instances. Each one of them represents a possible instance of the phenomenon that satisfied the constraints imposed by previous bindings. In the end, only a complete SBCM represent an actual instance of the phenomenon.

In Algorithm 2, function $\text{vars}()$ receives a set of probabilistic realizations and returns the list of realized variables. The function $\text{order}()$ sorts a set of variables according to some partial order given by causal relations contained in the DAG $G$. In this partial order, variable $V_1$ is ordered before than $V_2$, denoted $V_1 \preceq V_2$ if $V_1$ is not descendant

4.2. THE SEMANTIC BAYESIAN CAUSAL MODEL

Algorithm 1: Function $\text{instantiate}(\mathcal{M}, K, Y)$

```plaintext
input: A SBCM $\mathcal{M}$, a knowledge base $K = (\mathcal{T}, \mathcal{A})$ and a (possibly empty) set of semantic bindings $Y$.
output: A (possibly empty) list of SBCM instances.

$\text{FUNCTION instantiate}(\mathcal{M}, K, Y);$

Create new SBCM instances of $\bar{M}$ from $K$ observing $Y$;

begin
contexts $\leftarrow \text{context}(\mathcal{M})$;
instances $\leftarrow \emptyset$;
foreach $C_i \in$ contexts do
\hspace{1em} generate SBCM instances for each context;
\hspace{1em} $\bar{Y} \leftarrow \text{evalQuery}(K, C_i, Y)$;
\hspace{1em} if $\bar{Y} \neq \bot$ then
\hspace{1em} \hspace{1em} if $\bar{Y} = \top$ then \hspace{1em} // $C_i$ has no distinguished variables
\hspace{1em} \hspace{1em} \hspace{1em} $I_M \leftarrow \text{new SBCMInstance}()$;
\hspace{1em} \hspace{1em} \hspace{1em} $I_M.Y \leftarrow Y$;
\hspace{1em} \hspace{1em} \hspace{1em} $I_M.R \leftarrow \mathcal{M}.C$;
\hspace{1em} \hspace{1em} \hspace{1em} instances $\leftarrow$ instances $\cup$ $I_M$;
\hspace{1em} \hspace{1em} else \hspace{1em} // $\bar{Y}$ contains multiple sets of bindings
\hspace{1em} \hspace{1em} \hspace{1em} for $j \leftarrow 1$ to $|\bar{Y}|$ do
\hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} create a SBCM instance for each set of bindings $\bar{Y}[j]$;
\hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} $I_M \leftarrow \text{new SBCMInstance}()$;
\hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} foreach $(v_i = v_i) \in \bar{Y}[i]$ do
\hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} if isSemVar($v_i$) then \hspace{1em} // $v_i$ is a semantic variable
\hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} if $\bar{Y}[j].getValueFor(v_i) \in \text{Dom}(V_i)$ then
\hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} $v_i \leftarrow \bar{Y}[j].getValueFor(v_i)$;
\hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} end
\hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} end
\hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} $I_M.Y \leftarrow Y \cup \bar{Y}[i]$;
\hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} $I_M.R \leftarrow \mathcal{M}.C$;
\hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} instances $\leftarrow$ instances $\cup$ $I_M$;
\hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} end
\hspace{1em} \hspace{1em} end
\hspace{1em} \hspace{1em} end
\hspace{1em} \hspace{1em} \hspace{1em} end
\hspace{1em} \hspace{1em} return instances;
\hspace{1em} end
```

Algorithm 2: Function setEvidence($I_M, K$)
of $V_2$ in $G$. As well, function `parents()` returns the list of direct parents of the given variable in a DAG.

During evidence feeding, there are evaluated only those variables without parents or with at least one realized parent (line 16). In this way, evidence feeding is interrupted in causal paths containing unobserved variables that d-separate descendant variables. We would be able to draw an upper subgraph that covers those variables realized in the SBCM instance.

The function `evalVariable()` checks if some event related to a SBCM instance $I_M$ is expressed in $K$. For doing so, it evaluates all semantic descriptors $D_T(V_i)$ in the given order. The first descriptor evaluated positively is used for incorporating semantic and probabilistic information in $I_M$. Meanwhile this procedure produces a single probabilistic realization, query matching might throw different bindings, which causes the creation of multiple SBCM instances with as many bindings as produced. This function is described in Algorithm 3.

### 4.3 The Semantic Causal Network

I extend the SBCM formalism for considering additional constraints like universal or existential quantifiers as well as the incapacity for observing certain effects considered in decision making.

**Definition 29 (SCN).** A *Semantic Causal Network* (SCN) is used for representing the perspective an agent $Ag$ has of a process, i.e. a plan. A SCN is an extension of a SBCM and is represented by:

$$M = \langle Ag, M, U, Q \rangle$$

where $Ag$ is an agent type, $M = \langle m, T, D_T(v), C, F \rangle$ is a SBCM on which a single variable is referenced in $F$, i.e. $V_i = v_i \in F \land V_j = v_j \in F \rightarrow V_i = V_j$, $U \subseteq Z$ represents effects that cannot be observed (nor manipulated) by agents of type $Ag$, i.e. $V_i \in U \land (V_i, V_j) \in G \rightarrow V_j \in U$, and $Q : X \times Vars(D_T(X)) \times q$ is a relation of quantifiers $q \in \{\text{ONE, ALL}\}$ associated to semantic variables used to constraint the execution of actions in the plan. Besides the constraint on semantic descriptors $Q_{ij} \neq \bot$ is eliminated (see Def. 20).

The function `Ann()` is used for retrieving the CQs that semantically describe a probabilistic event. It receives as argument a probabilistic realization $V_i = v_i$ and returns a disjunction of conjunctive queries $Q_1 \lor \ldots \lor Q_n$ obtained from each $D_T(V_i, v_i, Q_j) \in D_Y$. In particular, `Ann(C)` returns the unique contextual condition `context(M)` (Definition 24) meanwhile `Ann(F)` returns the disjunction of final conditions `finali(M)` (Definition 25), also abbreviated as `Ann(F)` or `Ann_i(F).

The annotation `Ann(X_i = True)` contains the action role assertions $\{do(E, \alpha), aType(\alpha)\}$ indicating the execution of the action $aType$. The annotation `Ann(X_i = False)` is set to $\bot$. Classifying a variable as controllable means that agent $Ag$ is capable of executing
CHAPTER 4. SEMANTIC CAUSAL MODELS

input : A SBCM instance $I_M = (Y, R)$, a knowledge base $K = (T, A)$ and a variable $V_i$.
output: A (possibly empty) list of SBCM instances.

FUNCTION evalVariable($I_M, K, V_i$);

Evaluates semantic descriptors of $V_i$ in $K$, incorporating semantic and probabilistic values in $I_M$;

begin
for $l \leftarrow 1$ to $|\mathcal{M}.D_T(V_i)|$ do

evaluate every semantic descriptor in the given order;

descriptor $\leftarrow \mathcal{M}.D_T(V_i)[l]$;

$Q_{ijl} \leftarrow$ descriptor.query;

$V_i \leftarrow$ descriptor.variable;

$v_i \leftarrow$ descriptor.value;

$\overline{Y} \leftarrow$ evalQuery($K, Q_{ijl}, I_M.Y$);

if $\overline{Y} \neq \bot$ then

if $\overline{Y} = \top$ then  // $Q_{ijl}$ has no distinguished variables

$I_M' \leftarrow$ new SBCMInstance();

$I_M'.Y \leftarrow I_M.Y$;

$I_M'.R \leftarrow I_M.R \cup (V_i, v_i)$;

return $\{I_M'\}$;

else  // $\overline{Y}$ contains multiple sets of bindings

instances $\leftarrow \emptyset$;

for $i \leftarrow 1$ to $|\overline{Y}|$ do

create a SBCM instance for each set of bindings $\overline{Y}[i]$;

$I_M' \leftarrow$ new SBCMInstance();

if isSemVar($v_i$) then  // $v_i$ is a semantic variable

$v_i \leftarrow \overline{Y}[i].getValueFor(v_i)$;

if $v_i \notin Dom(V_i)$ then updateVarDomain($\mathcal{M}, V_i, v_i$);

end

$I_M'.Y \leftarrow I_M.Y \cup \overline{Y}[i]$;

$I_M'.R \leftarrow I_M.R \cup (V_i, v_i)$;

instances $\leftarrow$ instances $\cup I_M'$;

end

return instances;
end
end

return $\emptyset$;
end

Algorithm 3: Function evalVariable($I_M, K, V_i$)
or enabling the action $\alpha$ associated to $X_i$. For expressing that $Ag$ is responsible for directly executing $X_i$, $Ann(X_i = True)$ must contain the predicate $do(Ag, \alpha)$. In the other hand, for expressing that $Ag$ is responsible for enabling $X_i$, but not necessarily executing $X_i$, $Ann(X_i = True)$ must contain the predicate $do(Ag_2, \alpha)$, where $Ag_2 \neq Ag$.

A variable $V_i$ with a single semantic descriptor $D_T(V_i, v_i, Q_{ij})$ for $v_i$ where $Q_{ij} = \bot$ will not be realized to this value ($V_i = v_i$) during evidence feeding.

Covariates ($Z_i \in Z$), on the other hand, are multivalued (inclusively boolean). Each annotation $Ann(Z_i = z_i)$ represents events or conditions associated to $Z_i$. Negation as failure is used on a covariate ($Ann(Z_i = z_i) = \top$) for indicating the default value that will be set when the variable is evaluated. The lack of a negation as failure annotation on a covariate realization indicates that the variable will not be set with observed evidence unless a valid condition ($Ann(Z_i = z_i)$) be met.

Causes that cannot be observed by $Ag$ are omitted in the SCN. Nevertheless, effects $U_i$ that cannot be observed by $Ag$ can still be represented in the SCN for calculating the causal effect of actions $X_i$ over them. The SCN constrains that unobservable variables can only be cause of other unobservable event, i.e. if $V_i \rightarrow V_j \in G$ and $V_i \in U$ then $V_j \in U$.

In a SCN, an event is represented by the set of annotations associated to a probabilistic variable realization, $E_{vi} = Ann(V_i = v_i)$. A causal relation between two events $E_{vi} = Ann(V_i = v_i)$ and $E_{vj} = Ann(V_j = v_j)$ is expressed on $G_V$ by an arc going from $V_i$ to $V_j$. The total causal effect of $E_{vi}$ over $E_{vj}$ is given by $P(V_j = v_j|V_i = v_i)$, which is obtained from $P(v_i|pa_i)$.

A quantifier $Q_i \in Q$ is represented by a tuple $Q_i = (X_i, ?s, q)$, where $q \in \{\text{ONE}, \text{ALL}\}$, $X_i$ is a control variable and $?s$ is a distinguished variable of the annotation of $X_i$, i.e. $?s \in Dis(Ann(X_i = True))$. The quantifier $q$ indicates how many actions are required in the phenomenon modeled in terms of the involved individuals. For instance, in a SCN describing a sealed auction, for indicating that the bid submission action ($X_{bid}$) must be performed by every bidder ($?bidder$), the semantic variable $?bidder$ must be quantified with ALL, represented $Q_b = (X_{bid}, ?bidder, \text{ALL})$.

### 4.3.1 SCN Strategies

Given that a SCN might contain multiple causal chains connecting the contextual condition to the node representing the final condition, we may have multiple ways of achieving the goal in a single SCN. Each one of these causal chains is considered a strategy and is represented by a sequence of realizations on control variables.

**Definition 30** (SCN Strategy). Given a a SCN $M = (Ag, M, U, Q)$, a SCN strategy is a finite and ordered sequence of realizations on control variables $S = \langle x_1, ..., x_n \rangle$ where $x_i$ stands for $X_i = True$ or $X_i = False$, and exists one and only one realization for each
$X_i \in X$, such that

$$\sum_{f_i \in F} P(f_i|C, do(x_1), ..., do(x_n)) > 0$$

In order to calculate every possible SCN strategy it must be considered any partial order of control variables $X_i \in X$. The order of covariates $(Z)$ is not important. The set of SCN strategies identified in a SCN $M$ is denoted $\mathcal{S}(M) = \{S_1, ..., S_n\}$. $\mathcal{S}(M)$ can be represented through a Binary Decision Diagram (BDD) [2].

**Definition 31 (Strategies BDD).** A Binary Decision Diagram (BDD) represents a set of strategies $\mathcal{S}(M) = \{S_1, ..., S_n\}$ identified in the SCN $M$, if each decision node $N_i$ represents a control variable $X_i$, denoted $\text{Var}(N_i)$, the low child represents the decision $X_i = \text{False}$, denoted $\text{Low}(N_i)$, the high child represents the decision $X_i = \text{True}$, denoted $\text{High}(N_i)$, and the Boolean function $f(S_i)$ it represents is given by:

$$f(x_1, ..., x_n) = \begin{cases} 1 & \text{if } P(F|C, do(x_1), ..., do(x_n)) > 0 \\ 0 & \text{otherwise} \end{cases} \quad (4.9)$$

From this definition we can observe that a non-terminal node of a BDD may be used for deciding between executing $X_i$ ($X_i = \text{True}$) and not executing $X_i$ ($X_i = \text{False}$), if the node has both children. Besides, it may exist multiple BDD nodes representing the same $X_i$. If this is the case, we conclude that $\mathcal{S}(M)$ contains at least two strategies $S_1$ and $S_2$ where $X_j = \text{True} \in S_1$ and $X_j = \text{False} \in S_2$, being $i > j$ w.r.t. the partial order of $X$.

If all the strategies in $\mathcal{S}(M)$ contain realizations for every $X_i \in X$ and such realizations are sorted in the BDD with respect to a partial order of $X$, the BDD becomes an *Ordered BDD* (OBDD). Strategies generated from different partial orders of $X$ are arranged in separated OBDDs.

**Definition 32 (Strategies OBDD).** Being $O = \{X_1, ..., X_n\}$ a partial order of control variables of $M$ and $\mathcal{S}(M) = \{S_1, ..., S_n\}$ a set of SCN strategies identified in $M$ such that $S_i$ contains a realization of every variable $X_i \in O$, a BDD representing $\mathcal{S}(M)$ is considered *ordered* if given a node representing $X_i$ its children represent $X_{i+1}$ w.r.t. $O$. The OBDD is represented by $\mathcal{S}(M, O)$.

A partial strategy $S^i = (x_1, ..., x_i)$ of the strategy $S = (x_1, ..., x_n)$ encoded in $\mathcal{S}(M, O)$, is represented by the node $N_{i+1}$ if $\mathcal{S}(M, O)$ contains a set of OBDD nodes $N_i$ such that $\text{Var}(N_i) = X_i$, $\text{Low}(N_i) = N_{i+1}$ if $X_i = \text{False} \in S$ and $\text{High}(N_i) = N_{i+1}$ if $X_i = \text{True} \in S$, for $1 \leq i \leq n$.

Finally, we use both the SCN Bayesian network and the strategies OBDD for proposing another view of the SCN. The graph is unfolded following the strategies identified in the strategies OBDD producing another rooted DAG that, additionally to actions, contains intermediate conditions (covariates) that justify agent decisions.
4.3. THE SEMANTIC CAUSAL NETWORK

From the BDD diagram we define a new one on which three constraints are relaxed:
1) nodes may also represent covariates, 2) interventions \( X_i = False \) are omitted, and
3) the number of children for a node is unconstrained. Besides, the terminal node is
replaced by the covariate associated to \( F \) and the diagram only includes those actions
to execute, i.e. \( X_i = True \).

**Definition 33** (Causal Decision Diagram). A Causal Decision Diagram (CDD) is a
rooted DAG on which each node represents a variable \( V_i \) of a SCN \( M \), children are
connected through arcs labeled with one or more values \( v_i \in \text{Dom}(V_i) \), the terminal
node is represented by the covariate \( V_F \) associated to the final condition \( F \) of the SCN,
and the diagram contains a rooted directed path \( V_1 \rightarrow ... \rightarrow V_n \rightarrow V_F \)
if and only if

\[
\sum_{f_i \in F} P(f_i|V_1 = v_1, ..., V_n = v_n, x_1, ..., x_j) > 0
\]

for each \( x_j \in \overline{S}(M) \), where each \( V_i \) is not descendant of \( V_{i+1} \) and \( v_i \) is labeled in the arc outgoing from \( V_i \).

The set of variables included in the CDD is *minimal* if only includes those covariates in
a directed path between two actions \( X_i \) and \( X_{i+1} \). If every path follows the same order,
the CDD is said to be *ordered*, OCDD for short. The CDD contains as many terminal
nodes as strategies exist in the SCN; each terminal node represents a strategy.

4.3.2 Graphical Representation

SCNs are represented graphically as causal networks that incorporate semantic anno-
tations on its nodes. Controllable variables on \( X \) are represented by boxes whereas
covariates on \( Z \) are represented by rounded boxes. Each variable is identified by a \( Z \)
or an \( X \) with a consecutive subindex, located next to the (rounded) box. Annotations
\( \text{Ann}(V_i = v_i) \) for each \( v_i \in \text{Dom}(V_i) \) are represented in the \( V_i \) box. Annotations
\( \text{Ann}(V_i = v_i) \) for each \( v_i \in \text{Dom}(V_i) \) are represented in the \( V_i \) box.

If \( Z_i \) has a single annotation different from \( T \), the annotation \( \text{Ann}(Z_i = z_i) \) is contained
in the rounded box representing \( Z_i \). If \( Z_i \) has more than one annotation different from \( T \),
each annotation \( \text{Ann}(Z_i = z_i) \) is included in the rounded box representing \( Z_i \) preceded
by an annotation label \( z_i \): and separated from each other by an horizontal dotted line.
Annotations like \( \text{Ann}(Z_i = True) \) can be represented without label, i.e. label \( True \)
is assumed. A semantically valued covariate \( Z_i \) is identified by an annotation label
starting with a question mark (?) and followed by a variable name distinguished in
the conjunctive query contained in the annotation. Unobservable covariates (\( U_i \)) have a
gray background.

Control variables, represented by boxes, use a notation slightly different to conjunctive
queries:

\[ ?ag \alpha Type ?ct \text{prop}_1 \text{val}_1 ... \text{prop}_n \text{val}_n \]

where \( ?ag \) represents the execution of \( \alpha \) by \( ag \) over the caused/modified thing \( ?ct \), and
\( \text{prop}_i \) represents a modifier (parameter) of the action with value equal to \( \text{val}_i \). This
CHAPTER 4. SEMANTIC CAUSAL MODELS

notation is translated in the conjunctive query:

\[
\text{do(?ag, ?act), \alphaAction(?act), prop_1(?act, val_1), ..., prop_n(?act, val_n)}
\]

where \( \alphaAction \) is a DL concept representing the type of action that \(?ag\) will perform and \(?act\) represents the actual execution of the action.

A conjunctive query \( Q = (V, q) \) representing an annotation \( Ann(V_i = v_i) \) is depicted using only \( q \) and formatting distinguished variables (\( Dis(Q) \)) in bold. Covariates which realization is included in the contextual condition are illustrated with rounded boxes with a bold outline. Covariates used for representing final conditions are illustrated by rounded boxes with double-line.

Figure 4.1 shows an example of a SCN illustrating the graphical representation. All variables except for \( Z_4 \) are boolean. \( Z_4 \) has two annotations and both are represented by their respective conjunctive query. The context of this SCN is \( C = \{ Z_2 = True \} \) and its final state is \( F = \{ Z_3 = True \} \). The execution of the action \( \alpha \) is represented by \( X_1 \). The unique unobservable variable in this example is \( Z_5 \). In annotations, \( C_i \) represents concepts and \( P_i \) represents roles.

![Figure 4.1: Graphical representation of a SCN.](image)

**Workflows**

SCNs can be used for representing different types of workflows as shown in Figure 4.2. A **sequential workflow** is represented in a SCN by a sequence of events and actions chained causally (Fig. 4.2.a). A **choice or parallel workflow** is represented by multiple causal chains diverging from a single node and converging in another (Fig. 4.2.b).

The probabilistic distribution is used for indicating if events (or actions) present in diverging paths can occur simultaneously or not. Given the example of Fig. 4.2.b,
4.3. THE SEMANTIC CAUSAL NETWORK

$P(x_1|x_2, z_1) = 0$ indicates that $x_1$ cannot occur whenever $z_1$ and $x_2$; if $P(x_2|x_1, z_1) = 0$ then a choice must be done. Whereas $P(x_1|x_2, z_1) > 0$ indicate that $x_1$ might occur simultaneously with $x_2$.

![Diagram of workflow types](image)

(a) Sequence  (b) Choice / Parallel  (c) Mixed dependencies

Figure 4.2: Types of workflows represented through SCNs.

Any combination of both kind of workflows can be represented through the SCN and its probabilistic distribution (Fig. 4.2.c).

**Theorem 1** (Strategy identification in SCN Workflows). The causal effect of the strategy $x_1, ..., x_n$ is identifiable in any SBCM $M$ containing sequential, choice, parallel workflows or any combination of them, as long as the $g$-identifiability criterion be satisfied.

Proof. According to the $g$-identifiability criterion (see equations 2.13 and 2.14), plan identification relies on the identification of sets of covariates ($W_k$) that must be observed for eliminating causal dependencies that might confound the causal effect $P(f_i|do(x_k), do(x_{k+j}))$ for any two pairs of control variables $X_k$ and $X_{k+j}$ in the plan. The capability of identifying plans in sequential workflows is feasible as long as $W_k$’s remove back-door dependencies in the identification of causal effects in the plan.

In the other hand, choice or parallel workflows contain alternative paths that lead to $F$ on which a control variable $X_k$ is allocated in one path and $X_{k+j}$ is allocated in another. $X_k$ is not ancestor of $X_{k+j}$ and viceversa. Nevertheless, in [72] Pearl and Robins demonstrated that $X_k$ must not be necessarily ancestor of $X_{k+j}$. For this reason we can conclude that plan identification in SCNs representing parallel/choice workflows is feasible.

Combinations of these workflows can be reduced to sequential and parallel segments. Hence the result can be extended to any SCN configuration. □
4.3.3 The SCN Case

I introduce a new formalism called *SCN Case* which groups a set of SBCM instances (Def. 26) used for achieving a goal through the execution of actions identified in SCN strategies. This set of SBCM instances must satisfy the constraints imposed by SCN quantifiers.

Similarly to SBCMs (see section 4.2.3), a SCN can be instantiated and updated. The status of the SCN case depends on the status of the SBCM instances contained in it and is modified by the criterion of observability $U$ introduced in SCNs.

In the first place, let us introduce a criterion for determining if a set of SBCM instances satisfy the constraints imposed by SCN quantifiers.

The Q-Sat criterion

Quantifiers defined in a SCN $M = (Ag, M, U, Q)$ indicate how many action executions represented by $X_a = True$ should be executed in order to satisfy the given specification of the process. Given that each SBCM instance represents a single action execution of $X_a$, a set of them is required for satisfying an ALL quantifier. In the other hand, a ONE quantifier is satisfied by any execution of $X_a$.

Let's use the notation $I_M = \{I_1, ..., I_n\}$ for representing a set of instances of the SBCM $M$. Individuals or values identified by the same semantic variable $?S_i$ in all SBCM instances contained in a $I_M$ are represented by

$$vals(I_M, ?S_i) = \{s_{ij} | (?S_i = s_{ij}) \in Y, I_i(Y_i, R_i) \in I_M\}$$

Each SBCM instance $I_i \in I_M$ represents the potential or actual execution of actions identified by control variables $X_i$. Action execution, or *act*, is represented by $X_i = True \in R$, where $R$ contains probabilistic realizations associated to the SCN case. $Y$ stores the values of causes (or parameters) of the action. *Omission* is represented by $X_i = False \in R$, and doesn't requires nor produces semantic bindings in $Y$.

If the action $X_i$ is omitted in all the SBCM instances that compose a SCN case, quantifiers $Q_i = \langle X_i, ?S_i, ONE \rangle$ are considered satisfied. But if at least one SBCM instance reflects the execution of $X_i$, then the corresponding quantifiers are evaluated.

An *ONE-quantifier* $Q_{ai} = \langle X_a, ?S_i, ONE \rangle$ is satisfied in $I_M$ if exists at least one SBCM instance $I_k = \langle Y_k, R_k \rangle \in I_M$ where $X_a = True \in R_k$, $(?S_i = s_{ij}) \in Y_k$ for some $s_{ij} \in vals(I_M, ?S_i)$.

In the other hand, an *ALL-quantifier* $Q_{ai} = \langle X_a, ?S_i, ALL \rangle$ is satisfied in $I_M$ by a subset of SBCM instances $I'_M \subseteq I_M$ such that for each $s_{ij} \in vals(I_M, ?S_i)$ exists at least one $I_k = \langle Y_k, R_k \rangle \in I'_M$ where $X_a = True \in R_k$ and $(?S_i = s_{ij}) \in Y_k$.

**Definition 34** (Q-Sat Criterion). All quantifiers of action $X_a$, denoted $Q(X_a)$, are satisfied by a set of SBCM instances $I_M = \{I_1, ..., I_n\}$ if and only if:
4.3. THE SEMANTIC CAUSAL NETWORK

i. for all $I_i = (Y, R) \in I_M$, $X_i = False \in R$,

ii. $Q(X_a)$ only contains ONE-quantifiers and exists some $I_i = (Y, R) \in I_M$ such that $X_i = True \in R$, or

iii. $Q(X_a)$ contains some ALL-quantifiers and exists a subset $I'_M \subseteq I_M$ on which every $I_i = (Y, R) \in I'_M$ has $X_i = True \in R$ and there is some $I_i \in I'_M$ having $(?S_1 = s_1, ..., ?S_n = s_n) \in Y$ for each $Q_j = (X_a, ?S_j, ALL) \in Q(X_a)$ and each $s_j \in \text{vals}(I_M, ?S_j)$, for $0 < j < m$,

which is denoted $satisfiesQ(I_M, X_a)$.

SCN Case formal definition

Now it is formally defined a SCN case.

Definition 35 (SCN Case). A case of the SCN $M = \langle Ag, M, U, Q \rangle$ is represented by

$$C_M = (I_M, S(M, O), N)$$

(4.10)

where $I_M = \langle I_1, ..., I_n \rangle$ is a list of instances of the SBCM $M$ called trials, $S(M, O)$ is a OBDD, $N$ is a vector of size $n$ where each $N_i \in N$ points to some node of $S(M, O)$.

If $N_i$ is the terminal node, there is no action $x_i$ encoded in $S(M, O)$ that can be executed for $I_i$. Being $N_i$ a pointer to a non terminal node, if $\text{High}(N_i) \in S(M, O)$ then $\text{Var}(N_i) = True$ is included in some $S \in S(M, O)$. Similarly, if $\text{Low}(N_i) \in S(M, O)$ then $\text{Var}(N_i) = False$ is also included in some $S \in S(M, O)$.

A SCN case $C_M = (I_M, S, X)$ is considered complete if for each SBCM instance $I_i = \langle Y_i, R_i \rangle \in I_M$: 1) $(V_F = v_F) \in R_i$ for some $v_F \in \text{Dom}(V_F)$ where $V_F$ is referenced in the final condition $F$, or 2) for every $V_i \in Z \setminus U$, exist a realizations $V_i = v_i \in Y_i$. Otherwise, the SCN case is considered incomplete.

A SCN case $C_M = (I_M, S, X)$ is considered inviable if exists at least one SBCM instance $I_i = \langle Y_i, R_i \rangle \in I_M$ such that: 1) $N_i = \bot$ or 2) for every children $Ch_i$ of $N_i$, $\sum_{f_k \in F} P(f_k|R_i, do(x_i)) = 0$ where $X_i = True$ if $Ch_i = \text{High}(N_i)$ or $X_i = False$ if $Ch_i = \text{Low}(N_i)$.

4.3.4 SCN-driven Plans

An incomplete SCN case can be used for driving the execution of a plan. The SCN case is used for identifying enabled and viable actions that lead to the achievement of some $f_i \in F$. In the first place, we define potential actions in SCN cases.

Definition 36 (Potential action). Given a SCN case $C_M = (I_M, S(M, O), N)$, a potential action is represented by a valid intervention $x_i$ in a trial $I_i \in I_M$ according to the OBDD pointer $N_i$, and is denoted $\alpha = (C_M, I_i, x_i)$. 
Feasibility of action execution depends on the current conditions represented by the action’s trial.

**Definition 37** (Enabled action). A potential action $\alpha = \langle C_M, I_i, x_i \rangle$ of the trial $I_i = \langle Y, R \rangle$ is considered *enabled* if

$$P(x_i|R) = 1$$

(4.11)

which indicates that there is total certainty of the action execution in current conditions $(R)$.

In the opposite way, $\alpha$ is considered *disabled* if $P(x_i|R) = 0$. If $0 < P(x_i|R) < 1$, we cannot assure if the action can be executed or not. This uncertainty may be caused by unobserved preconditions of $X_i$. $W_k$ sets obtained during strategies identification are used for identifying these preconditions. $W_i$ identifies the variables that must be observed in order to determine the causal effect of $X_i$ on $F$, and includes variables that d-separate the effect of previous actions $x_{i-j}$ in the plan.

**Definition 38** (Unobserved action preconditions). Given a potential action $\alpha = \langle C_M, I_i, x_i \rangle$ of the trial $I_i = \langle Y, R \rangle$, a covariate $Z_i$ is an *unobserved precondition* of $\alpha$ if $Z_i \in (W_i \cap An(X_i))$ and $Z_i = z_i \notin R$ for any $z_i \in Dom(Z_i)$.

An action that has at least one unobserved precondition and which execution is not assured or denied with certainty is considered an *uncertain action*. The set of sets of unobserved preconditions that enable an uncertain action can be determined evaluating the possible values they can hold.

**Definition 39** (Missing conditions). The potential action $\alpha = \langle C_M, I_i, x_i \rangle$ of the trial $I_i = \langle Y, R \rangle$ is *enabled* by a set of probabilistic realizations $\text{Mis}(x_i) = \{(Z_j = z_j) | Z_j \in (W_i \cap An(X_i)), (Z_j = z_j) \notin R, 0 < \{j, k, l\} \leq n\}$ if

$$P(x_i|R, \text{Mis}(x_i)) > 0$$

The causal effect, or probability of success, of uncertain actions can be evaluated in the possible worlds given by each $\text{Mis}(x_i)$ found. This calculation can be generalized if we consider that $\text{Mis}(x_i) = \emptyset$ when all the preconditions for $x_i$ are observed.

**Definition 40** (Action causal effect). The *causal effect* of the enabled action $\alpha = \langle C_M, I_i, x_i \rangle$ on the desired final state $F$ of $M$ is given by

$$ce(\alpha) = \sum_{f_i \in F} P(f_i|R, \text{Mis}(x_i), do(x_i))$$

(4.12)
4.3. THE SEMANTIC CAUSAL NETWORK

The causal effect of an action $x_i$ can be used for determining its viability for reaching some final state. Nevertheless, if $\text{Mis}(x_i) \neq \emptyset$, the missing conditions must be set before executing $x_i$. In Section 5.4.3 it is described how to determine if such actions can be set.

**Definition 41** (Viable Action). An action $\alpha = \langle C_M, I_i, x_i \rangle$ is considered viable if $ce(\alpha) > 0$ and $\text{Mis}(x_i) = \emptyset$ or $\text{Mis}(x_i)$ can be set by other means.

The causal effect of a viable action also indicates the probability of success for the trial. In the evaluation of the action causal effect we are evaluating multiple strategies simultaneously.

**Theorem 2** (Accrued strategies evaluation). The causal effect $ce(\alpha)$ of the action $\alpha = \langle C_M, I_i, x_i \rangle$ calculates the accrued probability of success of all the valid strategies represented by the node $N_i$ of the strategies OBDD $\overline{S}(M, O)$ on which $x_i$ is identified.

*Proof.* The OBDD $\overline{S}(M, O)$ represents multiple strategies $S_i = \langle x_1, ..., x_n \rangle$ (see Def. 32). Each OBDD node $N_i$ represents a partial strategy $S^i = \langle x_1, ..., x_i-1 \rangle$. The subset of strategies $\overline{S'} \subseteq \overline{S}(M, O)$ containing the partial strategy $S^i$ are represented by a single OBDD node $N_i$. Each strategy $S_i \in \overline{S'}$ has a different set of actions $S^{i-n} = \langle x_i, ..., x_n \rangle$ that produce the branches that descend from $N_i$. Each one strategy $S_i \in \overline{S'}$ containing $x_i$ is represented by the edge connecting to the (high or low) child of $N_i$.

Through $ce(\alpha)$ we are calculating the probability of setting some $f_i \in F$ through any strategy $S_i \in \overline{S'}$ containing $x_i$. The viability and causal effect of remaining actions $S^{i-n}$ for each strategy are encoded in the SBCM and are considered when $P(f_i|do(x_i), R)$ is evaluated. \qed

Additionally, a viable action must satisfy the Q-Sat criterion for being considered valid.

**Definition 42** (Action validity). A viable action $\alpha = \langle C_M, I_i, x_i \rangle$ of the trial $I_i = \langle Y, R \rangle$ is considered invalid if violates the quantifiers associated to $X_i$, denoted $Q(X_i)$. This violation occurs when:

1. $Q(X_i)$ only contains ONE-quantifiers and already exists some $I_2 = \langle Y_2, R_2 \rangle \in C_M$ such that $x_i \in R_2$, or

2. $Q(X_i)$ contains some ALL-quantifiers and already exists some $I_2 = \langle Y_2, R_2 \rangle \in C_M$ such that $?S_j = s_j \in Y_2$ and $?S_j = s_j \in Y$ for all quantifiers $Q_j = \langle X_i, ?S_j, ALL \rangle \in Q(X_i)$.

Otherwise, the viable action is considered valid.

In this way, the SCN case can produce viable actions that do not violate the constraints imposed by quantifiers. Going further, we can identify those trials that contain valid actions.
Definition 43 (SCN Trial Viability). A plan trial \( I_t = \langle Y, R \rangle \in I_M \) of a SCN Case \( C_M = \langle I_M, \overline{S}(M,O), N \rangle \) is considered *viable* if it has at least one valid action \( \alpha = \langle I_P, I_t, x_i \rangle \). Otherwise, the trial is considered *inviable*.

Now we must decide if a trial is necessary or not for satisfying the SCN quantifiers.

Definition 44 (Unnecessary SCN Trial). A plan trial \( I_t \in I_M \) of a SCN Case \( C_M = \langle I_M, \overline{S}(M,O), N \rangle \) is considered *unnecessary* if its potential actions refers to \( X_i \) and satisfies \( Q(I'_M, X_j) \) holds for \( 1 < j < i \).

Finally, we can determine if a SCN case \( C_M \) satisfies the constraints imposed by \( M \)'s quantifiers.

Definition 45 (Q-Sat SCN Case). A SCN Case \( C_M = \langle I_M, \overline{S}(M,O), N \rangle \) *satisfies all quantifiers* in \( M \) if satisfies \( Q(I'_M, X_j) \) holds for every \( X_j \in X \).

In this way we can say that an open SCN case \( C_M \) is *unsuccesful* if the goal variable \( V_F \) has been set and the constraints imposed by quantifiers are not satisfied, i.e. \( C_M \) is complete but is not Q-Sat.

### 4.3.5 SCN Primitives

SCN cases \( C_M = \langle I_M, S, X \rangle \) are *created* by identifying, in the first place, a set of SBCM instances \( I_M \) that satisfy the contextual condition \( C \) of \( M \) in a knowledge base \( K \). This is done through the operation \( I_M = \text{instantiate}(M, K, Y) \) where \( Y \) is a set of semantic bindings that constrain the SBCM instances to create (see 4.2.3).

Next, the procedure \( \text{setEvidence}(I_i, K) \) is applied to every SBCM instance \( I_i \in I_M \). If evidence feeding produces more than one SBCM instance then \( I_i \) is replaced in \( I_M \) by these new instances. For each \( I_i \in I_M \) is added a pointer to the root of \( \overline{S}(M,O) \) in \( N \), denoted \( N_i \).

This operation is denoted \( \text{createSCNCases}(M, K, Y) \) and returns a SCN case with a non-empty trials list \( I_M \), or \( \bot \) if no SBCM instance was produced by the evaluation of \( C \).

In the other hand, a SCN case \( C_M = \langle I_M, S, X \rangle \) is *updated* with information contained in a knowledge base \( K \) representing the current state of the world. In the first place, we apply the procedure \( \text{setEvidence}(I_i, K) \) to every SBCM instance \( I_i \in I_M \). Likewise, SBCM instances produced by this function replace the original ones in \( I_M \). The OBDD pointer \( N_i \) is replicated for new SBCM instances derived from \( I_i \). This operation is denoted \( \text{updateSCNCase}(C_M, K) \) and returns a (possibly empty) list of SCN cases.

The OBDD pointer \( N_i \) for the trial \( I_t = \langle Y_t, R_t \rangle \) in the SCN case \( C_M \) is updated when a controllable variable is intervened. This is done through the function \( \text{updActExec}(C_M, I_t, x_i) \). In the first place, it is checked if \( x_i \) is considered in \( \overline{S}(M,O) \), i.e. if \( X_i = \text{True} \) then
it must exist a node \( Ch_i = High(N_i) \) and if \( X_i = False \) then it must exist a node \( Ch_i = Low(N_i) \). If this is the case, \( N_i \) is pointed to \( Ch_i \) and \( x_i \) is added to \( R_i \). Otherwise, \( N_i \) is set to \( \perp \).

### 4.4 Learning

Bayesian learning algorithms can be used for refining the probabilistic distribution of the SBCM. These algorithms support missing information and can learn from partial information, i.e. observations of subsets of variables \([29, 52]\). There are algorithms for performing batch or sequential learning that can be done off-line or on-line, respectively.

#### 4.4.1 Parametric Learning

Parametric learning is made generating a set of \( N \) instances using the actual model and combining them with a set of \( M \) closed SCN cases. Considering the structure fixed, the set of instances, representing the joint probability of the process, is used for recalculating the conditional probability tables of the variables. This procedure has the following considerations:

1. The sample must include at least one instance containing \( X_i = False \) for each \( X_i \in X \),
2. if the sample doesn't contain any sample for a given set of parents \( pa(v_i) \), it is used the previous value, i.e. \( P(v_i|pa(v_i)) \), and
3. if \( pa(v_i) \) is not defined in the previous probabilistic distribution, a default value \( V_i = v_i \) is assigned, denoted \( default(V_i) \).

The first consideration avoids obtaining \( P(X_i = True|R) = 1 \) when the action \( X_i \) is being evaluated and some preconditions have not been set. If all the instances had \( X_i = True \), the probability \( P(X_i = False|R) \) would be zero for any \( R \). The second and third consideration avoids having an incomplete probabilistic distribution due to the lack of instances in the sample for calculating \( P(v_i|pa(v_i)) \). The first option is recurring to the previous probabilistic distribution, and the second is using a default value.

During the learning phase there are added new values \( v_i \) to the domain of \( V_i \) if any appears in the sample. This occurs when semantically valued variables are instantiated to a new value observed during the plan execution. In this cases, the third consideration is used for assigning a value when a condition \( pa(v_i) \) is not contained in the sample.

The size of the sample \((N + M)\) and the ratio \( N : M \) define the convergence speed of the learning algorithm. A large sample produces a more accurate result, meanwhile a small sample might omit cases necessary for calculating an entry \( P(v_i|pa(V_i)) \). Independently
of the size of the sample, the considerations made assure having a complete probabilistic
distribution on which the most common cases prevail over the less common.

On the other hand, having a set of generated instances much larger than the set of
observed instances will produce a slow learning rate. The opposite will produce a fast
convergence that will override previous experience prevailing recent observations.

4.4.2 Structural Learning

Pearl’s Inductive Causation with Latent Variables (IC*) algorithm [71, p. 52] is used for
structural learning. This algorithm is implemented in Weka [40] and produces Bayesian
networks that may contain cycles due to the identification of bidirectional edges represent­
ing confounded relations on which it is not clear the temporal precedence between
variables. Using a set of instances generated with the actual Bayesian network, the IC*
algorithm is executed in order to obtain a new structure.

From the set of edges found by the algorithm there are selected those edges $V_i \rightarrow V_j$
that satisfy the following conditions:

1. the edge is not contained in the actual structure,
2. it must connect two covariates, i.e. arcs ending or starting on a control variable
   $(X_i)$ are dismissed,
3. $V_j$ cannot be ancestor of $V_i$, i.e. $V_j \not\in \text{An}(V_i)$, and
4. the edge must not introduce a cycle in the model.

The safest selection of new edges is given by those new edges on which
$V_j \in \text{An}(V_i)$, i.e. the cause certainly precedes the effect, according to the specification given by the expert.

The selected edges are added to the structure and the parametric learning algorithm is
used for learning the parameters of the new structure. The same sample used in the IC*
run is used for running the parametric learning. In this case there are not considered
observed instances.

4.5 Summary

I introduced two types of causal models: Causal Rules and Semantic Causal Networks
(SCNs). Both of them work properly with the previously introduced world representa­
tion (ABoxes). The former, inspired in the nonmonotonic causal logic C+ [34], rep­
sents single causal relations and is used to express changes in the world state derived
from entities’ actions, exogenous events and domain regularities.
4.5. **SUMMARY**

The latter, inspired in Bayesian Causal Networks [71], represents multiple chained causal relations and is used to identify and isolate the causes, events and actions that intervene in a process aligned towards the achievement of a goal. SCN cases represent occurrence of events and actions associated to a causal network and are used for: 1) identifying viable actions, 2) calculating the effect of such actions over goals, and 3) learning through observation.

I provided a set of considerations for performing parametric and structural learning on the probabilistic representation of SBCMs using available algorithms. These considerations take advantages of additional information of the SBCM representation; for instance, structural learning will not add causal relations incoming or outgoing to/from variables identified as controllable \((X \subset V)\), or include arcs that violate the temporal precedence explicitly expressed in the original model.
Chapter 5

A Causal Theory of Artificial Intelligence Design

Just like other artificial intelligence paradigms have been inspired in theories borrowed from other disciplines, e.g. neural networks, genetic algorithms, simulated annealing, etc, this work borrows Causality and Metaphysics notions from primary philosophy for representing intelligence. Taking advantage of modern A.I. approaches to these two disciplines, I propose a theory for designing and creating intelligent artifacts.

This theory, called Causal Artificial Intelligence Design (CAID), is grounded in the ontological framework introduced in Chapter 3, providing static and dynamic representations of the world. In the other hand, Bayesian Causal Networks and the nonmonotonic causal logic C+ are used for modeling world’s dynamics through causal models in order to foresee the consequences of individual’s acts and learn from experience (Chapter 4).

In this chapter, these formalisms are used for providing a definition of intelligent entities through four main principles that guide their design and creation. This definition is further extended to intelligent organizations and agents, as well as human users.

5.1 The CAID Principles

I propose the Causal Artificial Intelligence Design (CAID) theory with the purpose of providing a methodology for designing artificial intelligent entities in terms of Causality theory. The specification obtained from the design process is used for implementing an artifact that will show a bounded intelligent behavior. We formulate four main principles, expressed in terms of classical causal and metaphysical notions, for the design and implementation of intelligent artifacts. These principles propose a solution on aspects like specification, cooperation, coordination and learning.
5.1.1 The Total Causality Principle

In the first place I’ll try to answer the question: How is an intelligent entity?. An intelligent entity can be represented by an essential form that represents the characteristics that must show the entity. The specification of such essential form can be given in terms of the elements that intervene in the actions performed by the entity. These elements are organized in the causal categories proposed by Aristotle.

**Definition 46 (Total Causality Principle (TC)).** An intelligent entity can be expressed in terms of the final, efficient, formal and material causes that intervene on its action.

Final causes are represented by those goals the entity is intended for. Efficient causes are represented by the roles it can play in its interaction with other entities. Formal causes refer to the information and knowledge that the entity knows and uses. Finally, material causes refer to the resources that the entity controls or consumes.

These causes are represented in the essential form through DL class constraints. Table 5.1 shows the roles used on the construction of these constraints (see Table 3.1).

<table>
<thead>
<tr>
<th>Cause type</th>
<th>Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficient</td>
<td>subclassOf.Agent or Agent</td>
</tr>
<tr>
<td>Formal</td>
<td>knows.Knowledge</td>
</tr>
<tr>
<td>Final</td>
<td>pursues.Goal</td>
</tr>
</tbody>
</table>

Table 5.1: Main causes representation in an intelligent entity definition.

5.1.2 The Ontological Commitment Principle

The second principle tries to answer the question: What does it motivates intelligent entity’s action?. In order to answer this question we delimit the type of goals an intelligent entity can commit with and indicate the necessity of implementing causal models to enable their achievement. The principle is enunciated as follows.

**Definition 47 (Ontological Commitment Principle (OC)).** An intelligent entity is ontologically committed to achieve goals consistent with its essential form and to act in consequence.

Essential goals identify the kind of goals the intelligent entity can achieve, delimiting its competencies. Any goal contained in an essential goal is considered compatible with its essential form. Hence, we can say that the entity is ontologically committed to goals subsumed by its essential goals.
Acting in consequence requires that the agent knows the causal models that allows achieving such essential goals. A causal model that achieves an essential goal in consequence achieves any goal subsumed by this essential goal.

Suppose we have a goal $G$ to which the intelligent entity $Ag$ is ontologically committed due to an essential goal $G_0$ such that $G \subseteq G_0$. $Ag$ must control a plan $P$ that achieves $G_0$, and $G$ in consequence. It should only be necessary that $Ag$ to be aware about a precondition that triggers plan $P$ execution for $Ag$ to start acting towards $G_0$. $Ag$ can become aware of such condition through its own sensors or through some message sent by some another agent.

In this way, cooperation between agents is based on the implicit agreement for achieving a set of goals and those goals derived from them. Coordination and negotiation, on the other hand, requires additional protocols. Coordination requires that an agent notifies others about the achievement or cancelation of goals, whereas negotiation requires an agent to be capable of calculating the effort necessary for achieving certain goal and the capacity for delegating and accepting commitments at will.

### 5.1.3 The Intentional Causation Principle

The third principle answers the question: *How does an intelligent entity coordinates with others?*. Such coordination is motivated by an ultimate goal from which are derived other subgoals pursued by the intelligent entities participating in the system. The participation of these entities is motivated by the same ultimate goal, which also provokes the creation of such entities. The principle is stated as follows.

**Definition 48** (Intentional Causation Principle (IC)). *Intelligent entity’s action is driven by goals derived from a common final cause which achievement is described in a causal model.*

As stated in the previous principle, a goal must count with causal models that enable its achievement. In general, we can refer to a causal chain consisting of at least one causal relation. Conditions represented in the causal chain, or plan that represent it, are progressively achieved as explained above. During plan execution, auxiliary subgoals are generated and achieved by other participants. These subgoals are meant by their respective executors in order to achieve the former goal. Goal decomposition in subgoals can be seen hierarchically.

In conclusion, causal dependency of actions and conditions that constitute the plan provide the coordination on the achievement of the goal towards which is oriented the plan.
Intelligent entity creation

A plan might consider the existence of an intelligent entity as a necessary condition for the achievement of the goal. Hence the creation of this entity is an action that can be considered part of the plan as well. In consequence we can enunciate the following corollary.

Definition 49 (Intentional Creation). An intelligent entity is created with a purpose derived from an upper level goal.

Creation is the action through which an agent instantiates or assemble a new entity following an specification (essential form) and using certain components or resources in order to achieve certain goal. The creator agent is the efficient cause meanwhile the created entity is the caused thing or effect. The essential form used as specification is the formal cause. Components and resources used during the instantiation or assembling constitute the material causes. The goal pursued by the creator that motivates the creation of the new entity is the final cause of the creation act.

The creation of an intelligent entity can be represented by an action rule as the following:

$$R_{create} = \{\{AgentType1(Ag_1), do(Ag_1, c), CreateAction(c), creator(c, Ag_1),
SensorType1(S_i), pursues(Ag_1, G_1), meansFor(G_2, G_1),
EssentialForm(AgentType2)\}, create, \{AgentType2(Ag_2),
creator(Ag_2, Ag_1), hasSensor(Ag_2, S_i), hasEssentialGoal(Ag_2, G_2)\}, \emptyset\}$$

In this example, $S_i$ is a sensor that $Ag_1$ knows or owns and is specified on the creation act as a parameter. Additionally, $Ag_1$ pursues a goal $G_1$ that has a subgoal $G_2$ which can be achieved by $Ag_2$ as long as in the essential form $AgentType2$ is specified $G_2$ as an essential goal.

5.1.4 The Causal Effect Accountability Principle

The fourth principle tries to answer the question: How does the intelligent entity adapt to environmental changes?. Rephrasing the question we would ask: how should the intelligent entity change its decisions when environmental conditions change? Pearl’s causality theory provides a basic component in the answer: accountability of causal effects. Based on a causal network, the intelligent entity is capable of identifying the causal effect of an intervention over certain event of interest and, based on observation, is capable of detecting variations produced by environmental changes. In our framework, control variables susceptible of intervention represent actions controlled by the owner of the SCN, meanwhile events of interest constitute goals pursued by him. I.e. the entity controlling the SCN should be capable of identifying and calculating the causal effect of its actions over its goals.

The principle is enunciated as follows.
5.1. THE CAID PRINCIPLES

**Definition 50** (Causal Effect Accountability Principle (CEA)). *Intelligent entity’s decision is based on the observation of the causal effect of its actions over the goals it pursues.*

Additionally, we introduce a new dimension to the solution by considering entity’s preferences over goals. Preferences are expressed in a *goodness scale* that goes from -10 to 10, where 10 represents the maximal goodness and -10 represents the maximal badness, being 0 neutral. In other words, the agent considers that an event qualified with a preference greater than 0 must be *achieved*, meanwhile an event qualified with a preference lower than 0 must be *avoided*. Goodness of goal $G$ is represented by $\text{goodness}(G,p)$ where $-10 < p < 10$.

In a SCN describing a process $P$, every realization $(F_i = f_i) \in F$ represents a possible way of achieving the goal $G$ to which the process is directed. The agent can express a different preference for each $F_i = f_i$, denoted as $\text{Pref}_G(F_i = f_i, p)$, where $-10 \leq p \leq 10$. Additionally, other goals can be met during the execution of $P$; let us call them *side goals*. A side goal $G'$ is identified by an annotation $\text{Ann}(Z_i = z_i)$. The preference for the achievement/avoidance of $G'$, with $\text{goodness}(G, p)$, is represented by $\text{Pref}_{G'}(Z_i = z_i, p)$.

Given that every goal pursued by the intelligent entity contributes, directly or indirectly, to its final cause, it is possible to calculate the contribution of an action to the finality of an agent.

**Definition 51** (Finality function (FFn)). Given a trial $I_i = (Y, R)$ of a SCN case $C_{M_P}$ controlled by the intelligent entity $Ag$, the contribution of the action $x_i$ to the finality of $Ag$ is given by the causal effect of $x_i$ over the possible ways of achieving the process goal, $\text{Pref}_G(F_j = f_j, p)$ for every $F_j = f_j \in F$, and its preferences over side goals. Formally,

$$\text{FFn}(I_i, x_i) = \sum_j p_j \cdot P(F_j = f_j | do(x_i), R) + \sum_k p_k \cdot P(Z_k = z_k | do(x_i), R)$$

where $\text{Pref}_{G_k}(Z_k = z_l, p_l)$ represents the preference of $Ag$ for $G_k$. $\text{FFn}(I_i, x_i)$ is given in the scale $[-10, 10]$.

The finality function allows to determine if the action $x_i$ is considered good ($\text{FFn}(I_i, x_i) > 0$) or bad ($\text{FFn}(I_i, x_i) < 0$) for the entity’s final cause. Bad actions are avoided, meanwhile good actions are eligible. Also, the finality function allows to determine the degree of goodness of $x_i$. In this way the entity is capable of choosing between those actions that contribute in a good way to its finality and rank them by their goodness. If the intelligent entity chooses the action $x_i$ with the highest $\text{FFn}(I_i, x_i) > 0$ it would be choosing the action that most likely would achieve its more preferred goal.

Updating the probabilistic distribution of $M$ with every observed case will modify the finality contribution of actions contained in $M$. External factors affecting the achievement of $G_k$ will produce variations in the causal effect $P(Z_k = z_k | do(x_i))$, producing an increment or decrement of $\text{FFn}(I_i, X_i)$. In this way, increments or decrements of the causal effect of an action $X_i$ may affect its qualification making it better or worse than others,
affecting its eligibility and in consequence modifying entity's behavior. For instance, a good action \( x_i \) can become ineligible if the entity learns that \( P(Z_k = z_k | do(x_i), R) = 0 \) for every \( \text{Pref}(Z_k = z_k, p) \) with \( p > 0 \).

5.2 Modeling Intelligent Entities through CAID

In this section we propose a characterization of Intelligent Entities from a causal perspective based on CAID Principles.

**Definition 52.** An *intelligent entity* is an entity

- defined in terms of final, efficient, formal and material causes (TC), that
- commits with and acts upon goals consistent with its essence (OC),
- coordinates with other intelligent entities pursuing own goals aligned with an ultimate goal (IC),
- and adapts its behavior based on awareness of the effect its actions has over goals it has committed to (CEA).

Beyond reactive behavior, as well known as *instinct* in animals, our definition of intelligent entities proposes individual's awareness of: its capabilities, its purpose, the consequences of its actions, and the existence of other similar individuals willing to cooperate on the pursuit of their own goals.

5.3 Modeling Intelligent Organizations through CAID

According to Aristotelian metaphysics any mental or material object can be considered an entity. We characterize an Intelligent Organization (IO) as an intelligent entity through the CAID principles as follows.

**Definition 53.** An *intelligent organization* is an intelligent entity

- defined by its organizational goals, participants, knowledge and resources (TC), that
- enables a set of processes directed towards achieving organizational goals (OC) on which its participants achieve their respective individual goals (IC),
- and adapts to environmental changes through the observation of organizational metrics (CEA).
5.3. MODELING INTELLIGENT ORGANIZATIONS THROUGH CAID

### 5.3.1 [TC] Intelligent Organization’s Main Causes

According to the Total Causality principle, an IO can be specified through the main causes that intervene on its action. Each type of cause is associated to the IO’s essential form through some constraint.

**Definition 54 (Organization Representation).** An intelligent organization is represented by an essential form, subclass of the `IntelligentOrganization` class, in terms of organizational goals, metrics, roles, knowledge, information, resources and processes.

**Organizational Goals and Metrics**

The final cause of an IO is represented by goals pursued by the organization as well as performance metrics that indicate how these goals should be met.

**Definition 55 (Organizational Goal).** An organizational goal is a goal that identifies the objective of a process carried out in an organization. It’s associated to the IO class through the constraint `pursues = G_i` where `G_i` is an individual of the `Goal` class.

**Definition 56 (Organizational Metric).** An organizational metric indicates how desirable or not is certain condition produced during the execution of an organizational process. It is denoted by a tuple `M = (C, p)` where `C` is a condition and `p` is a numeric value that indicates desirability (`0 < p ≤ 10`) or undesirability (`-10 ≤ p < 0`). It’s associated to the IO class through the constraint `observes = OrgMetric_i`, where `OrgMetric_i` is an individual of the `OrgMetric` class which is the ontological representation of the `M` tuple.

Likewise, organizational goals have an importance factor associated in the same scale than organizational metrics. This factor is represented by `importance(G) ∈ [-10, 10]`. On this way, a goal `G` is of type AVOID if `importance(G) < 0` and is of type ACHIEVE if `importance(G) > 0`. `importance(G)` represents the goodness of `G`.

**Organizational Roles**

In an intelligent organization we expect the participation of human and software agents in the different processes designed for achieving organizational goals. Specialized software agents perform certain tasks autonomously in the same way that humans do. Both competencies are described through the notion of organizational roles. Depending on the organization, is defined a taxonomy of roles that incorporates the different types of agent specialities and human roles required for modeling organizational processes.

The design of organizational processes allows to identify certain tasks that can be done autonomously by software agents. Opposite to human specialization where a person becomes expert through the repetitive execution of certain tasks, the design of software agents is specialized on a finite set of tasks that will constitute their area of expertise. In
this way we define types of agents with certain attributes that define their competencies in the organization.

**Definition 57** (Organizational Agent Role). An *organizational agent role* is defined by an essential form derived from the *SoftwareAg* class (see Section 3.1), that describes those capabilities the organization requires from an autonomous software agent.

People in the real organization participate in organizational processes requesting services and performing operations that require their knowledge and those external resources managed by them. Human specialization represented by the different positions in the real organization can be transferred to our IO design through the notion of *organizational human role*. Nevertheless, new process might require the creation of new organizational roles.

**Definition 58** (Organizational Human Role). An *organizational human role* is defined by an essential form derived from the *Person* class (see Section 3.1).

Participation of human users and software agents in the IO is denoted through the property *hasParticipant.Agent*, using constraints like \( \geq 0 \) *hasParticipant.AgentTypeX* or \( = 1 \) *hasParticipant.AgentTypeX*. The first constraint expresses possibility whereas the second one expresses the requisite of having one and only one agent of type *AgentTypeX* simultaneously in the organization.

A person participating in the organization is associated to at least one organizational role. Besides, a single person can simultaneously play multiple non-excluding roles in the organization. Mutually excluding organizational roles are declared as disjoint classes in the Organizational Ontology. Each role played by a person is represented by an instance of the respective organizational role class which description contains personal information respect the given role.

Human users are represented semantically as instances of the class *Person* or any subclass of it. Besides, every organizational role *OrgRole_i* played by him/her is expressed by the type assertion *type(user_X, OrgRole_i)*. This last assertion forces *user_X* to satisfy the constraints used to express *OrgRole_i*, which can include required information or disjointness with other *OrgRole_j*. For a detailed description of human users representation see Section 5.5.

Finally we point out that attributes and competencies of agent types and organizational roles are product of the process definition as explained in chapter 7.

**Knowledge and Information**

Knowledge and information are used by actors to perform their respective tasks in the different organizational processes. Knowledge and information is identified as the formal causes intervening in the IO.
Information is modeled in an ontology through classes and properties and information pieces are expressed through individuals of these classes. Information is associated to the IO class through the constraint $\forall \text{uses}.\text{InfType}$, where $\text{InfType}$ is the class that identifies a type of information handled in the IO.

Knowledge, on the other hand, may have multiple representations, e.g. rules, decision trees, etc. For this reason it is only required to define the types of knowledge used on the organization and provide a suitable ontological representation. Once again, classes and properties are used to model knowledge and individuals are used to express knowledge pieces. Similarly, knowledge is associated to the IO class through the constraint $\forall \text{uses}.\text{KnowledgeType}$.

Information and knowledge may reside in repositories that manage its content using the declared types. These repositories are considered resources and are declared as subclasses of the $\text{InformationRepository}$ and $\text{KnowledgeRepository}$ classes, respectively. The content of these repositories is expressed through constraints like $\exists \text{contains}.\text{InfType}$ and $\exists \text{contains}.\text{KnowType}$.

Resources

Material resources controlled or consumed by actors along organizational processes constitute the material causes of the IO. Material resources can provide services but are not considered intelligent entities as long as they only execute orders, e.g. an email server. The different types of material resources are modeled in the ontology as subclasses of the $\text{Resource}$ class and resources available in the organization are declared as instances (individuals) of these classes.

The different types of resources used in the IO are associated through constraints like $\geq 0$ $\text{consumes}.\text{ResourceType}$ or $= 1$ $\text{consumes}.\text{ResourceType}$, where $\text{ResourceType}$ is a subclass of $\text{Resource}$; the first constraint denotes possibility meanwhile the second constrains the association of a single resource of type $\text{ResourceType}$ to the IO. Actual resources are associated to the IO class through the constraint $\text{controls} = \text{Resource}_i$ or $\text{consumes} = \text{Resource}_i$, where $\text{Resource}_i$ is an instance of a subclass of $\text{Resource}$.

5.3.2 [OC] Managing Organizational Processes

In order to achieve organizational goals, an IO must provide a formal representation of those processes that lead to their achievement. Likewise, the IO must provide a mechanism for enabling these processes and improve them with respect to the corresponding organizational metrics. In our approach, we propose the use of SCNs for modeling organizational processes and the participation of a software agent for its management.

**Definition 59** (Organizational Process Specification). An organizational process $P$ directed towards the achievement of goal $G = (A, \Omega)$ is specified through a SCN

$$M_P = \langle PMA_P, \mathcal{M}, U, Q \rangle$$
where \( \mathcal{M} = (m, T, D_{TV}, C, F) \) is a SBCM, \( m = (V, G_V, P(v_i|p_a_i), X, Z) \) is a MCBCM, \( U = \emptyset, A \subseteq Ann(C), \Omega \subseteq Ann(F_i) \) for each \( F_i \in F \), and \( PMA_P \) identifies the process manager agent type.

Organizational processes are semantically represented by instances of the class \textit{Process}. They are associated to the IO by the property \textit{implements Process}. A constraint like \( \text{implements} = P \) indicates that the IO implements process \( P \).

The predicate \( \text{satisfies}(P, G) \) indicates that an instance of process \( P \) is capable of satisfying an instance of goal \( G \). A semantic description of \( P \) is obtained from causal dependencies represented in \( M_P \) and the annotations on the SCN’s variables. This description contains material resources and knowledge required in the process, its participants and the actions performed by them. SCN’s annotations use the types of resources, information, knowledge and participants associated to the IO essential form. Semantic variables used in SCN’s annotations identify the entities participating in the process, from which we can distinguish roles played by agents.

\textbf{Definition 60 (Process Role).} A process role is identified in an organizational process \( P \) by a semantic variable \( x \) if exists in \( M_P \) an annotation \( Ann(V_i = v_i) = \langle V, q \rangle \) such that \( x \in V, TypeX(x) \in q \) and \( TypeX \subseteq Agent \), i.e. a variable identifying an entity of type \textit{Agent} or any of its subclasses.

\textbf{The Process Manager}

In our approach, process management consists in the following: setting the necessary conditions for the process, monitoring process execution, and improving goal’s achievement keeping in mind organizational metrics. As can be seen, management doesn’t imply playing an active role in the process, but just observe and set the conditions necessary for the process execution. An intelligent agent, with the supervision of a human user, is on charge of such duties.

\textbf{Definition 61 (Process Manager).} A process manager \( (PMA) \) is an intelligent agent with a human user for efficient cause that manages an organizational process \( P \). Semantically,

\[
PMA \subseteq SoftwareAg \land \forall \text{ hasEfficientCause.P}.Person \land \geq 1 \text{ manages.Process}
\]

Certain conditions identified in the process definition as necessary for \( P \)'s execution can be unset at certain moment. The PMA must control organizational protocols for enabling such conditions. These protocols are represented with the same formalism used for organizational processes, just that its final condition doesn’t contain an organizational goal, but a condition in the organization. For instance a necessary condition for executing a process \( P \) would be counting with the presence of certain kind of agent.
5.3. MODELING INTELLIGENT ORGANIZATIONS THROUGH CAID

If no agent of type \( A \) is present in the organization, the PMA should initiate an organizational protocol \( P_2 \) for facilitating the presence of such agent.

Process execution’s monitoring is made through the observation of SCN cases of \( M_P \). These cases are used for building a probabilistic distribution of the process through which it will be possible to measure the efficiency of the process \( (P(F|S)) \) and predict the efficiency of some action \( \alpha (P(F|do(\alpha), S)) \). PMA’s mechanisms for observing \( M_P \) cases are described in section 5.3.4.

The PMA periodically revises the probabilistic distribution of the process looking for unsuccessful strategies. A strategy identified in the OBDD of the process has originally a probability higher than zero. If experience, represented by parametric learning on closed cases, shows that the probability of a strategy \( S \) becomes zero, \( S \) is considered unsuccessful. This would motivate re-calculate remaining strategies and calculating new ones. If there is no successful strategy after a learning phase, this is reported to the process owner.

The PMA improves process goal achievement by establishing certain conditions that constrain process execution. As long as the PMA is responsible for initiating the process execution, represented through a SCN case, it can set some semantic bindings in \( Y \) that indicate to other participants that must observe such constraints.

Finally, identification of side effects over an organizational metric \( M = \langle C, p \rangle \) consists on identifying a covariate realization \( (Z_i = z_i) \) such that \( C \subseteq Ann(Z_i = z_i), P(Z_i = z_i|do(x_j)) > 0 \) and \( x_j \in S \) where \( S \) is a valid strategy in \( M_P \); this is, identifying an effect \( Z_i = z_i \) of an action \( x_j \) required for achieving \( F \).

5.3.3 [IC] Individual goals and Organizational commitment

Each participant of an organizational process pursues an individual goal when participates on it. In the same way, the participant assumes certain commitments with the organization when accepts to participate in a process playing certain role. Assuming that we cannot force a participant to perform certain tasks, especially if it’s a human user, we can rather measure his organizational commitment instead.

If the agent \( A_1 \) satisfy an individual goal through a partial participation in process \( P \) and leaves the organization without fulfilling its obligations in \( P \), \( A_1 \) is considered uncommitted with the organization. Unfulfillment of \( A_1 \)'s obligations can cause that another agent \( A_2 \) participating in \( P \) being unable of achieving its individual goals. In order to prevent this, we propose to measure organizational commitment of agents participating in every process and limiting the participation to only those that show a good score.

**Definition 62** (Process unfulfillment). An agent \( A_i \) doesn’t fulfill a process \( P \) if decides doing the action \( x_i \) such that \( x_i \notin \overline{S}(P) \), this means that \( x_i \) is not part of an valid strategy for \( P \).
Meanwhile unfulfillment in a software agent can be attributed to malfunction, in the case of human users it can be attributed to lack of commitment with the organization or to external factors not considered in the implementation of $P$.

The proportion of process unfulfillment by human user participation in $P$ constitutes the index of *user organizational commitment*. A minimal rate $r_P$ is set for each process, indicating how critical is the process. User participation with an organizational commitment below that threshold will motivate a report to the human owner of the process, which will be responsible for calling the attention of the user or for adjusting the process.

The proportion of process unfulfillment by software agent participation in $P$ constitutes the index of *agent process efficiency*. This rate is calculated per agent class given that it is expected that agents of the same class behave on a similar way. In order to consider the agent efficiency in $P$, the agent type $T$ is included as a cause of actions carried out by software agents. In this way, monitoring of $P$ will allow to predict how likely is that an agent of type $T$ performs satisfactorily the action.

### 5.3.4 [CEA] Process Monitoring and Optimization

In order to observe and learn from an organizational process is necessary to monitor all relevant information produced from its execution. This is done by an intelligent agent who has a hole view of the process and recovers information from its participants. This information is used by the PMA for detecting and alerting of possible failures to the owner of the process, as well as for providing valuable information to its participants.

**Definition 63 (Process Monitor).** A *Process Monitor* (PMO) is an intelligent agent that monitors all the executions of an organizational process $P$. Semantically,

$$PMO \subseteq SoftwareAg \sqcap \forall monitors.Process$$

Our definition of *process monitoring* comprehends: observing the intermediate states of the process and reporting each observed case to the PMA of the process. In order to observe those intermediate states, the PMO must be aware of the conditions that initiate an instance of the process $P$. In the same way, the PMO can participate in the process as intermediary or inquiring other participants about the outcome of their actions. The PMO must detect the ending of the process in order to inform the PMA. Before sending the case the PMO complements missing information.

The PMO participates actively in the process, making decisions that affect the result of the process. For instance, choosing which agent class or which resource would better fit in an instance of the process, given the particular characteristics of the case.
5.3.5 A formal definition of Intelligent Organizations

I propose a formal representation of an Intelligent Organization based on our ontological framework.

**Definition 64 (CIO).** A Causal Intelligent Organization (CIO) is represented by a tuple $O = (\mathcal{G}, \mathcal{M}, \mathcal{A}, \mathcal{K}, \mathcal{R}, \mathcal{P}, \mathcal{I})$ where

- $O \subseteq \text{IntelligentOrganization}$ is an essential form,
- $\mathcal{G}$ is a set of organizational goals pursued by $O$,
- $\mathcal{M}$ is a set of organizational metrics observed by $O$,
- $\mathcal{A}$ is a set of agent types and organizational roles participating in $O$,
- $\mathcal{K}$ is a set of knowledge and information types used in $O$,
- $\mathcal{R}$ is a set of resource types controlled or consumed in $O$,
- $\mathcal{P}$ is a set of organizational process implemented by $O$, and
- $\mathcal{I}$ is a set of importance factors for $G \in \mathcal{G}$ considered by $O$,

such that

- for each goal $G \in \mathcal{G}$ exists at least one process $P \in \mathcal{P}$ and a single $\text{importance}(G) \in \mathcal{I}$, and
- $\mathcal{A}$ contains at least one agent type $\text{PMA}_P \subseteq \text{manages}.P$ and at least one agent type $\text{PMO}_P \subseteq \text{monitors}.P$ for each process $P \in \mathcal{P}$.

5.4 Modeling Intelligent Agents through CAID

I propose a definition of intelligent software agents participating in an intelligent organization based on CAID principles.

**Definition 65.** An intelligent agent is an intelligent entity

- defined by its essential goals, actions, knowledge and resources (TC), that
- participates in organizational processes (IC) in order to achieve its essential goals (OC)
- and adapts its behavior based on the effects of its actions in the organization (CEA).

Next are described in more detail how the CAID principles are applied to the development of an Intelligent Agent.
CHAPTER 5. A CAUSAL THEORY OF AI DESIGN

5.4.1 [TC] The Agent Class

An intelligent agent is causally defined by the main causes that intervene on its action. This definition has an Agent Class as output, which serves as blueprint for developing an implementation of such intelligent agent.

**Definition 66 (Agent Class).** An agent class is an essential form describing a kind of intelligent agent and is defined semantically as subclass of the SoftwareAg class, in terms of its goals, actions, knowledge, information, resources and plans.

An instance of the agent class \( C \) is compliant with the organizational agent role \( T \) if \( T \subseteq C \).

**Essential Goals**

An Intelligent Agent (IA) is designed with a specific purpose, which makes it more efficient than a general-purpose one. This purpose is represented by goals associated to the agent class to express that an instance of this agent class will pursue these kind of goals on any chance it has. On the other hand, this definition delimits the competencies or specialization of the IA.

**Definition 67 (Essential Goal).** An essential goal \( G = (A, \Omega) \) is a goal that an intelligent agent will pursue on any chance it has. It is associated to the agent class representing this kind of intelligent agent through the constraint \( \text{pursues} = G_i \), where \( G_i \) is an instance of the Goal class.

An essential goal is inherited by all the subclasses of the agent class on which is defined. Finally, it is important to mention that an agent can pursue other goals as means for achieving an essential goal. These subgoals are given by external conditions like the design of an organizational process on which it must participate in order to achieve an essential goal.

**Actions**

Being cause of change, actions also define an Intelligent Agent. In a controlled context like an organization, the set of actions an IA can perform can be identified in the organizational processes specification.

**Definition 68 (Organizational agent action).** Action \( \alpha_i \) is performed by an agent capable of playing the role AgentRole if exists an organizational process specification \( M_P \) containing two annotations \( Ann_\alpha \) and \( Ann_T \), such that \( \{\text{do}(\text{?ag}, \text{?act}), \alphaType(\text{?act})\} \in Ann_\alpha, \alphaType \subseteq \text{Action}, \text{AgentRole}(\text{?ag}) \subseteq Ann_T \) and \( \text{AgentRole} \subseteq \text{Agent} \).

The ability of agents playing the role AgentRole for executing actions \( \alphaType \) is represented through the constraint \( \text{AgentRole} \subseteq \geq 0 \text{do.}\alphaType \).
The particular way on which an agent perform such actions (the internal details), are not relevant for the organization, but it is necessary to have a common name for coordinating the process with other participants.

**Information and Knowledge**

Information and Knowledge are used by an IA as formal causes of the actions it performs. In our framework, an IA needs to know certain information or knowledge piece in order to use it. The mechanism for acquiring such information/knowledge is discussed latter. Types of information or knowledge expressed in organizational processes and known by an IA must be declared and associated to the IO.

The property \( \text{knows}.\text{Form} \) associated to the class \text{Agent} allows to express information or knowledge possession. For instance, an IA knows by default an information or knowledge piece \( K \) if its agent class contains the constraint \( \text{knows} = K \). Similarly, a constraint like \( = 2 \text{knows}\.\text{KnowledgeType} \) expresses the necessity of knowing exactly two different pieces of information of type \text{KnowledgeType}.

The type of information and knowledge an IA must know to participate in an organizational process can be obtained from the process specification.

**Definition 69** (Organizational agent knowledge). The knowledge or information type \( K \) is known by an agent of type \text{AgentType} if exists at least one organizational process specification \( M_P \) containing three annotations \( \text{Ann}_A, \text{Ann}_T \) and \( \text{Ann}_K \), such that \( \text{AgentType}(a) \sqsubseteq \text{Ann}_A, K(k_i) \sqsubseteq \text{Ann}_T \) and \( \text{knows}(a, k_i) \sqsubseteq \text{Ann}_K \).

**Resources**

Resources are used by an IA as material causes of its actions. In our framework, an IA needs to possess or control certain resource in order to use it. Types of resources expressed in organizational processes and possessed/controlled by an IA must be declared and associated to the agent class.

Properties \( \text{has}.\text{Material} \) and \( \text{controls}.\text{Resource} \) associated to the class \text{Agent} allows to express possession and control of a resource, respectively. For instance, an IA controls by default a resource \( R \) if its agent class contains the constraint \( \text{controls} = R \). Similarly, a constraint like \( = 2 \text{has}.\text{MaterialType} \) expresses the necessity of having exactly two different resources of type \text{MaterialType}.

Similarly to information and knowledge, types of resources owned or controlled by a type of IA can be obtained from the organizational processes specifications.

**Definition 70** (Organizational agent resource). At least one resource or component type \( R \) is owned by an agent of type \text{AgentType} if exists at least one organizational process specification \( M_P \) containing three annotations \( \text{Ann}_A, \text{Ann}_T \) and \( \text{Ann}_R \), such that \( \text{AgentType}(a) \sqsubseteq \text{Ann}_A, R(r_i) \sqsubseteq \text{Ann}_T \) and \( \text{has}(a, r_i) \sqsubseteq \text{Ann}_R \). Similarly, this resource is controlled by agents of type \text{AgentType} if \( \text{controls}(a, r_i) \sqsubseteq \text{Ann}_R \).
5.4.2 [OC] Achieving essential goals

An Intelligent Agent (IA) pursues certain essential goals which achievement is guided by plans. Essential goal instances trigger the execution of plans designed for achieving such goals.

An IA is motivated to participate in an organizational process if its participation contributes to achieving an own essential goal. Such participation is modeled as a plan representing a view of the process that comprises the actions performed and events observed by a participant, identified by a process role. The agent can initiate the process execution or can be motivated to participate on it.

Essential plans

An IA must be capable of elaborating or must know plans for achieving its essential goals.

**Definition 71 (Essential Plan Specification).** An essential plan $P$ directed towards the achievement of an essential goal $G = \langle A, \Omega \rangle$ of the Agent class $Ag$ is specified through a SCN

$$M_P = \langle Ag, M, U, Q \rangle$$

where $M = \langle m, T, D_{TV}, C, F \rangle$ is a SBCM, $m = \langle V, G_V, P(v_i|p_a), X, Z \rangle$ is a MCBCM, $X \neq \emptyset$ describes actions controlled by $Ag$, $A \subseteq \text{Ann}(C)$, and $\Omega \subseteq \text{Ann}(F_i)$ for each $F_i \in F$.

Essential plans are represented semantically as instances of the class Plan. An IA is said to control a plan $P$ if its agent class contains the constraint $\text{controls} = P$.

Process views

Depending on the role played by an agent in a process, the set of events it can observe and actions it can control vary. These actions and events, along with the causal relations between them, constitute a view of the process for an agent class.

**Definition 72 (Process View).** The partial view of the organizational process $P$ intended for achieving organizational goal $G_O = \langle A_O, \Omega_O \rangle$ perceived by an agent of type $Ag$ playing role $R$ on it, is called process view and is represented as an essential plan directed towards the achievement of the essential goal $G_A = \langle A_A, \Omega_A \rangle$ specified by:

$$M_P^R = \langle Ag, M, U, Q \rangle$$

where $M = \langle m, T, D_{TV}, C, F \rangle$ is a SBCM, such that $A_O \subseteq A_A$ and $\Omega_O \subseteq \Omega_A$.

Additionally, organizational metrics are identified in a process view as organizational side effects.
Definition 73 (Organizational side effects). An organizational metric $M = (C, p)$ is identified in the organizational process view $M_P^R$ if exist an annotation $Ann_C$ such that $C \subseteq Ann_C(Z_i = z_i)$. $M$ is represented by an organizational side effect $SE_M = (S, s)$ on which $S = (Z_i = z_i)$ and $s = \text{importance}(G)$.

Given that a process view is an essential plan, the participation of an agent in the organizational process $P$ is initiated as soon as $M_P^R$'s contextual condition holds.

Triggering and executing plans

Periodically, an IA checks if it is possible for him to achieve one of its essential goals. To do so, it evaluates the precondition of each essential goal $G = (A, \Omega)$ in the actual world state $W$, i.e. $W \models A$. If this evaluation produces any match, the essential goal is said to be instantiated. Semantic bindings $p_i$ obtained from this evaluation are used for creating a goal instance $I_G = (G, p)$. A new instance of an essential goal is considered by the IA as an opportunity for accomplishing its purpose.

The goal instance $I_G$ is used for starting the execution of an essential plan $M_P$ such that $satisfies(M_P, G)$. The execution of $M_P$ is represented by a SCN case $C_{M_P}$ which we will denote in the following as $I_P$, also called plan instance. $I_P$ is used for paying attention to events relevant in the plan execution and for deciding which action to perform in order to reach the final condition of the plan, which in consequence will satisfy $G$.

An IA can adopt a goal instance $I_G = (G, p)$, and commit to its achievement, if $G \subseteq G_E$ where $G_E$ is an essential goal of the agent. Goal commitment is translated into plan instantiation, i.e. instantiating some $M_P$ such that $satisfies(M_P, G)$.

All previous definitions given in chapter 4 for SCN cases apply to plan instances. In this way we can talk about complete, (in)viable and stalled plan instances.

5.4.3 [IC] Enabling plans

During plan execution, other auxiliary plans can be initiated for setting certain conditions necessary for the execution of the original plan, providing an automatic subgoaling mechanism.

Enabling stalled plans

In section 4.3.4 I described how to determine the viability of actions and determined the set of preconditions that must be observed for determining such viability. This calculation can be made even with unobserved preconditions. Now I will discuss how to determine if such conditions can be set by the agent.
Assuming that the cost of enabling an uncertain action \( x_1 \) is higher than the cost of executing an enabled action \( x_2 \), the agent will prefer \( x_2 \) over \( x_1 \). The lack of enabled actions lead to an impasse of the plan execution represented by the SCN case.

**Definition 74** (Stalled Plan Execution). A plan execution represented by the SCN Case \( C_M = (I, \overline{S}(M, O), N) \) is considered **stalled** if it doesn’t have any enabled actions but counts with uncertain actions.

A stalled plan instance can be active again if the agent gathers information or sets those conditions that avoid determining viability of uncertain actions. This is done through the execution of some other plan.

The definition of missing conditions (Def. 39) give us a set of realizations that might contain a condition \( (Z_i = z_i) \) that is caused (directly or indirectly) by other missing conditions \( (C_{\text{Mis}(X_i)}) \) in conjunction with \( R \). Given that setting \( C \) will cause \( Z_i = z_i \), we can remove this realization from \( \text{Mis}(X_i) \). Starting with a tentative set of missing conditions containing realizations of ancestors of \( X_t \) not contained in \( R \), and making this elimination iteratively we can obtain a **minimal set of missing conditions**.

Figure 5.1 illustrates a stalled trial where the execution of \( X_1 \) is considered inviable given the current evidence \( R = \{Z_1 = A, Z_2 = \text{True}, Z_3 = \text{True} \} \) and \( P(X_1 = \text{True}|Z_1 = A, Z_2 = \text{True}, Z_3 = \text{True}, Z_4 = \text{True}, Z_6 = \text{True}) \geq \tau_a \). The minimal set of missing conditions is \( \text{Mis}(X_1) = \{Z_4 = \text{True}, Z_7 = \text{True}\}; Z_6 = \text{True} \) is eliminated from an initial \( \text{Mis}(X_1) \) given that the subset \( C = \{Z_7 = \text{True}\} \) makes \( P(Z_6 = \text{True}|Z_7 = \text{True}) > 0 \).

![Figure 5.1: Missing conditions for the execution of an action.](image-url)

The semantic description of missing condition is used for identifying those SCN cases that can set the missing condition.

**Definition 75** (Auxiliary plan). A SCN Case \( C_{M_2} \) is capable of setting the condition
Z_i = z_i of the trial I_i of the SCN Case C_M if

$$Ann_M(Z_i = z_i) \subseteq Ann_{M_2}(F_i = f_i)$$

(5.1)

for each \((F_i = f_i) \in F_{M_2}\). This relation is represented as \(enables(C_{M_2}, C_M, Z_i = z_i)\).

The actual execution of \(C_M\) will require the generation of an auxiliary goal to set \(Z_i = z_i\).

Definition 76 (Auxiliary goal). Given a trial \(I_i = (Y, R)\) of a stalled plan instance \(C_M\), the auxiliary goal instance \(G_2 = (\langle \top, Ann(Z_j = z_j) \rangle, Y_{P_2})\) will trigger the execution of \(C_{M_2}\) that in turn will set the missing condition \(Z_j = z_j \in Mis(X_i)\).

\(Y_{P_2}\) represents the set of semantic bindings of \(C_M\) but using the equivalent variable names of \(M_2\). These equivalences are obtained through the calculation of \(Ann_M(Z_i = z_i) \subseteq Ann_{M_2}(F_i = f_i)\).

The relation between the goal \(G_1\), being enabled through \(C_{M_2}\), and the auxiliary goal \(G_2\) is represented by the relation \(meansFor(G_2, G_1)\).

5.4.4 [CEA] Agent adaptation

An IA immersed in an IO is obliged to act according to its design purpose and finds an opportunity for doing so in the organization. The IA will be capable of making decisions of benefit for the organization and for itself as long as it becomes aware of how its actions impact organizational goals. Organizational and environmental conditions might affect the efficacy of its action forcing the IA to change its behavior in order to continue fulfilling its purpose.

Plans controlled by an IA identify the actions controlled by the agent. An instance plan, on the other hand, provides an idea of the current conditions in the world. Thanks to this it is possible to identify viable actions for the agent in a given context.

Qualifying causal effects

Action causal effect is additionally qualified by preferences of the agent and the organization on side effects produced during the plan execution. The agent qualifies the different ways of achieving a goal \(G\) in \(P\) defining a preference for each \((F_i = f_i) \in F\) of \(M_P\).

Definition 77 (Essential Goal Preference). An essential goal preference \(EGP = \langle \{F_i = f_i\}, p\rangle\) represents the agent preference \(Pref_G(F_i = f_i, p)\) for \((F_i = f_i) \in F\).

Organizational goals and metrics can be identified in \(M_P\) as well. An organizational goal \(G_O = \langle A_O, \Omega_O \rangle\) is identified in \(P\) if \(M_P\) contains an annotation \(Ann(Z_i = z_i)\) such that \(\Omega_O \subseteq Ann(Z_i = z_i)\). The importance of the organizational goal is represented in
CHAPTER 5. A CAUSAL THEORY OF AI DESIGN

$M_P$ with the preference $\text{Pref}_{GO}(Z_i = z_i, \text{importance}(G_O))$. An organizational metric
$M = \langle C, p \rangle$ is identified in $P$ if $M_P$ contains an annotation $\text{Ann}(Z_i = z_i)$ such that $C \subseteq \text{Ann}(Z_i = z_i)$. The resulting organizational preference is given by $\text{Pref}_M(Z_i = z_i, p)$.

**Definition 78** (Organizational side effect). An organizational side effect represents an organizational preference $\text{Pref}_{GO}(Z_i = z_i, p)$ or $\text{Pref}_M(Z_i = z_i, p)$ identified in the plan specification $M_P$ controlled by an agent $Ag$, and is denoted $SE_{M_P} = \langle Z_i = z_i, p \rangle$.

In this way, the finality function (Definition 51) can be modified for considering organizational preferences in the agent decision.

**Definition 79** (Organizational Finality Function (OFFn)). Given a trial $I_i = \langle Y, R \rangle$ of a SCN case $C_{M_P}$ where $P$ is an essential plan controlled by the intelligent agent $Ag$, the contribution of the action $x_i$ to the finality of $Ag$ is given by the action causal effect of $x_i$ over every $\text{EGP}_j = \langle \{F_j = f_j\}, P_j \rangle$ for $(F_j = f_j) \in F$ identified in $M_P$ and every organizational side effect $SE_{M_P}^k(Z_k = z_k, p_k)$. Formally,

$$\text{OFFn}(I_i, X_i) = \sum_j p_j \cdot P(F_j = f_j | do(x_i), R) + \sum_k p_k \cdot P(Z_k = z_k | do(x_i), R)$$

$\text{OFFn}(I_i, X_i)$ is given in the scale $[-10, 10]$.

**Learning through experience**

Keeping track of every execution of a process or a plan will allow to maintain probabilistic distributions updated with environmental response. Consistent patterns will modify the causal effect of certain actions and in consequence its desirability. On this way, the experience will teach the agent to modify its behavior under different conditions.

The probabilistic distribution of each process or plan is calculated from a set of cases (plan instances), which summarization according $G_V$ produces $P(v_i, pa_i)$. The number of cases used for generating such probabilistic distribution will produce a learning rate that must be controlled for each agent. This number is calculated with base on the number of possible realization combinations of the model and with the frequency of observed cases.

**5.4.5 A formal definition of Intelligent Agents**

I propose a formal definition of an Intelligent Agent based in our ontological framework.

**Definition 80** (CIA). A Causal Intelligent Agent (CIA) is represented by a tuple $Ag = \langle G, \bar{A}, \bar{R}, \bar{P}, \bar{S}, \tau_o, \bar{GP} \rangle$ where

- $Ag$ is an agent class,
5.5. Modeling Human Users through CAID

- $\overline{G}$ is a set of organizational goals pursued by $Ag$,
- $\overline{A}$ is a set of actions that can be done by $Ag$,
- $\overline{K}$ is a set of knowledge and information types known by $Ag$,
- $\overline{R}$ is a set of resource types controlled or owned by $Ag$,
- $\overline{P}$ is a set of essential plans controlled by $Ag$,
- $\overline{S}$ is a set of organizational side effects observed by $Ag$,
- $\tau_a$ is the action confidence threshold observed by $Ag$, and
- $\overline{GP}$ is the set of essential goal preferences of $Ag$,

such that

- for each goal $G \in \overline{G}$ exists at least one essential plan $P \in \overline{P}$ that satisfies $G$,
- for each final condition $F_i = f_i \in F$ of every $P \in \overline{P}$ exists an essential goal preference $EGP \in \overline{GP}$, and
- $\overline{A}$ contains every agent action identified in at least one plan $P \in \overline{P}$.

5.5 Modeling Human Users through CAID

Human beings are intelligent entities by definition. We can even claim that intelligence is defined having human mind as the model. In this section I define the role that plays a human being when participates in an intelligent organization and provide a semantic representation for this purpose.

**Definition 81.** A human user is an intelligent entity

- defined by its organizational roles, goals associated to them, knowledge and controlled resources (TC), that
- freely acts towards his/her own goals and goals derived from roles he plays in the organization (OC),
- is committed to organizational goals aligned with its own individual goals (IC),
- and changes its participation according to organizational reinforcement, product of his performance in the system (CEA).

Next is provided a semantic representation of the participation of human users in the organization. Additionally, section 6.4.2 describes the computational implementation of a software agent that acts in the organization on behalf of the human user.
5.5.1 A formal definition of Human Users

The human user is represented semantically to express its competencies and its participation in organizational processes through the notion of organizational roles. I extend the definition of organizational roles (Def. 58) borrowing some notions used on the definition of Intelligent Agents.

**Definition 82** (Organizational Human Role (cont.)). An organizational human role is represented by a tuple \( OR = (A, K, R, P) \) where

- \( OR \subseteq Person \) is an essential form,
- \( A \) is a set of actions that \( OR \) can do,
- \( K \) is a set of knowledge and information types known by \( OR \),
- \( R \) is a set of resource types controlled or owned by \( OR \), and
- \( P \) is a set of process views controlled by \( OR \),

such that \( \bar{A} \) contains all action roles associated to \( OR \) in every process view \( P \in \bar{P} \).

Now I define human users in terms of its participation in the organization.

**Definition 83** (Human User). A human user is represented by an individual of the class \( Person \), identified by a unique URI, and typed with at least one organizational human role. Semantically is expressed as \( HumanUser = Person \sqcap \geq 1 \text{type.OrgRole} \), where \( OrgRole \) is any subclass of \( Person \).

A human user expressed on this way must satisfy all the constraints imposed by the definition of every organizational role it plays. A consistency checking test allows to detect anomalies on an actual human user description.

5.6 Summary

I presented a theory called *Causal Artificial Intelligence Design (CAID)* constituted by four main principles that guide the design and implementation of intelligent entities. This theory is grounded in the previously introduced ontological framework and causal models.

CAID principles refers to aspects of specification, cooperation, coordination and learning of intelligent entities. The Total Causality (TC) principle establishes that the design of an intelligent entity must consider the four types of causes that intervene in its action. The Ontological Commitment (OC) principle states that any intelligent entity will try to achieve those goals that ontologically defines it using causal models. The
5.6. SUMMARY

Intentional Causality (IC) principle proposes the existence of a single final cause that guides the action of intelligent entities participating in a system. Finally, the Causal Effect Accountability (CEA) principle establishes that entity's decisions are based in the awareness and observation of the effects of its actions over the pursued goals.

I introduced a causal definition of intelligent entities based on CAID principles. This definition was extended for characterizing intelligent organizations, intelligent agents and human users in an organizational context.

In each definition CAID principles are applied distinctly. An Intelligent Organization, being an abstract concept that exists thanks to the interaction of multiple entities, is defined in terms of its components. An Intelligent Agent, representing an artifact existing actually in the system, are defined by its characteristics and capabilities. And a human user, being a representation of an actual entity existing outside the system, is defined in terms of the roles he/she plays in a process and the obligations derived from them.

The definition of an intelligent organization comprises: 1) those common goals pursued by its members, 2) metrics used for qualifying the achievement of such goals, 3) organizational processes defined for achieving such goals, 4) the type of agents (human or software) participating in these processes, and 5) the information, knowledge and resources used by these agents in its processes.

An intelligent agent is defined mainly by: 1) essential goals that represent the purpose of its design, 2) plans that coordinate its actions towards the achievement of its goals, 3) the actions that perform in these plans, 4) information, knowledge and resources that uses in these actions, and 5) other goals that must observe as consequence of its participation in an organization.

A human user is defined in terms of: 1) the roles (s)he can play in the organization, 2) actions (s)he must perform to play such roles, 3) information, knowledge and resources (s)he uses in the execution of such actions, and 4) those goals (s)he commits with as consequence of the roles (s)he plays.

The ontological framework introduced in Chapter 3 is used for formalizing these definitions. Likewise, semantic causal networks are adapted for representing organizational processes and agent plans.
In this chapter I describe a computational implementation of Causal Intelligent Agents [18]. The *Causal Agent* architecture, as we call it, is an extension of a BDI architecture that incorporates notions and principles used in the definition of a Causal Intelligent Agent. Figure 6.1 illustrates this architecture.

The Causal Agent provides a custom implementation of the BDI inference model. Essential goals and plans, as well as other characteristics of the agent, are associated to the
agent class and are used by the inference engine for satisfying its ontological definition, i.e. for doing what the agent is meant to do. A semantic layer is used for representing beliefs and perceptions, and for incorporating external events, interpreting shared definitions and activating actuators. Semantic schemas, including the agent definition, are obtained through a client component connected to a centralized repository. Valid actions’ causal effect over goals and side effects are used for selecting the best action to execute.

The combination of Bayesian, causal and semantic representations allows integrating functionality proved useful in other agent architectures (like subgoaling), and additionally provides a new sense of awareness to the agent. Semantical relations among plans are used for implementing a subgoaling mechanism. The Bayesian nature of plan representation allows to perform parametric learning. Agent class definitions allows to determine what kind of agents would be useful in a plan execution, and justifies instantiation of agents by agents.

Causal agent architecture and functionality is detailed along this chapter. I conclude presenting three main types of agents implemented with this architecture.

### 6.1 A Causal BDI inference engine

In this section it is shown how the elements described in section 5.4 are applied in the inference process of the Causal Agent. It is proposed a representation of beliefs, desires and intentions in terms of goals, plans and actions. Likewise, operations described in semantic causal models and CAID principles are used for providing a procedural implementation of a goal-driven BDI inference model.

#### 6.1.1 Beliefs, Desires and Intentions

Agent’s beliefs about the current state of the world, denoted \( CurrBel \), are represented by an ABox containing entity descriptions encoded using schemas defined in the Organizational Ontology (OO). Definitions contained in the OO are used to infer information from explicit entity descriptions.

Agent’s desires are represented by goal and plan instances contained in agent’s beliefs. Additionally to previous definitions, a goal instance has a status identifying its stage. The goal’s property \( \text{status} = \{ \text{ACTIVE}, \text{INACTIVE}, \text{SATISFIED}, \text{UNSATISFIED} \} \) is used for this purpose. Active goal instances are those being pursued currently. Inactive goal instances are those kept in hold meanwhile a sub-goal derived from it is currently tried. A satisfied goal instance is one that was achieved through the execution of some plan, or incidentally. An unsatisfied goal instance is one for which all the known plans failed and the final condition \( (\Omega) \) was not reached.

Plan instances identify possible action routes for achieving current goal instances. A
6.1. A CAUSAL BDI INFECTION ENGINE

plan instance $I_P$ intended for achieving a goal instance $I_G$ is denoted $intendsFor(I_P, I_G)$. From the set of plan instances $I_P$ intending for $I_G$, a single plan instance is maintained active, the rest of them are inactive. Complete SCN cases represent complete plan instances, meanwhile inviable SCN cases are considered failed plan instances.

Goal and plan life cycles are illustrated in Figure 6.2; labels on edges denote the inference stage on which the status changes.

![Figure 6.2: Goal and plan life cycles.](image)

Finally, agent’s intentions or options are represented by valid actions identified in active plan instances (see Def. 42).

### 6.1.2 Belief Revision

Belief revision receives for input two ABoxes: one representing perceptions ($Perc$) and another containing the current set of beliefs ($CurrBel$). As result of belief revision, $CurrBel$ is updated by causal rules incorporating information from $Perc$ into $CurrBel$ and updating $CurrBel$ with conclusions derived from new information.

The perception set, $Perc$, was previously filled by the effect of agent sensors (see Section 6.2.3). Perception incorporation is made through the execution of a finite set of exogenous causal rules $R_E = \{A, A', \delta^+, \delta^-\}$, the external ABox is represented by $Perc$ and the world state is represented by $CurrBel$. Effects denoted by $\delta^+$ are applied on $CurrBel$.

Belief revision is done using a finite set of immediate causal rules $R_I = \{A, \delta^+, \delta^-\}$, where the world state is represented by $CurrBel$, which is affected by the effects of the rule, $\delta^+$ and $\delta^-$. This step is repeated until none immediate rule is triggered.

OO definitions are applied automatically, i.e. each time that $CurrBel$ is modified. This process is described in the Procedure ReviseBeliefs($Perc$, $CurrBel$, $R_E$, $R_I$).

Every exogenous rule identifying relevant perceptions for an essential plan $P_i$ is associ-
CHAPTER 6. THE CAUSAL AGENT ARCHITECTURE

input: An ABox perc containing perceptions, an ABox currBel containing beliefs, a set of exogenous causal rules \( R_E \) and a set of immediate causal rules \( R_I \).

<table>
<thead>
<tr>
<th>begin</th>
</tr>
</thead>
<tbody>
<tr>
<td>foreach ( R_i \in R_E ) do</td>
</tr>
<tr>
<td>applyExRule(currBel, ( R_i ), perc);</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>repeat</td>
</tr>
<tr>
<td>triggered ( \leftarrow \bot );</td>
</tr>
<tr>
<td>foreach ( R_i \in R_I ) do</td>
</tr>
<tr>
<td>triggered ( \leftarrow ) triggered ( \lor ) applyImRule(currBel, ( R_i ));</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>until ( \neg ) triggered;</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

Procedure ReviseBeliefs(Perc, CurrBel, \( R_E \), \( R_I \))

associated to the plan’s DL instance through the assertion requiresPerceptionRule(\( P, ER_i \)), where \( ER_i \) is of type ExogenousRule which is the semantical representation of an exogenous causal rule. Similarly, each immediate rule used for updating agent beliefs is associated to the plan through the assertion requiresBRRule(\( P_i, IR_i \)), where \( IR_i \) is of type ImmediateRule which is the semantical representation of an immediate causal rule.

6.1.3 Options Generation

Once the set of beliefs is updated and relevant exogenous information has been incorporated, current goal and plan instances are revised. Then, new possible goal instances and their respective plan instances are generated.

The options generation function receives for inputs the ABox CurrBel, the list of current goal instances \( \overline{I_G} \) and the list of plan instances \( \overline{I_P} \). It uses CurrBel and the list of essential goals \( \overline{G} \) and essential plans \( \overline{P} \) of the agent for updating \( \overline{I_G} \) and \( \overline{I_P} \).

Plan and goal revision

In the first place, evidence is fed from CurrBel into every \( I_P \in \overline{I_P} \) intended for active goal instances. This step uses the function setEvidence() (see section 4.3.5). If a plan instance is incomplete or inviable, it is removed from \( \overline{I_P} \) and is stored apart in a relation called toLearn (see Section 6.3.2).

Next, the final condition (\( \Omega \)) of every active goal instance \( I_G \) is evaluated. Positive evaluations change the goal instance status to SATISFIED. In the other hand, if \( \overline{I_P} \) doesn’t
have any plan instance for $I_G$, denoted $\overline{I_P}(I_G)$, the status of $I_G$ is set UNSATISFIED.

Finally, (un)satisfied goal instances as well as their respective plan instances are removed from $\overline{I_G}$ and $\overline{I_P}$, respectively. This process is detailed in the Procedure PlanGoalRevision($\text{CurrBel}$, $\overline{I_G}$, $\overline{I_P}$).

```plaintext
input: An ABox currBel containing beliefs, a set of goal instances $\overline{I_G}$ and a set of plan instances $\overline{I_P}$.

begin
  foreach $I_P \in \overline{I_P}$ do // feed evidence in active plan instances
    if $I_G$.status = ACTIVE then
      setEvidence($I_P$, currBel);
      if isComplete($I_P$) \lor isInvisible($I_P$) then
        tolearn ← tolearn \cup I_P;
      end
      $\overline{I_P} \leftarrow \overline{I_P} \setminus I_P$;
    end
  endforeach
  foreach $I_G \in \overline{I_G}$ do // check goal instances satisfaction
    if $I_G$.status = ACTIVE then
      if evalQuery(currBel, $I_G$, $\Omega$, $I_G$, $p$) \neq \bot then
        $I_G$.status ← SATISFIED;
      else if $\overline{I_P}(I_G) = \emptyset$ then
        $I_G$.status ← UNSATISFIED;
      end
    end
  endforeach
  foreach $I_G \in \overline{I_G}$ do // clean up (un)satisfied goal instances
    if $I_G$.status \in \{ SATISFIED, UNSATISFIED \} then
      $\overline{I_G} \leftarrow \overline{I_G} \setminus I_G$;
      $\overline{I_P} \leftarrow \overline{I_P} \setminus \overline{I_P}(I_G)$;
    end
  endforeach
end
```

Procedure PlanGoalRevision($\text{CurrBel}$, $\overline{I_G}$, $\overline{I_P}$)

Goal and plan instantiation

Next, the agent evaluates the precondition ($A$) of every essential goal $G = (A, \Omega) \in \overline{G}$ over $\text{CurrBel}$ to generate new essential goal instances according to its current beliefs. A new goal instance $I_G$ is stored if it doesn’t exist any other goal instance $I_{G_2}$ such that $I_G \subseteq I_{G_2}$. New goal instances are active by default.
CHAPTER 6. THE CAUSAL AGENT ARCHITECTURE

For each new goal instance $I_G = \langle G, p \rangle$ it's identified a set of plans $\overline{P'} \subseteq \overline{P}$ such that $P_i \in \overline{P'} \land \text{satisfies}(P_i, G)$. For each $P_i$ is generated a set of new plan instances through the function $\text{instantiate}()$. Every new SCN case $I_{P_i}$ intended for achieving $I_G$ is added to $I_P$. Then the function $\text{setEvidence}()$ is used for feeding information from $\text{CurrBel}$ into each $I_{P_i}$.

If no plan instance is generated for $I_G$, this situation is reported to the efficient cause of the agent and the new instance goal is not added. Otherwise, the new goal instance is added and the list of plan instances intended for it are added. Previously, plan instances are sorted in ascending order according to the formula $|X| - |X'|$, where $X$ is the set of controllable actions in the plan $P$ and $X'$ is the set of directly controllable actions in $P$.

In this way there are privileged those plans with least (indirect) controllable actions.

This process is described in the Procedure $\text{GoalPlanInstantiation}(\text{CurrBel}, \overline{G}, \overline{P}, I_G, I_P)$.

6.1.4 Options Filtering

The options filtering function receives for input the updated list of goal instances $\overline{I_G}$ and plan instances $\overline{I_P}$ as well as the $\text{CurrBel}$ ABox. It selects the first plan instance for each $I_G \in \overline{I_G}$, and returns the list of valid actions contained in them.

For each $I_P$ selected for an active instance goal $I_G$, there are identified potential actions. Then they are filtered from potential to viable and then valid. If there are valid actions, the list of actions is returned. Invalid actions are used for discarding unnecessary trials.

Details of implementation are shown in the Function $\text{filterOptions}(\overline{I_G}, \overline{I_P})$. The function $\text{getPotentialActions}(I_P)$ uses the OBDD pointer $N_i$ for obtaining potential actions $\alpha = \langle I_P, I_i, x_i \rangle$ from each trial $I_i$ of $I_P$. The function $\text{calcMissing}(\alpha)$ determines the sets of missing conditions that must be considered in the evaluation of $\alpha$, see Def. 39. Each set of missing condition ($\text{mis} \in \text{missing}$) is used for evaluating the action in a possible world resulting of combining the actual evidence ($R$) and $\text{mis}$.

If the set missing conditions contain at least one realization for an intermediate variable, the action $\alpha$ is discarded. If $P(\alpha|R) = 1$, despite the existence of missing conditions, the action is considered viable. If this probability is greater than 0 but lower than 1, and at least one condition identified as missing can be enabled by an auxiliary plan, $\alpha$ is added to the list of viable actions.

If a viable action is considered unnecessary, with respect to other trials in $I_P$, the trial on which $\alpha$ was identified is discarded. Finally, $\alpha$ is considered valid if its causal effect is greater than zero; this calculation includes the set of missing conditions. In this way, the causal effect of a potential action is calculated on each possible scenario produced by the sets of missing conditions, and is selected the scenario with the highest causal effect on the target variable ($F$).
6.1. A CAUSAL BDI INFERENCE ENGINE

```plaintext
input: An ABox currBel containing beliefs, a set of essential goals \( \overline{G} \), a set of essential plans \( \overline{P} \), a set of goal instances \( \overline{I_G} \) and a set of plan instances \( \overline{I_P} \).

begin
  foreach \( G \in \overline{G} \) do
    \( \overline{Y} \leftarrow \text{evalQuery}(\text{currBel}, G.A, G.p); \)
    foreach \( Y_i \in \overline{Y} \) do // create goal instances
      \( I_C \leftarrow \text{new GoalInstance}(G, Y_i); \)
      \( \text{Repeated} \leftarrow \bot; \)
      foreach \( I'_G \in \overline{I_G} \) do
        if \( I_G \subseteq I'_G \lor I'_G \subseteq I_G \) then
          \( \text{Repeated} \leftarrow \text{Repeated} \lor \top; \)
        end
      end
      if \( \neg \text{Repeated} \) then
        \( I_{G}\text{.status} = \text{ACTIVE}; \)
        \( \overline{P} \leftarrow \text{plansSatisfying}(\overline{P}, G); \)
        \( \overline{I_P} \leftarrow \emptyset; \)
        foreach \( P_i \in \overline{P} \) do
          \( I_{P_i} \leftarrow \text{instantiate}(P_i, \text{currBel}, I_G.p); \)
          \( \text{setEvidence}(I_{P_i}); \)
          \( \overline{I_P} \leftarrow \overline{I_P} \cup I_{P_i}; \)
        end
        if \( \overline{I_P} \neq \emptyset \) then
          \( \overline{I_G} \leftarrow \overline{I_G} \cup I_G; \)
          \( \text{sort}(\overline{I_P}); \)
          \( \overline{I_P} \leftarrow \overline{I_P} \cup \overline{I_P}; \)
        else
          \( \text{setError}('\text{No feasible plan for '+'I_G}); \)
        end
      end
  end
end

Procedure GoalPlanInstantiation(CurrBel, \( \overline{G} \), \( \overline{P} \), \( \overline{I_G} \), \( \overline{I_P} \))
```
input : The set of goal instances \( \overrightarrow{I_G} \) and the set of plan instances \( \overrightarrow{I_P} \).
output: A list of viable actions \( \alpha_i = (I_p, I_i, x_i) \).

begin
valid ← \( \emptyset \);

foreach \( I_G \in \overrightarrow{I_G} \) such that \( I_G.status = ACTIVE \) do

\( I_P \leftarrow \text{first}(\overrightarrow{I_P}(I_G)) \);

viable ← \( \emptyset \);

potential ← \text{getPotentialActions}(I_P);

foreach \( \alpha \in \text{potential} \) do // look for enabled actions

apriori ← \( P(\alpha.x_i|\alpha.I_i.R) \);

missing ← \text{calcMissing}(\alpha);

if missing ≠ \( \emptyset \) then

if apriori > 0 then

| viable ← viable ∪ \( \alpha \);

| end

else if meansFor(\( I_G \)) = \( \emptyset \) then

foreach \( \text{mis} \in \text{missing} \) do

\( \alpha_i \leftarrow \text{new}(\alpha, \text{mis}) \);

if apriori ≥ 1.0 then

| viable ← viable ∪ \( \alpha_i \);

else if apriori > 0 ∧ \text{canEnable}(I_P, \text{mis}) then

| viable ← viable ∪ \( \alpha_i \);

end

end

end

foreach \( \alpha \in \text{viable} \) do

if isUnnecessary(I_P, \( \alpha.\text{trial} \)) then

\( I_P.trials \leftarrow I_P.trials \setminus \alpha.\text{trial} \);

else if \( P(F|\alpha.I_i.R, \text{mis}(\alpha), do(\alpha.x_i)) > 0.0 \) then

| valid ← valid ∪ \( \alpha \);

end

end

end

end

return valid;

Function \( \text{filterOptions}(\overrightarrow{I_G}, \overrightarrow{I_P}) \)
6.2. THE SEMANTIC LAYER

6.1.5 Action Selection

In this phase, it used the finality function, \( OFF_n(\alpha) \), for selecting the best valid detected in the previous step. The action with the highest value, above zero, is chosen. This action is semantically encoded and used to trigger the corresponding agent actuator.

The action selection function receives for input the \( \text{vactions} \) set, the \( CurrBel \) ABox, the list of goal importance factors and the list of organizational side effects. It returns an updated version of \( CurrBel \) containing the statements that indicate the execution of the selected action, or the same \( CurrBel \) if no action is selected.

The action \( \alpha_B = \langle I_p, I_i, X_i = True \rangle \) with the highest \( OFF_n(\alpha) \) is selected. If \( \alpha_B \) has missing conditions, the subgoaling mechanism is initiated. Otherwise, if \( X_i \) is directly controlled by the agent, the action is encoded in statements and added to \( currBel \). Using semantic bindings in \( Y \), for \( I_i = \langle Y, R \rangle \), are replaced in \( Ann_p(X_i = True) \) producing a set of statements \( A_{\alpha_B} = \{s_i, p_i, o_i|0 < i \leq n\} \) that is added to \( CurrBel \).

If the chosen action is \( X_i = False \), the statement \( (?agdo?act) \in Ann(X_i = True) \) is replaced by \(?agomit?act\) and the resulting statements are asserted in \( CurrBel \). If some variable \( ?arg \in Ann(V_i = v_i) \) have not been set in the SCN trial, it is replaced by an anonymous resource. In this way, if the agent decides to perform an action \( X_i = True \) with some precondition \( z_i \in pa(X_i) \) unobserved, the annotation \( Ann(X_i = True) \) is asserted with anonymous values for those semantic bindings not set by \( z_i \).

The corresponding actuator is called indicating the action type and the arguments. Arguments are represented by a conjunctive query pattern and a set of semantic bindings. If the actuator method implementing the action succeeds, \( X_i = True \) is added to \( R \); otherwise \( X_i = False \) is added. The outcome of the action will be introduced in \( CurrBel \) by the actuator (see Section 6.2.3).

If \( X_i \) is not directly controlled by the agent or \( \alpha_B = \langle I_p, I_i, X_i = False \rangle \), \( x_i \) is added to \( R \) and neither \( currBel \) nor \( Y \) are updated.

In the next agent cycle, statements \( (?self?do?act) \) and \(?agomit?act\) are removed after belief revision. This allows triggering rules after the execution or omission of actions by the agent.

6.2 The Semantic Layer

A semantic layer is incorporated around BDI reasoning through the introduction of the Jena Toolkit and custom sensors and actuators interfaces. The Jena Toolkit is used for storing and querying models, as well as performing semantic inference. The sensor interface is used to translate external stimuli into RDF statements, whereas the actuator interface is used by actuator implementations for triggering action execution and store information gathered from environment.
CHAPTER 6. THE CAUSAL AGENT ARCHITECTURE

6.2.1 Semantic inference

The OntModel class provided by the Jena Toolkit is used for representing beliefs sets through ABoxes. As mentioned in section 6.1.2, definitions loaded from the Organizational Ontology are used for performing semantic inference. The inference level configured in these models is a custom set of rules that comprehends RDFS and some OWL Lite rules. Entailment is performed automatically on each modification of the ABox. Evaluation of annotations and goal conditions over the beliefs ABox uses Jena facilities for querying the model.

6.2.2 The Ontology Client Component

The Ontology Client Component (OCC) is used by the agent for connecting with an ontological repository from which it can download the Organizational Ontology, as well as the agent class, the list of plans controlled by it, and rules that uses for updating its beliefs set. This component is used too for determining query containment: it inquires the ontological repository which contains precalculated query subsumption relations.

The OCC uses the agent class and actual roles for retrieving the set of semantic definitions, essential goals ($G$), essential plans ($P$), and immediate ($R_I$) and exogenous ($R_E$) rules associated to plans and to the agent.

6.2.3 Sensors and Actuators

Sensor and actuator interfaces must translate RDF information into Java objects, and vice versa. Both interfaces are declared as abstract classes and provide a constructor on which it is made the link between the agent object and the sensor or actuator (generically known as components). Actual implementations of sensors and actuators extends these interfaces. Besides, these are declared semantically as individuals of the corresponding DL class. In the case of actuators, their definition includes the set of actions they implement. In both cases, a pointer to the Java class is given.

The Sensor Interface

The Sensor interface additionally provides the abstract boolean function sense() which receives for parameter the set of perceptions and returns true only if the sensor added information on it. This function is overridden in actual implementations of sensors. Sensor implementations must use those schemas the agent recognizes as valid perceptions, defined along with the agent class.
6.2. THE SEMANTIC LAYER

The Actuator Interface

On the other hand, the Actuator interface establishes a convention for declaring methods that implement actions specified as subclasses of the class \textit{Action}. This convention indicates that these methods receive the actual set of beliefs (\textit{currBel}), the set of beliefs in the next agent cycle (\textit{nextBel}), a set of bindings for semantic variables and a query pattern indicating the meaning of the values contained in the bindings. The method is named with the local name of the action class uri.

If the query pattern is known by the method implementation, the set of bindings is used directly. Otherwise, the implementation can check if the given query pattern contains all the arguments it needs using query containment. If so, it extracts the corresponding arguments using the respective CQ mapping.

Results obtained from this routine are added to \textit{nextBel} indicating that the agent perceived this information from the environment. The method returns true if the action was executed successfully; otherwise returns false.

6.2.4 The Agent class

The actual implementation of a Causal Agent is defined by a subclass of one or more classes representing human or software roles in the organization. This new class or concept must have additional constraints that produce a single interpretation of the agent type. These constraints can be given at design phase, being included in the definition, or can be given during instantiation of the agent, as initial information.

In this way, the causal agent will be configured with the union of the constraints included in the classes that implements (is subclass of). These constraints expresses the elements described in Definition 80.

6.2.5 Communication Protocols

Any process or plan on which intervene more than one agent has a communication protocol associated. Each message is represented by the tuple \textit{msg} = \langle\textit{sender}, \textit{receiver}, \textit{when}, \textit{content}\rangle, where \textit{sender} and \textit{receiver} are the process roles (i.e. variable names used in the SCN for denoting agents), \textit{when} is a set of probabilistic realizations that indicates the moment on which the message must be sent or will be received, and \textit{content} is a set of probabilistic realizations whose triggered annotations are used for encoding the message. All messages use the performative INFORM.

Outgoing messages are triggered when a trial \(I_i = \langle Y, R\rangle\) of a SCN case met the condition \textit{when}. This revision is performed: 1) during plan-goal revision, after evidence feeding of trials on active plan instances, and 2) after plan execution. The content of the message is built using the conjunctive query \(\text{Ann}(V_1 = v_1) \cup ... \cup \text{Ann}(V_n = v_n)\), and the semantic bindings \(Y\) of the SCN trial \(I_i = \langle Y, R\rangle\). The receiver agent is obtained from \(Y\) using
the variable name `receiver`. If the receiver agent is not known, the message is queued and send later.

### 6.3 Causal Agent functionality

Integration of semantic, Bayesian and causal representations allows learn through experience and adjust the behavior of the agent. Additionally we show how agent’s potential properties, expressed on agent class definitions, are used by actual agents for consciously instantiating new agents.

#### 6.3.1 Subgoaling

In this section we describe a non-recursive subgoaling mechanism for enabling stalled plans. We describe its implementation on the procedures previously described.

**In options filtering**

The subgoaling mechanism is invoked if the selected action $\alpha_B$ has missing conditions. We assume that at least one condition identified as missing can be enabled by an auxiliary plan and that the goal instance $I_G$ for which the plan containing $\alpha_B$ is intended, is not a subgoal. This avoids infinite cycles.

For each plan capable of enabling a missing condition $Z_i = z_i$ on any trial $I_i$ is generated a subgoal. These subgoals are associated to the respective $I_i$ and there are filtered in the plan instance to avoid duplicate goals. The $I_P$’s list of subgoals is used for adopting subgoals every time the plan got stalled.

The status of the goal $I_G$ is set INACTIVE and it is set the property $I_{G2}.\text{meansFor} = I_G$. Then it is asserted $\text{satisfies}(P_2, I_{G2})$, which will allow the instantiation of $P_2$ during plan instantiation in the options generation phase.

**In plan and goal revision**

Before removing a satisfied or unsatisfied goal instance $I_G$ from $\overline{T_G}$, it is checked if the goal is means for another goal $G_1$. If so, the status of the goal $G_1$ is set ACTIVE again. Evidence feeding and goal status revision is repeated until no goal is reactivated.

#### 6.3.2 Learning

In the actual implementation of the agent, learning is done off-line with batch algorithms, i.e. using closed SCN cases reported by agents for refining the original model given by
6.4. AGENT TYPES

the expert. See section 4.4. The updated Bayesian network is replaced in the SCN representation of processes and plans and is used by new agents.

6.3.3 Agent instantiation

Instantiation of agents is done specifying three of the main causes: the formal (the agent class), the efficient (the creator agent ID), and the material (property values). The OCC connects to the ontological repository for downloading the schemas describing the agent class and those schemas relevant for the agent. In the same way, Java classes for the agent template and its components are loaded locally or from a given URI. Finally, agent properties are initialized with the definitions contained in the OO and with the parameters indicated on its instantiation, which include the efficient cause. The final cause of the agent instantiation is not explicitly given to the agent, but it is assumed that it is contained in one of its essential goals. The agent instance is represented by an URI, which can be generated automatically or be given as an additional parameter.

The instantiation act can be performed by any agent, human or software. The first agent in the system is necessarily created directly by the administrator, which is a human user, meanwhile the rest of the agents are created by this primary agent on behalf of the human user. This primary agent is on charge of an organizational process (a PMA agent) and its implementation, as described in section 5.3.2, is detailed in section 6.4.3. Besides, the primary agent receives an initial world state as additional parameter on its initialization.

6.4 Agent Types

Using the Causal Agent architecture, we propose three types of agents with different purposes and levels of complexity. The Specialized Agent provides a type of service by performing a specific task repetitively. The User Agent is designed to enable the participation of a human user in organizational processes. Finally, the Manager Agent monitors and facilitates an organizational process thanks to the global view it has of it.

6.4.1 The Specialized Agent

A Specialized Agent is a Causal Agent entirely advocated to perform indefinitely certain task in an organization. The purpose of the agent is to perform that task, and is expressed through an essential goal.

It controls at least one essential plan for each essential goal it has. Process views indicate how the agent participates on the different organizational processes by performing its designated task.
Examples of specialized agents are the Process Monitor (PMO), which pursues the same goal than the organizational process that it monitors, and other utilitarian agents that provide services like users notification, database access, etc.

6.4.2 The User Agent

The User Agent is a Causal Agent that acts in behalf of a human user (identified univocally by a Person ID) that controls human-computer interfaces (HCI) and keeps track of all the organizational roles its represented user plays in the organization.

Main essential goals of a User Agent are: 1) to inform the human user when its participation is required, and 2) to receive human input. Process views describe the participation of the human user in the organizational process. HC interfaces controlled by the User agent employed for both subtasks are: email servers and input webpages.

Additionally, the User Agent keeps track of the preferences of the human user and his/her organizational commitment on each organizational process.

6.4.3 The Process Manager Agent

The Process Manager Agent (PMA) is a Causal Agent that monitors and facilitates an organizational process on behalf of a human user. In order to do so, it owns a global view of the organizational process that manages and which uses to keep track of every execution of the process.

The efficient cause of the PMA is a human user that initiates and is responsible for the execution of a process. This human user, known as process admin, provides additional information to the PMA and enables additional conditions required for the execution of the process through Human-Computer interfaces controlled by the agent. HC interfaces are used by the PMA to inform the process admin of the status of the process and its performance.

Meanwhile other types of Causal Agents control plans for requesting the participation of other agents, the PMA controls plans for instantiating those agents. In this way, the PMA guarantees that the process counts with enough agents. Knowledge, information and resources required for the process execution are given by the process admin during its instantiation (in the initial world state).

The process admin can adjust organizational goals and metrics associated to the organizational process. These adjustments include: 1) changing the importance or the conditional expressions of an organizational goal/metric, and 2) set up or give up an organizational goal/metric. The PMA is on charge of propagating these adjustments between the agents participating in the process. The identification of side effects in plans controlled by other agents is centralized in the PMA, which updates the agent plans repository.
6.5 Summary

I introduced an intelligent agent architecture called *Causal Agent*. This architecture implements a goal-driven BDI inference engine supported by Semantic Causal Networks and Causal Rules, providing subgoaling, learning and instantiation of agents on demand. Through this architecture I implement three types of agents with distinct purposes and complexity.

In the *causal BDI inference engine*, beliefs and perceptions are represented by ABoxes, meanwhile desires are represented by goals. Plans (SCNs) controlled by the agent are initiated for satisfying such goals. Whereas open SCN cases allow identifying valid options and calculating their contribution to the finality of the agent, closed SCN cases are used to update the probabilistic distribution of plans.

*Belief revision* is made through causal rules applied over beliefs and perceptions, treated as nonmonotonic ABoxes. During this phase, basic semantic inference exploits domain’s common sense formalized in ontological definitions. Sensors format perceptions as RDF triplets, meanwhile actuators access the beliefs ABox for retrieving action parameters and update the perceptions ABox with action’s output.

The Causal Agent is endowed with an *Ontology Client Component (OCC)* which retrieves goals, plans, rules and software components declared in the agent’s ontological definition. In this way, multiple kind of agents are implemented through a single Java agent class configured on setup according to a given ontological definition. The agent class definition also allows to determine the agent capability for playing certain role in a plan or process. This information facilitates either inviting an existing agent to participate in the process or instantiating a new agent of a suitable type with this purpose.

Agents are designed for performing certain tasks defined by their essential goals. *Specialized agents* are dedicated to a small number of actions providing simple services to other agents in the system. *User agents* acts on behalf of a human user according to the roles they play in current organizational processes. *Manager agents* monitor and enable complex processes through the instantiation of agents and the dissemination of information between them.
Chapter 7

The Causal Multi-Agent System

In this chapter is introduced a framework for automating processes in an intelligent organization through a multi-agent system. I call this framework *Causal Multi-Agent System*, CMAS for short. In the first place, it is presented the specification of the intelligent organization and the computational infrastructure that supports the participat Causal Agents. Then I present some basic operations of the CMAS and conclude proposing a methodology for specifying and implementing the intelligent organization and the causal agents employed on it.

7.1 The CMAS Architecture

The CMAS architecture supports a group of Causal Agents carrying out organizational processes defined in an intelligent organization. Figure 7.1 illustrates this architecture.

In a CMAS is made explicit the participation of human administrators and members of the organization (users). A Knowledge Management System (KMS) is used for managing the Organizational Ontology (OO) and the Agents Repository (AR). The former contains the specification of the intelligent organization, which includes information and resources used on it, organizational roles, agent types, etc. The latter contains the specification of actual agent implementations defined in the OO. A Multi-Agent System platform is used for deploying instances of causal agents that will implement active organizational processes. Finally, it is used a Knowledge-Information Interpreter for accessing external resources and information repositories using the schemas defined in the OO.

7.1.1 The Organizational Ontology

The Organizational Ontology (OO) is stored in a custom database providing persistence for schemas, information and results of the system operation. A RDF store or XML
database can be used for this purpose. The Intelligent Organization definition (see Def. 64) and all the elements used on it are defined in the OO. Organizational, agent and human, roles are stored here, but the complete definition of human users is usually stored in an organizational database mapped through the K-I interpreter, meanwhile that actual agent classes and their implementations are given in the Agent Repository. In the same way, external resources and information/knowledge repositories are declared as entities used, consumed or known by the organization.

Along with the OO, there are stored the organizational processes in a separate named graph called Process Repository. This repository is codified using RDF and follows the convention for representing conjunctive queries, annotations, variables, domains, graphs and probabilistic distributions, defined in an OWL ontology with this purpose.

7.1.2 The Knowledge Management System

A Knowledge Management System (KMS) is used by the system administrator for managing the Organizational Ontology, the Agent Repository and mappings used by the K-I Interpreter. A tool like Protege\(^1\) or Top Braid Composer\(^2\) can be used with this purpose. Ontological repositories are codified through the RDF and OWL languages. The ontological repositories managed through the KMS are illustrated in Figure 7.2.

---

\(^1\) The Protege Ontology Editor and Knowledge Acquisition System. [http://protege.stanford.edu/](http://protege.stanford.edu/)

7.1.3 The Agent repository

The Agent Repository contains the specification of actual agent classes defined in the OO as well as code libraries. Agent classes extend organizational agent types with additional constraints on properties, sensors, actuators, rules and plans. Code libraries contain Java platform-independent code for the agent template, sensors and actuators.

7.1.4 The Knowledge-Information Interpreter

The Knowledge-Information Interpreter is responsible for mapping information into knowledge and vice-versa. Information from external resources or repositories is translated into knowledge (meaningful information) once that is represented using the schemas represented in the OO. In order to do so it uses a D2R³ server and custom Java APIs.

The D2R server allows extracting information from relational databases and present it using OO schemas. Modifications on relational databases mapped through the D2R Server are made through parameterized SQL commands. These commands are executed through a Java API that takes the parameters from the action invocation, execute the command on the database and returns the results using OO schemas.

Additional custom Java APIs are used for encapsulating the access to external resources. These APIs are accessible to Causal Agents and implement methods defined as action classes in the OO. Method’s input and output is given in terms of causal rules and uses ABoxes as vehicle for information exchange.

D2R mappings and SQL commands are managed by the System Administrator through

the KMS. Java APIs are stored in the Agent Repository as sensor and actuator implementations.

7.1.5 Organizational Assets

Among the organizational assets used by agents in a CMAS we can have relational databases, email servers, web servers, web sites, among others. The different types of external resources used in the organizational processes are modeled in the OO as subclasses of classes Material and Form (see section 3.1). Properties of external resources are encoded in the OO as well. Properties include actions and constraints; for instance, an email server can send at most 300 emails simultaneously. Actual resources owned or controlled by the organization are declared in the OO as individuals of the respective classes.

7.1.6 System and Process Administrators

The System Administrator is the human responsible for initiating the MAS Platform, managing the OO and the Agent Repository, updating mappings in the K-I Interpreter, initiating processes and instantiating agents common to all processes. In the agent platform, the system administrator is represented by a User agent.

A Process Administrator is a human user responsible for the execution of an organizational process. Autonomous management of the process is made by a PMA whereas a direct intervention in the process is done through a User agent.

In order to manage the different organizational processes, as well as common resources and agents, it is defined a system process that is managed by the system administrator as its process manager. Through the system process, the system administrator initiates new organizational processes, modeled in the OO, initiating new PMAs and setting process administrators as their efficient causes.

7.1.7 System users

Organization members participate in the organizational processes implemented in a CMAS through User agents. When the number of members in the organization is large, the user identifier and organizational roles played by a member are stored in organizational databases mapped through a D2R mapping. The PMA is responsible for instantiating User agents whenever a human user is asked to participate in a process. In consequence, the PMA controls the repository containing user’s information. Preferences of human users modeled and used by the User agent are stored in the OO by the PMA.
7.1.8 The MAS platform

During the CMAS start up, a FIPA\(^4\)-compliant Multi-Agent System platform is initiated and registered in the Organizational Ontology. The MAS platform is used for deploying agent instances on the different layers explained below. JADE\(^5\) or FIPA-OS\(^6\) platforms can be used for this purpose.

Agent Layers

Causal agents performing in a CMAS are arranged in layers according to its type and the role they play in a process. These layers are illustrated in Figure 7.3.

User agents are located in the *Client layer*. Their instantiation, requested by some agent in any layer, is made by the System Administrator in the system process. In this layer we can find a single User agent representing a human user. If agents in different processes simultaneously require the participation of certain human user they

---

\(^4\) FIPA: The Foundation for Intelligent Physical Agents, http://www.fipa.org\n
must address the current instance of the User agent representing this human user or request the instantiation of the respective User agent.

The Process layer contains PMAs and specialized agents arranged around process groups. Each process group counts with a set of dedicated agents that include the responsible PMA and several specialized agents (including PMOs) required for the process execution. Specialized agents in a process group are instantiated by the respective PMA.

The Utility layer contains specialized agents providing common services to all active process. Specialized agents in this layer are instantiated by the System Administrator through the system process. A service like the Directory Facilitator, which indicates which agents are currently active in the MAS platform, is provided by an specialized agent located in this layer.

In summary, User and Specialized agents instantiated by the system administrator through the system process are deployed in the Client and Utility layers, respectively. On this way, common services are warranted using the same mechanism of agent instantiation by PMAs on behalf of human users.

### 7.2 The CMAS Methodology

I propose a methodology denominated Causal Multi-Agent System (CMAS) for modeling Causal Intelligent Organizations and implementing organizational processes through a Multi-Agent System. The methodology starts with the formalization of processes originally performed by organization members and concludes with the specification of organizational processes carried out by software agents and supervised by humans.

My methodology consists of the following steps arranged in phases:

I. The Original Process

1. Define organizational goals ($\overline{G}$) pursued by the organization and their importance ($\overline{T}$).
2. Specify one or more organizational processes ($\overline{P}$) for achieving each goal $G \in \overline{G}$.

II. The Automated Process.

3. Introduce software agents specialized on functions susceptible of automation through intensive application of knowledge.
4. Incorporate human supervision when needed.
5. Define metrics that should be observed by the organization ($\overline{M}$) and incorporate their observation in as many $P \in \overline{P}$ as possible.

III. The Agentified process.
6. Designate PMA and PMO agents for each process $P \in \overline{P}$.

7. Generate process views per each process role defined in every process $P \in \overline{P}$.

8. Identify the strategies that each participant might follow in every process $P \in \overline{P}$.

9. Generate the communication protocol of each process $P \in \overline{P}$.

IV. The Ontological Model

10. Build the *Organizational Ontology* (OO) defining the organization $(O)$ and extracting from every $P \in \overline{P}$ the definition of participants $(A)$, information/knowledge $(K)$, and resources $(R)$ and associate each of them to the class representing the IO. Declare organizational assets involved in these processes as well.

11. Validate the specification of goals and plans.

V. Agent implementation.

12. Specify concrete agent classes implementing one or more organizational roles.

13. Implement sensors and actuators, customize controlled plans and incorporate causal rules.

14. Design auxiliary protocols that support organizational processes $P \in \overline{P}$.

### 7.2.1 Defining Organizational Goals

Define as many goals as the organization pursues. Use a template like the shown in Figure 7.4. Attributes marked with * are mandatory. Conditions are expressed using $C(x)$ and $P(x,y)$ for representing entities declarations and properties, respectively; $C$ is an essential form, $P$ is a property, and $x$ and $y$ can be variables (preceded by a question mark, ?) or constants (typed literals). Condition’s distinguished variables are underlined in the expression. The $\Omega$ condition contains only relevant conditions that must hold for considering the goal achieved.

<table>
<thead>
<tr>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong>:</td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Whenever (A)</strong>:</td>
</tr>
<tr>
<td><strong>Achieve (\Omega)</strong>:</td>
</tr>
<tr>
<td>**Importance **:</td>
</tr>
</tbody>
</table>

Figure 7.4: Goal template.
7.2.2 Specifying Organizational Processes

In this phase the current organizational processes are formalized as they are carried out actually, even if they only involve the participation of humans in manual tasks. In the first place we give a textual description of the process and then it is introduced the use of the SCN graphical representation as modeling tool (see Figure 4.1).

Use a template like the shown in Figure 7.5 for the textual description of the process. In this template, $C$ and $F$ represent the contextual and the final condition; in consequence, $A \sqsubseteq C$ and $\Omega \sqsubseteq F$, where $A$ and $\Omega$ describe the goal $G$ satisfied by $P$.

<table>
<thead>
<tr>
<th>Name*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whenever (\textit{Q})*</td>
<td>Condition</td>
</tr>
<tr>
<td>Achieve (\textit{F})*</td>
<td>Condition</td>
</tr>
</tbody>
</table>

![Figure 7.5: Process template.](image)

Next proceed with the formal specification of each organizational process following these steps:

**OP-1. Initial and Final conditions.** Initiate the SCN with three nodes: a covariate $Z_C$ representing $C$, a covariate $Z_F$ representing $F$ and a control variable $X_\alpha$ representing some action or set of actions performed by some human and required for achieving $F$. Connect the nodes: $Z_C \rightarrow X_\alpha \rightarrow Z_F$. Note that each covariate or control variable used in the model must be labeled for its easy identification and reference.

**OP-2. Actions and intermediate conditions.** Replace $X_\alpha$ by causal chains of control variables $X_i$ and covariates $Z_i$, representing actions performed by human users and intermediate conditions, respectively. The different types of workflows that can be modeled using causal chains are described in section 4.3.2.

Intermediate conditions are represented by rounded boxes filled with an expression (a conjunctive query), that contains typed entities declarations and properties of these entities.

On the resulting DAG it must exist at least one causal chain from $Z_C$ to $Z_F$. Each causal chain is a possible strategy that an agent must choose when reaches a covariate with multiple outgoing arrows.

**OP-3. Action causes.** Check that the main causes involved on each action, represented by a control variable $X_i$, are present in the model. If some formal, material or efficient cause was not included in the first place introduce a new covariate $Z_i$ representing such cause and add the arrow $Z_i \rightarrow X_i$. The annotation of $X_i$ would be complemented for representing the participation of these causes.
7.2. THE CMAS METHODOLOGY

OP-4. Entities declaration. Verify that every variable representing an entity participating in the process has been previously declared. Entities declaration can be done in the same node $V_i$ where the variable $v$ is referenced or in an additional covariate $Z_i$ marked as cause of $V_i$, i.e. $Z_i \rightarrow V_i$. In the corresponding node annotation add $T(v)$, where $T$ is the class or type of the entity.

OP-5. Merge excluding conditions. If two covariates $Z_1$ and $Z_2$ have the same parent(s) and are mutually excluding, both conditions must be merged. To do so, in $Z_1$ include the expression(s) used in $Z_2$, separating each expression with a dotted line and using a label for differentiate them. Edges originally outgoing from $Z_2$ are changed for outgoing from $Z_1$. Finally, remove $Z_2$.

OP-6. Arc labels. Arcs outgoing from covariates with multiple conditions can be labeled when the process execution depends on the value of the covariate. Multiple outgoing arcs on which the decision of following one path or another depends on the agent’s choice are not labeled. Arcs are marked with the respective label used to identify the covariate expression.

OP-7. Entity types classification. Identify those variables representing resources, forms and organizational roles in the SCN. According to the type $T$ used in their entity declaration, $T(e)$, classify them according to the taxonomy given in section 3.1. To do so, elaborate a relation

$$types = \{T_i \subseteq C_i | C_i \in \{Person, Resource, Knowledge, Information\}\}$$

OP-8. Possession, control and awareness. Once that efficient causes $(Ag_i)$ and their actions $(a_i)$ have been identified, it must be explicitly indicated the relation between the agent and causes of its actions. Analyze each control variable representing the execution of $a_i$ by $Ag_i$, searching for action parameters: $?ct$ and $?val_i$. The covariate $Z_T$ containing the declaration of $Ag_i$, $T(Ag_i)$, is complemented with a role assertion $p(Ag_i, ?val_i)$ such that $p = \{controls, has\}$ if $val_i$ is a resource and $p = knows$ if $val_i$ is a piece of information or knowledge.

OP-9. Quantifiers. Now we introduce quantifiers in control variables to indicate how many executions of the action must be performed in the process execution. The types of quantifiers considered are existential (ONE) and universal (ALL), and are introduced before each action parameter $val_i$ indicating that the action must be performed with only one $val_i$ or with all $val_i$. In the absence of a quantifier it is assumed ONE.

OP-10. Adjust on semantic variables. It can be used valued semantic descriptors (see Definition 22) for adjusting the model to the values contained in semantic variables. To do so, in a given covariate $Z_i$ introduce the name of the semantic variable $?v$ on which you desire to adjust the model as label. This means that any value that $?v$ holds in some instance of the model will be added to the domain of $Z_i$ in the learning phase.
In the resulting process model set $C = \{Z_C = True\}$ and $F = \{Z_F = True\}$. Then fill the probabilistic distribution deterministically, i.e. $P(v_i|pa_i) = \{0, 1\}$, depending on the arc labels or assuming the label $True$ when none exists on the arc. If we denote the set of values contained in the label connecting $V_i$ to $V_j$ with $lb(V_i \rightarrow V_j)$, we can fill $P(v_i|pa_i)$ automatically for boolean variables $V_i$ by doing $P(V_i = True|PA_i = pa_i) = 1$ and $P(V_i = False|PA_i \neq pa_i) = 1$ where $pa_i \in lb(PA_i \rightarrow V_i)$, and filling the rest of the $V_i$'s CPT with zero. Variables with non-boolean domains can be filled automatically with a constant distribution, where $P(V_i = v_i|PA_i = pa_i) = \frac{|Dom(V_i)|}{|Dom(V_i)|}$. Custom configurations of the probabilistic distribution can be expressed by a human expert using functional causal models; see (2.4).

This phase concludes with a SCN for each process on which it can be identified participant agents as well as resources, knowledge and information required in the process. The achievement of the organizational goal motivating the process is decomposed in intermediate conditions produced by the execution of actions performed by the participants. These steps must be followed in subsequent modifications of the SCN.

7.2.3 Introducing Specialized Agents

Identify specific functions that software agents can perform autonomously with or without intensive application of knowledge and define agent roles for each function. For instance, monitoring changes in a database or in a web site is a function that can be delegated to a software agent.

Software agents, identified by a type $T \subseteq SoftwareAg$, are introduced on the corresponding process specification using the following type of nodes: $Z_A$, $Z_R$, $Z_K$ and $X_A$. $Z_A$ contains the agent type declaration $T(Ag)$, $X_A$ describes an action performed by $A$ ($Z_A \rightarrow X_A$), $Z_R$ contains the resources that $Ag$ must own or control for performing the action ($Z_R \rightarrow X_A$), respectively $Z_K$ contains information or knowledge known by $Ag$ ($Z_K \rightarrow X_A$).

The effect of $X_A$ is an existing condition $Z_O$ in the process ($X_A \rightarrow Z_O$). For indicating that the execution of $X_A$ depends on the occurrence of an event represented by $Z_I$, is added the edge $Z_I \rightarrow X_A$; dependency is adjusted setting $P(X_A = True|Z_I = z_I, pa_{X_A}) = 1$.

Each action performed by a software agent is represented following the previous convention. This is illustrated in Figure 7.6. Similar functions performed on different processes are standardized through the definition of agent roles, their actions and their respective causes.

Actions like $X_A$ controlled by autonomous agents of type $T$ are later used for specifying agent's essential goals. The conjunction of causes of $X_A$ will constitute the initial condition for the essential goal, i.e. $A = Ann(Z_A) \cup Ann(Z_R) \cup Ann(Z_K) \cup Ann(Z_I)$, whereas its final condition will be given by the intersection of all possible outcomes of $X_A$, i.e. $\Omega = \bigcap_i Ann(Z_{O_i})$. If it doesn't exist such intersection, i.e. $\Omega = \emptyset$, it must be
introduced a common condition in every output.

7.2.4 Introducing Human Supervision

Identify critical tasks performed by software agents that must be supervised by a human user. Then designate the organizational role responsible for this supervision. Users playing this role are considered experts in the task and will be on charge of approving, rejecting, or directly correcting the action made by the software agent. This supervision cycle allows to train the software agent through reinforcement.

Originally, the process describes an action $X_A$ performed by a software agent and its effect is represented by $Z_O$. The expert user on charge of supervising the execution of this action is represented by a new covariate $Z_E$ annotated with $C(?e)$ where $C \subseteq Person$. A supervision action $X_S$ performed by the expert is introduced having as parents to $Z_O$ and $Z_E$. The effect of $X_S$ is a new covariate $Z_S$ containing four expressions: 1) the approval of the expert, 2) the rejection of the action and request for undo it, 3) the indication of a faulted action corrected directly by the expert, and 4) the request being neglected by the supervisor. Finally, edges originally outgoing from $Z_O$ now goes out from $Z_S$. This process is illustrated in Figure 7.7.

An additional causal relation from $Z_O$ to $Z_S$ indicate that the agent can choose not to request human supervision. In this case, it is introduced a new expression in $Z_S$ indicating that the action was validated by the agent itself. In order to make this decision, the agent must consider the organizational side effect of minimizing unnecessary human intervention (see Chapter 8).
7.2.5 Incorporating Organizational Metrics

Define organizational metrics for the possible outcomes of organizational processes and for side effects produced during their execution. Then incorporate the resulting additional conditions in the respective process models.

Process organizational metrics are used by the organization for assigning a different preference on each one of the possible outcomes of the process. The final condition $F$ of the process $P$ is complemented with an additional condition $O_i$ for representing the $i$-th possible outcome of $P$, obtaining $i$ new final conditions $F_i = F \cap O_i$.

General organizational metrics are represented by conditions that are rewarded or penalized on the different processes on which they can be identified. Use template shown in Figure 7.8 for capturing process organizational metrics, and template shown in Figure 7.9 for capturing general organizational metrics.

For a group of process organizational metrics, modify the content of $Z_F$ setting as many $F_i$ as possible outcomes were defined for the process. $P$’s final condition is now $F = \{Z_F = i|0 < i \leq n\}$ and $Ann(Z_F = i) = F_i$. Labels $i$ are used to distinguish $Z_F$’s realizations pondered distinctively by the organization.

Given a general organizational metric $M = (C, p)$, revise every process $P \in \overline{P}$ looking for covariates $Z_i$ containing the condition $C$, i.e. $C \subseteq Ann(Z_i = z_i)$. If the condition $C$ is not explicitly represented in $P$ but it is the consequence of a series of events occurred in the process, it can be introduced a new covariate $Z_M$ representing its occurrence, i.e. $Ann(Z_M = True) = C$. Add edges outgoing from every covariate $Z_i$ representing events causing $C$, and incoming into $Z_M$. 

---

**Figure 7.7:** Nodes introduced by human supervision.
7.2. THE CMAS METHODOLOGY

7.2.6 Introduce Process Managers and Monitors

For each organizational process designate an agent class $PMA_P$ representing the Process Manager Agent and an instance $H$ of the class $Person$ identifying the human user responsible for the process. In the same way, designate the human user that will be responsible for the process, i.e. the process admin.

If a single agent of class $A_i$ participates in an organizational process $P$, $A_i$ is necessarily the Process Monitor Agent for $P$, i.e. $PMO_P$. If there is more than one participant, the agent class $A_i$ of one of the participants can be designated as $PMO_P$. Otherwise, a new agent class is defined in the OO for this purpose. Finally, assert $monitors = P$ in the definition of $PMO_P$, in the OO.

For each organizational process $P \in \mathcal{P}$ use the template shown in Figure 7.10 for registering the information of PMAs and PMOs.

Now introduce the PMA and PMO agents in the process specification. To do so follow these steps:

PM-1. Agent nodes. Add two boolean covariates $Z_{PMA}$ and $Z_{PMO}$ for representing the PMA and the PMO agents, respectively. Add the annotations: $Ann(Z_{PMA} = True) = \{?PMA isA PMA_P\}$, $Ann(Z_{PMO} = True) = \{?PMA isA PMO_P\}$, $Ann(Z_{PMA} = False) = \bot$ and $Ann(Z_{PMO} = False) = \bot$. Finally add a relation
Z_{PMA} \rightarrow Z_{PMO}.

PM-2. Enabled actions. Define which actions are enabled by the PMA and which are enabled by the PMO. The former are represented by $\overline{X}_{PMA}$ and the latter by $\overline{X}_{PMO}$. Add a casual relation $Z_{PMA} \rightarrow X_i$ for each $X_i \in \overline{X}_{PMA}$. Do the same for the PMO.

PM-3. Adjust the CPTs. Constrain the CPT of actions $X_i$ to the presence of the PMA/PMO. Use the following formulas:

$$P(x_i | pa(X_i), Z_{PMA} = True) = P(x_i | pa(X_i))$$
$$P(X_i = True | pa(X_i), Z_{PMA} = False) = 0$$
$$P(X_i = False | pa(X_i), Z_{PMA} = False) = 1$$

PM-4. Adjust the contextual condition. Add $Z_{PMA} = True$ to $C$.

We conclude with an augmented process specification.

### 7.2.7 Generating Process Views

In this phase, the augmented process specification is used for generating the process view that every participant agent will control. The PMA and the PMO have particular process views compatible between them. The process view of the rest of the participants is obtained by extracting a subgraph from the augmented process specification.

For each process participants, identified by the semantic variable $?role$, a copy of the original process diagram is modified to generate the respective process(es) view(s) depending on the actions the participant controls directly or indirectly. In the copy of the augmented process diagram controlled by the PMA and the PMO, control variables not enabled by them are transformed into covariates. In the first place, there are calculated all the possible partial orders $O_i$ of control variables in the model. Follow the next steps for each process role with a new copy of the augmented process specification for each $O_i$:

---

**Figure 7.10:** Template for process PMA and PMO.
PV-1. **Self awareness.** Replace variable \(?role\) by variable \(?self\) on every annotation where \(?role\) appears. Mark every variable \(V_i\) where the replacement was made.

PV-2. **Action causes and effects.** Mark every direct cause or effect of control variables \(X_i\) where the efficient cause of the action is \(?self\).

PV-3. **Initial condition.** The initial condition \(C\) for the process view is relocated to some of the marked nodes. Locate the first action performed by the agent \(Ag\) in the process. The first action is the action \(X_1\) performed by \(Ag\) such that \(X_1 \leq X_j\) for all the actions \(X_j\) performed by \(Ag\) in all the partial orders of \(X\).

If it is not possible to identify a unique first action this means that there are multiple actions the agent can execute to initiate its participation in the process. In this case, generate as many copies of the process view as first actions were identified and designate a different \(X_1\) on each one of them.

On each process view derived from the identification of \(X_1\), identify those covariates ancestors of \(X_1\) having at least one parent in the process SCN, denoted \(A_1\). Then identify the combinations of realizations of \(A_1\) such that \(P(X_1 = True|a_1) > 0\). Each combination \(a_1\) produces a contextual condition \(C\). If multiple contextual conditions appear, the process view is cloned again setting \(C = a_1\) for each \(a_1\) in each copy.

PV-4. **Ancestors removal.** Remove all ancestors of marked nodes (excluding marked nodes), as well as arcs incoming or outgoing from them. Removal also includes retracting quantifiers and annotations referencing such variables.

PV-5. **Organizational goal and metrics.** Mark every \(Z_F\) and \(Z_M\), representing organizational goals and metrics, respectively, and set them as unobservable variables, i.e. \((Z_F \cup Z_M) \subseteq U\), unless they are direct effects of a control variable.

PV-6. **Unmarked nodes absorption.** Finally absorb not marked nodes. Sort unmarked variables according to some partial order given by \(G\) and progressively absorb each node.

For each unmarked variable \(V_i\), add an arc from each \(V_j\) to each \(V_k\) such that \(G\) contains an arc \(V_j \rightarrow V_i\) and an arc \(V_i \rightarrow V_k\). Then remove every arc incoming/outgoing to/from \(V_i\). References on quantifiers and annotations to \(V_i\) are removed as well.

Probabilistically, the resulting SCN constitute a subgraph of the original SCN. Process view variables constitute a subset of the variables contained in the process SCN. The resulting graph contains causal relations that represent direct or indirect causal relations on the original graph. In consequence, the probabilistic distribution of the new causal network can be calculated using the original CPTs and marginalizing on pruned variables.

We conclude this step with one or more process views controlled by every agent participating in an organizational process. Agents will adopt these process views as plans to
guide their decision making based on their observations, the actions they can perform and the effects these have on organizational goals and metrics.

### 7.2.8 Identifying participant strategies

Next identify all the possible strategies that can be followed on each organizational process and each process view derived from them. Represent strategies through one or more OBDDs (see Def. 31). For each organizational process and for those process views containing at least one controllable action $X_i$ on it, follow the next procedure.

**S-1. Calculate strategies.** Sort actions $X_i \in X$ according to their precedence in the causal diagram of the SCN $M$. If more than one partial order of $X$ is found, use each one of them, $O_i$ for calculating the strategies. The set of strategies associated to $O_i$ are denoted $\overline{S}(M, O_i)$.

**S-2. Remove equivalences.** If two strategies $S_i \in \overline{S}(M, O_i)$ and $S_j \in \overline{S}(M, O_j)$ contain the exact same set of realizations $x_i$, the strategies are considered *equivalent*. Remove $S_i$ or $S_j$.

**S-3. Generate OBDDs.** For each $\overline{S}(M, O_i)$, generate the corresponding OBDD.

Calculation of strategies requires considerable computational time, hence it is recommended to store them for future uses. In the other hand, OBDDs and OCDDs are calculated in very few time, hence they can be calculated on demand.

### 7.2.9 Generating the Communication Protocol

Process views, and the corresponding OBDDs, generated in the previous phase provide a decision tool for participants. Nevertheless, causal diagrams do not clearly illustrate which information is known by each agent neither the mechanism through which an agent becomes aware of the information contained on its process view. In this phase of the methodology we will design a communication protocol that makes explicit the information exchange required for a correct implantation of the process.

It is generated the OCDD for the process view of each participant of an organizational process for detecting the information needs of each one of them. Then it is defined a policy for generating the required messages, with format $MSG(sender, receiver, when, content)$ (see Section 6.2.5). The policy followed for generating the protocol is simple: 1) the PMO controls the process execution and is the information hub, 2) the PMA initiates the execution of the PMO and compiles the results captured by it, 3) actions are enabled by the PMO or by the PMA if the action occurs previous to the creation of the PMO, and 4) each agent is directly aware of the effects of its actions.

For every process specification $P$ follow these steps:
7.2. THE CMAS METHODOLOGY

CP-1. Generate the OCDD. For every process view \( PV_i \) of \( P \) controlled by the participant \( Part_i \), generate a OCDD using all available \( S(PV_j,O_i) \).

The minimal set of covariates \( Z_i \) for each strategy \( S_i \) is given by the set \( \text{ancestors}(V_F) \cap \text{descendants}(X_i) \), where \( V_F \) is the variable associated to \( F \) in the SCN and \( X_i \) is the first control variable in \( S_i \). Then remove covariates \( C_i \) referenced in the contextual condition \( C \) of \( PV_j \) from \( Z_i \).

Identify each combination of realizations \( z_i \), such that

\[
\sum_{v_f \in F} P(v_f|z_i, S_i, C) > 0
\]

Now form the OCDD using the sets \( path_i = \{ z_i \cup C \cup x_i \} \), such that \( x_i \in S_i \). The root of the OCDD is the variable \( V_i \) with higher precedence in the SCN graph.

CP-2. Messages. Traverse each OCDD \( \overline{S}(PV_j, O_i) \) controlled by a participant agent \( Ag_i \) other than the PMA and the PMO, for generating the messages required by the participants. Actions \( X_i \) directly controlled by an agent \( Ag_i \) that is not the PMA or PMO of the process produce a message \( MSG(PMA/PMO, Ag_i, \{ X_i = True \}, \{ X_i = True \cup pa(X_i = True) \}) \), for each \( pa(X_i = True) \) such that \( P(X_i = True|pa(X_i = True)) > 0 \); the sender is the agent enabling \( X_i \).

Covariate realizations \( z_i \) located before than \( X_i \) and not included in \( pa(X_i = True) \) produce a message \( MSG(PMA/PMO, Ag_i, \{ z_i \}, \{ z_i \}) \). Covariate realizations \( z_i \) located immediately after \( X_i \) produce a message \( MSG(Ag_i, PMA/PMO, \{ Z_i = z_j \}, \{ Z_j = Z_j \}) \) for each \( Z_j \in dom(Z_i) \) whenever \( Z_i \notin U \).

These messages are send/received to/from the PMA if the action \( X_i \) is enabled by it; respectively for the PMO.

CP-3. PMA and PMO Messages. Add a message \( MSG(PMA, PMO, z_n, z_c) \) where \( z_c \) is the contextual condition of the PMO’s process view and \( z_n \) is the covariate realization \( z_n \in z_c \) lowest located in the OCDD. In the same way, add a message \( MSG(PMO, PMA, f_i, f_i) \) for each \( f_i \in F \).

CP-4. Synthesize. All the resulting messages are put together, removing duplicates.

Then they are sorted traversing the OCDD of the process, according to the when condition. Messages associated to the same variable realization \( v_i \) are sorted in order of precedence: first the messages on which the sender doesn’t receive other messages with when = \( v_i \); if any cycle exist in these groups of messages, choose arbitrarily a first message.

Two contiguous messages \( m_1(Ag_i, Ag_j, v_i, z_1) \) and \( m_2(Ag_i, Ag_j, v_i, z_2) \) can be merged; the resulting message would be \( m_3(Ag_i, Ag_j, v_i, \{ z_1, z_2 \}) \). Also, messages \( m_1(Ag_i, Ag_j, v_1, z_1) \) and \( m_2(Ag_i, Ag_j, v_2, z_2) \) can be merged if \( v_1 \) precedes \( v_2 \) in the OCDD and there is no action node between them; the resulting message would be \( m_3(Ag_i, Ag_j, v_2, \{ z_1, z_2 \}) \). Merged messages are removed and the new message is added to the protocol.
CHAPTER 7. THE CAUSAL MULTI-AGENT SYSTEM

CP-5. Sequence Diagram. Represent messages in a UML sequence diagram. Represent every participating agent \( Ag_i \), including the PMA and the PMO, and set a life line from the first message send or received by \( Ag_i \) down to the last message send or received by it.

Arrange messages in combined fragments using the process OCDD. Traverse the OCDD and for each OCDD node \( N_i \) identifying the variable \( V_j \), draw the messages \( MSG(Ag_i, Ag_j, V_i = v_i, [Z_i = z_i]) \) in the sequence diagram from the lifeline of \( Ag_i \) to the lifeline of \( Ag_j \), indicating \( [Z_i = z_i] \) as the content of the message.

If two or more branches diverge from an OCDD node, it is incorporated a Parallel or Optional fragment, depending on the number of branches. Each operand of is labeled with \([V_i = v_1, ..., V_i = v_n]\) where \( v_i \) is labeled in the diverging branch.

If two operands are labeled with the same \([V_i = v_1, ..., V_i = v_n]\), both operands are merged, grouping together their messages and removing duplicates.

CP-6. Quantifier blocks. Finally, ALL-quantifiers in action parameters introduce parallel blocks indicating that the referred action and the subsequent events and actions will be the values such parameters hold. The parallel block starts in the action variable \( X_i \) and ends on the final state. Successive actions with ALL-quantifiers produce nested parallel blocks.

Message codification

The content of protocol’s messages is given by the annotations of those realizations used for labeling message arrows in the sequence diagram. Semantic variables contained in these annotations are replaced by actual entities corresponding to the SCN case previous to the information delivery.

Messages are codified according to FIPA ACL specifications\(^7\) as detailed in section 6.2.5.

Additionally, content annotations can be mapped into propositions and arguments and use a content languages like SL\(^8\). Given an annotation \( Ann(Z_i = z_i) = \langle V_i, q \rangle \), \( q \) is represented by a proposition name and distinguished variables \( v \in V \) are defined as arguments of the proposition. Argument types can be obtained from the declaration of these variables, whereas it is contained in the same annotation or in any other annotation of the process SCN.

72.10 Building the Organizational Ontology

Schemas and individuals that constitute the Organizational Ontology (OO) are obtained from the SCN annotations used to specify organizational processes. The OO is codified in OWL following these steps:

\(^7\) FIPA Agent Communication Language specifications. \( \text{http://www.fipa.org/repository/aclspecs.html} \)

\(^8\) FIPA SL Content Language Specification. \( \text{http://www.fipa.org/specs/fipa00008/index.html} \)
7.2. THE CMAS METHODOLOGY

OO-1. The Organization class. Define the organization class $OI$ as a subclass of $IntelligentOrganization$ and set a $Person$ instance as its efficient cause (the system administrator).

OO-2. Main organization elements. Use DL classes $Goal$, $Metric$ and $Process$ for declaring individuals representing organizational goals, metrics and processes. Associate these individuals to the organizational class $IO$ through a $hasValue$ constraint using properties $pursues$, $observes$ and $implements$, respectively. Goals' importance factor is set as an attribute of goal individuals. Assert the relation $satisfies(P,G)$ between related processes and goals.

OO-3. Entity types. Define the subclasses of $Person$, $SoftwareAg$, $Knowledge$, $Information$, and $Resources$ stored in the relation $types$ of each process definition. For each $Person$ or $SoftwareAg$ type $Ag$ assert $OI \subseteq 0$ $hasParticipant.Ag$. Similarly, for each $Knowledge$ and $Information$ type $K$ assert $IO \subseteq 0$ $uses.K$. Finally, assert $IO \subseteq 0$ $controls.R$ or $IO \subseteq 0$ $consumes.R$ for every $Resource$ type $R$.

OO-4. Actions. Each annotation $Ann(X_i = True)$ is used for declaring an action class $\alpha Action$ with properties representing action causes and effects in terms of entities. $\alpha Action$ is declared as potential of the agent class controlling $X_i$. i.e. $AgClass \subseteq 0$ $do.\alpha Action$.

The property $causes$ is constrained in $\alpha Action$ indicating the type and cardinality of caused entities, identified in the effect covariate $Z_i (X_i \rightarrow Z_i)$; entities modified but not created by effect of $\alpha Action$ are not associated through this property. Additionally, custom properties are defined in the ontology and constrained in $\alpha Action$ to identify action causes or parameters represented in $Ann(X_i = True)$ like $prop_i, val_i$.

OO-5. Entities properties. In covariate annotations, identify properties, their range and domain. Range is used for expressing potential properties as well. Define the identified properties in the OO.

OO-6. Process view plans and goals. Process views are defined as essential plans for the corresponding agent classes. Define a plan for each generated process view $P_V$ and associate it to the corresponding agent class $Ag$; use the class $Plan$ and the constraint $Ag \subseteq controls = P_V$, respectively.

In correspondence, an essential goal $G_A$ must be generated for each process view $P_V$ controlled by agent $Ag$. Define the goal $G_A = (Ann(C), \Omega)$ where $\Omega \subseteq Ann(F_i)$ for each $F_i \in F$ and $\Omega \neq \emptyset$; then assert $Ag \subseteq pursues = G_A$. If there is no condition $\Omega$ capable of satisfying this criterion, introduce a common condition $\Omega$ in every $F_i$.

OO-7. Agent goal preferences. Define an essential goal preference for each realization $(Z_F = z_F) \in F$ of every process view $P_V$ controlled by the agent class $Ag$. If an organizational metric $M = (C, p)$ was identified in $Ann(Z_F = z_F)$, define an
essential goal preference \( EGP_i = (Z_F = z_F, p) \) and associate it to \( Ag \) asserting \( Ag \sqsubseteq hasGoalPreference = EGP_i \); besides assert \( derivedOf(EGP_i, M) \) to indicate the origin of the goal preference. If none organizational metric is identified in \( Ann(Z_F = z_F) \), define an \( EGP_i = (Z_F = z_F, importance(G_p)) \) where \( G_p \) is the goal satisfied by \( P_V \), and associate it to \( Ag \) asserting \( Ag \sqsubseteq hasGoalPreference = EGP_i \).

**OO-8. Organizational side effects.** If a general organizational metric \( M = (C, p) \) was identified in an organizational process \( P \) through the introduction of a covariate \( Z_i \) and the annotation \( Ann(Z_i = z_i) \), and \( Z_i \) is present in a process view \( P_V \) controlled by \( Ag \), define an organizational side effect \( SE_i = (Z_i = z_i, p) \) and associate it to \( Ag \) asserting \( Ag \sqsubseteq observes = SE_i \). Besides assert \( derivedOf(SE_i, M) \) to indicate the origin of the side effect.

**OO-9. Organizational assets.** Identify the organizational assets required for initializing the system as well as their location. To do so, elaborate a table containing the types of entities defined in the process, its category, the location on which they will reside, and the list of initial assets. The location can be the OO or an external repository. Initial information located in an external repository is not listed in the table; instead, the corresponding retrieval mapping is indicated.

A subclass of the Repository class is created to define every external repository and the type of information or knowledge contained on it. Initial organizational assets and repositories listed in the table are declared in the OO as instances (individuals) of their respective class. These instances are associated to the \( IO \) class through a hasValue constraint. Finally, retrieval mappings are declared in the K-I interpreter.

Next document organizational processes in the Process Repository using the SCN graphs built in previous phases. For each process \( P \in \overline{P} \) declare \( M_P \) in the Process Repository following these steps:

**PR-1. Variables.** Declare random variables \( (V_i) \), their domains \( (v_{i,j}) \) and realizations \( (V_i = v_i) \). Variables depicted with a single expression, as well as control variables, are boolean. The domain of variables with labeled expressions is constituted by these labels.

**PR-2. Causal Relations.** Declare causal relations between variables \( (V_i \rightarrow V_j) \).

**PR-3. Annotations.** Declare the annotation of variables realizations \( (Ann(V_i = v_i)) \).

**PR-4. Probabilistic distribution.** Build an initial probabilistic distribution declaring conditional probabilities of the form \( P(V_i = v_i|PA_i = pa_i) \), where \( PA_i = pa_i \) represent a combination of realizations of parents of \( V_i \).
PR-5. Process SCN. Declare the SCN $M_P$ representing the process structure and associate declared variables, causal relations, conditional probabilities and annotations to it. Finally, in the OO register the relation between the organizational process $P$ and $M_P$ asserting $\text{specifies}(M_P, P)$.

Variables are associated through one of two predicates to the SCN: $\text{hasCovariate}(Z)$ and $\text{hasControlVariable}(X)$. Additionally, unobservable variables $(U)$ are distinguished through the predicate $\text{hasUnobservableCovariate}$.

PR-6. Action quantifiers. Express action ALL-quantifiers indicating the semantic variable name and the action node $X_i$. ONE-quantifiers are assumed by default.

PR-7. Strategies. Strategies $S_i$ and $W_k$ sets identified in every SCN are saved indicating the partial order $O_i$ used for calculate them.

PR-8. Protocols. Save the protocols, with their respective messages for each $P_i \in \overline{P}$.

Process views are declared in the Process Repository as well, but their construction reuses must of the elements declared for the original process diagram. It is chosen a subset of variables with their respective domains and realizations. It is built a new graph constituted by causal relations among these variables. Annotations are mainly borrowed from the process diagram; annotations modified in the process view are overridden. Finally, the probabilistic distribution of every process view is calculated using the one specified in the original process SCN and adjusting by marginalization based on the new causal graph.

7.2.11 Validating the specification

The specification is validated semantically at two levels: general and detailed. The Organizational Ontology is evaluated in general through a consistency checking of the asserted schemas and individuals. The definition of an agent participating in different processes can be inconsistent due to the assertion of conflicting constraints. Actual DL reasoners like Pellet\(^9\) or RacerPro\(^{10}\) are capable of detecting such inconsistencies.

A detailed validation of the specification consists on evaluating the conjunctive queries used for representing conditions and annotations, as well as the relations explicitly asserted or identified through a further analysis. Subsumption relations between conditions or annotations is registered in the OO indicating the equivalence between variables used in every pair of them. It is validated or identified: 1) well formed conditions and annotations, 2) inconsistencies between variable annotations, 3) expressed plan-goal satisfaction, and 4) enabling plans.

V-1. Conditions and annotations. Every condition and annotation used on the definition of processes, goals and metrics must be a well formed conjunctive query.

---


\(^{10}\) RacerPro. Racer Systems GmbH & Co. KG. http://www.racer-systems.com/
Besides, concepts and roles used in the query must be defined in the organizational ontology.

**V-2. Annotations inconsistencies.** Detect inconsistencies in annotations used in the specification of an organizational process. For each covariate \( V_i \in Z \) identify the set of annotations \( \text{Ann}(V_i = v_i) \). For each pair of annotations \( Q_1 = \text{Ann}(V_i = v_1) \) and \( Q_2 = \text{Ann}(V_i = v_2) \) in \( \text{Ann}(V_i) \), such that \( Q_1 \neq \top \) and \( Q_2 \neq \top \), check:

1. If \( v_1 = v_2 \), \( Q_1 \subseteq Q_2 \) and \( Q_2 \nsubseteq Q_1 \), establish that \( \text{Ann}(V_i = v_1) \) must be evaluated before \( \text{Ann}(V_i = v_2) \). If \( Q_1 \equiv Q_2 \), the order of evaluation is not relevant: \( Q_1 \) and \( Q_2 \) are different ways of expressing the same event.
2. If \( v_1 \neq v_2 \) and \( Q_1 \nsubseteq Q_2 \), report \( V_i \) as inconsistent.

**V-3. Annotation rewriting.** Calculate query rewriting for each annotation of the model according to the Organizational Ontology. Consider the set of conjunctive queries \( q_i \) associated to \( \text{Ann}(V_i = v_i) \) as a union of conjunctive queries (UCQ) \( q' \) and calculate the UCQ \( q'' \) resulting of a rewriting algorithm like [79, 24]. \( q'' \) is stored in an auxiliary repository in order to be used for query answering at runtime.

**V-4. Plan-Goal satisfaction.** For every goal \( G = (A, \Omega) \) pursued by the organization, identify in the OO those processes that satisfy it (satisfies(\( P, G \))) and identify in the PR their process specifications \( M_P \) (satisfies(\( M_P, P \))).

1. If \( A \neq \top \), check that \( A \subseteq \text{Ann}(C) \).
2. Check that \( \Omega \subseteq \text{Ann}(F_i) \) for every \( F_i \in F \).

**V-5. SCN strategies.** Validate that exist at least one SCN strategy for each participant in each process.

**V-6. Enabling plans.** Identify plan conditions that can be enabled by the execution of an auxiliary plan (Def. 75). Determine if some condition of every process \( P_i \in \bar{P} \) can be set by an auxiliary plan \( P_2 \), asserting the relation \( \text{enables}(P_2, P_1, Z_i = z_i) \).

**V-7. SCN constraints.** Validate that quantified variables exist in \( \text{Vars}(\text{Ann}(X_i = \text{True})) \) for some \( X_i \in X \). Also, validate that unobservable variables have only unobservable effects (if any).

### 7.2.12 Extending Agent Classes

Organizational agent roles (OARs) defined in the OO specify minimum requirements that must satisfy agents playing each organizational role modeled through processes. Now we must define actual agent classes that implement such functionality. To do so, in
7.2. THE CMAS METHODOLOGY

the agent repository (AR) we define agent classes that extend one or more organizational agent roles.

To indicate that an agent class $Ag$ is capable of playing an organizational role $R$ is asserted $Ag \sqsubseteq R$ in the AR. An agent class capable of playing multiple organizational roles is expressed by extending as many OARs as needed, i.e. $Ag \sqsubseteq R_1 \cap ... \cap R_n$; the resulting agent class must be consistent, i.e. it shouldn’t exist conflicts between constraints contained in $R_1, ..., R_n$.

Additionally, agent class definitions might be complemented with further constraints that precise the actual limitations of the intended software implementation. Once again, introduced constraints must not be in conflict with those provided by extended OARs. A DL concept reasoner can be used for this purpose.

Finally, process views controlled by each OAR extended by the agent class are inherited by it, as well as essential goals and other constraints.

7.2.13 Developing Software Agents

Each defined agent class must incorporate the following elements:

**SA-1. Knowledge acquisition.** Select a mechanism for acquiring information and knowledge specified as cause of actions controlled by the agent. The mechanism can be: 1) pass of parameters on initialization, 2) known a priori (set as constraint in the agent class), 3) sensor input or 4) asking to other agents.

Asking to other agents can be implemented through a pair of rules (one for asking and another for receiving the answer) or through an organizational protocol. In the first case, associate each causal rule $R_i$ to the agent class $Ag$ asserting $Ag \sqsubseteq requires = R_i$.

**SA-2. Sensors.** Implement a sensor class for perceiving information from the environment as specified in SA-1. Extend the Sensor class described in section 6.2.3.

**SA-3. Resource control.** Select a mechanism for gaining control over resources defined as causes of actions controlled by the agent or as agent properties. The access mechanism to these resources can be: 1) full control, or 2) shared control. Full control indicates that the agent can make use of the resource whenever it wants. A fully controlled resource can be indicated through a constraint in the agent class or it can be passed as parameter in initialization. Shared control is mediated by an organization protocol. For shared resources, specify the type of protocol used to gain control of it and designate an agent on charge of managing the resource.

**SA-4. Actuators.** Implement an actuator class that encapsulate the access to methods or functions representing actions controlled by the agent. Extend the Actuator class described in section 6.2.3.
SA-5. Plan rules. Define those immediate causal rules summarized in covariate-covariate causal relations defined in essential plans controlled by the agent. Associate each causal rule \( R_i \) to the essential plan \( P \) class asserting \( P \sqsubseteq \text{requires} = R_i \).

### 7.2.14 Designing Organizational Protocols

Knowledge acquisition and resource control protocols defined in the previous phase are modeled following the same methodology of organizational processes. But instead of pursuing an organizational goal these protocols are modeled pursuing a subgoal derived from current agent plans (see Section 6.3.1). Likewise, an agent participation protocol that indicates the method for recruiting agents should be modeled too.

These protocols, called *organizational protocols*, are controlled by the system PMA, which enables them by setting their necessary conditions. The system PMA can play the role of PMO if the process is not very complex.

Resulting process views are associated as essential plans to agent classes that access knowledge/information repositories, request control of shared resources or request the participation of other agents. These protocols are initiated through subgoaling by agents participating in organizational processes when some missing condition is detected.

Organizational protocols are stored in the Process Repository at the OO. Agents are associated to such protocols by asserting \( AgCls \sqsubseteq \text{controls} = \text{Protocol} \).

### 7.3 CMAS Tools

I developed a set of tools for verifying the specification and generating automatically models used in the implementation of the agents. The original process specification is used for generating process views, OBDDs, OCDDs and the communication protocol. These tools were developed in Java using the Jena Framework.

The main features provided by these tools include the following:

- Calculates conjunctive query subsumption and generates a mapping between query variables.
- Augments an original process specification introducing the PMA and the PMO.
- Extract process views for each participant agent.
- Calculates the strategies, build the OBDD and the OCDD for each partial view.
- Produces graphical representations of SCNs, OBDDs and OCDDs in the GraphViz format\(^\text{11}\).

\(^{11}\) Graphviz. [http://www.graphviz.org/](http://www.graphviz.org/)
7.4 CMAS OPERATIONS

- Generates a preliminary protocol that can be customized.
- Run the validation tests given in section 7.2.11.
- Initializes a CMAS from an intelligent organization semantic specification.

7.3.1 CMAS Log

Events, messages and closed cases are stored in a database for their analysis. Events, indexed by agent and the cycle on which they occur, contain a description of actual goals and trials, decisions made, evidence observed on each trial, etc. Messages exchanged between agents are indexed by the protocol that produces the message. Closed SCN cases are uploaded by each agent every time a goal concludes.

The CMAS log is queried using some facility provided by the database manager using standard SQL queries.

7.4 CMAS Operations

In this section I describe the operations that system and process administrators can perform in a CMAS. These tasks include the initialization/termination of the system or an organizational process. Human supervision allows a correct tuning of active processes.

7.4.1 Starting up

Having defined the Organizational Ontology, the agent repository and the K-I interpreter, the system administrator can initialize the MAS platform. The initialization of the CMAS is made through the instantiation of a PMA, \( PMA_O \), that manages the system process which receives as parameters: the Intelligent Organization class (IO), ontological repositories and the MAS platform.

The \( PMA_O \) initializes common services and creates an instance of IO that satisfy all the constraints expressed on its definition. This instance represents the actual intelligent organization. From the IO's definition, the \( PMA_O \) obtains the user name of the system administrator and the list of organizational processes to deploy. Each organizational process \( P \) is initiated through the corresponding \( PMA_P \).

7.4.2 Following up

The system administrator can use the KMS for querying the OO to obtain the status of a process. Additionally, system and process administrators use the facilities provided by their User agents for obtaining information about their respective process. If a process or
system administrator is inquired by its User agent respect certain condition that avoids continuing with the execution of the process, he/she can provide additional information or modify certain data in correspondence with the actions made in the real organization. User agents can implement some HC interface for providing information to human users through dashboards containing relevant indicators.

### 7.4.3 Tuning

Process administrators, including the system administrator, can adequate their respective processes adjusting their organizational goals and metrics. Such changes are propagated among the agents participating in the process as indicated in section 6.4.3. Other adjustments are made directly in the OO; for instance, the confidence on a rule used by agents.

Tuning is measured by process administrators every certain period of time, in order to determine if the introduced changes were beneficial for the process metrics. To do so, SCN cases representing process executions are arranged in periods of time comprehending different policies and summarized for comparing the effects of the policy change. In a meta-learning level, a SCN can be used for learning from process policy changes.

### 7.5 Summary

I presented a multi-agent system architecture called *Causal Multi-Agent System (CMAS)* in which organizational processes are implemented through Causal Agents that enable the participation of human users and intelligent agents. I also proposed a methodology that guides the modeling of organizational processes and the required Causal Agents. Finally I presented a brief overview of the operation of a CMAS.

The *CMAS architecture* is constituted by a set of ontological repositories, managed by an administrator through a Knowledge Management System, that contain: 1) the domain schemas and organizational assets, 2) the specification of organizational processes, 3) agent implementations, and 4) mappings for accessing external resources.

Organizational processes are enacted by agent instances deployed by human administrators in a FIPA-compliant multi-agent platform. Agents are arranged in three layers constituted by: 1) user agents, 2) process-dedicated agents, and 3) shared agents. Agents access external resources like databases through a Knowledge-Information Interpreter (K-I Interpreter).

The *CMAS methodology* proposes the use of SCNs for modeling organizational processes. In an initial phase, every organizational process is described through a goal, the sequence of actions that lead to its achievement, and the causes that intervene (roles, information, knowledge and resources). In a second phase, it is introduced the participation of autonomous agents, human supervision and organizational metrics, concluding with an
7.5. SUMMARY

automated process specification.

In the third phase of the methodology there are introduced process manager (PMA) and process monitor (PMO) agents for coordinating the process execution through a communication protocol. Every agent controls a partial view of the process and plays a role in the protocol. The fourth phase consists in the ontological modeling of the organization, processes, agents and actual assets, and concludes with the validation of these definitions. The last phase consists in the development of software agents compliant with the roles defined in the processes. Auxiliary protocols derived from agent implementations are modeled as well.

This section concludes describing the tools developed for automatically generating and validating the models required for agent implementations, and briefly describe the CMAS operation, indicating how to start up a CMAS, how to monitor the development of the organizational process and how to make adjustments for modifying its performance.
Chapter 8

A Case Study on Autonomic Information Auditing

The Tecnologico de Monterrey (ITESM) is a university with operations in 33 campuses in Mexico, sixty thousand undergraduate students, twelve thousand graduate students, eight thousand faculty members, of which around twelve hundred are research professors. Being in process of becoming a research university, ITESM has allocated financial resources for supporting research groups constituted by professors, assistants and students. As a result, research products, projects and groups proliferated becoming evident the necessity of managing assets generated from research activities. To do so, ITESM adopted knowledge management procedures to manage its research assets through a knowledge-based information system that assist officers and researchers in decision making [15].

Policies and regulations were established to support the operation of the corporate memory, being developed a computer platform for storing and distributing information of research assets [16]. Additionally to information obtained from institutional databases, the corporate memory is feed by professors and students who register, catalogue and relate their own research products, publications and thesis for instance.

To warrant the generation of accurate information for decision making, it was institutionalized an information auditing process. Auditing verifies the classification of stored research assets, their internal consistency and the overall consistency of the repository. In the beginning, this process was performed by human auditors, but as the amount of information increased the necessity of automation became evident. The intensive use of knowledge developed by auditors made the use of intelligent agents a natural choice in the automation process.

In this chapter I briefly describe the structure of the information system and the auditing process. Then I present the objectives pursued in the automation of this process and present a Causal MAS designed with this purpose. The modeled process is implemented twice: first using the Electronic Institution formalisms and tools developed at the IIIA-CSIC, and then using the Causal Agent architecture. I conclude this chapter presenting
8.1 A Knowledge-based information system for managing research programs and value creation

The knowledge-based information system is constituted by several interdependent modules: researchers, students, research groups, research centers, graduate programs, research projects, thesis and publications. Each module counts with web interfaces for capturing, classifying and relating research assets. In the same way, reports and indicators are delivered through web controlling the level of clearance of every user according to their organizational position.

Even though information resides in a central database, procedures and web interfaces are arranged conceptually in modules, facilitating the operation and maintenance of the system. The Publications module is one of the more delicate modules given the large amount of information feed by users from heterogeneous data sources. Publications are stored using common attributes and classified according to an institutional taxonomy. This taxonomy is used for qualifying the scientific production of professors and groups which in consequence determines their institutional allowance.

Figure 8.1 shows publication metadata stored in the publications module. Expert auditors have elaborated rules, expressed using publications metadata, for checking the internal consistency of records and the overall consistency of the database, and for correcting the records. Data along with inconsistency and correction rules constitute the publications repository.

One example of overall repository inconsistency is the duplicity of a publication record which consists in the existence of two publications in the repository having such a degree
of similarity that make the auditor suspect that both of them are actually the same publication. This kind of inconsistency is favored by the shared storage of publications with multiple authors.

Other example of internal inconsistency in publication records is detected when the status of the publication and the year of publication do not correspond each other. Given that it is allowed that professors register its production in progress indicating a tentative publication date and a publication status, it is possible that both of them become inconsistent with the pass of time. Publication status and the transitions between them are shown in Figure 8.2.

For correcting the first type of inconsistency, one of the two records is marked for deletion. The record with more information prevails. But if both records have different information, the auditor must identify which one is the correct one, even searching in the Web. If the auditor cannot determine which is the right information (s)he can ask the corresponding author. Anyway, the corresponding author is notified of the deletion in order to allow him/her to reply.

In the correction of the second type of inconsistency there are two possible choices: 1) changing the publication date, or 2) changing the publication status. In some cases the choice is evident, but in others is not. For instance, if an accepted publication has a past publication date, the status can be changed to published. Nevertheless, a publication in progress with a past publication date can be corrected by changing its publication date to a future date or changing its status to published. In the last case none of these correction rules have been proved successful all the time.

As can be seen, none of the previous problems can be detected online during information registering nor the correction is totally trustworthy in all cases. Inconsistencies detection must be done periodically off line and corrections must be supervised by expert auditors.
and authors.

8.2 Autonomic Information Auditing

In order to automate the process of detecting and correcting inconsistencies with a certain degree of autonomy, we must consider some factors. The first of them is the capacity for expressing expert rules with an adequate formalism that supports missing data. The application of these rules may produce the discovery of new rules or can demonstrate (statistically) that the rule is inaccurate. The expert auditor should be capable of providing new inconsistency rules and determining how and when they should be applied.

Figure 8.3 shows a causal diagram illustrating the auditing and correction process outlined so far. This diagram shows the required actions and the possible choices that human users and autonomous agents can make. The autonomous agent on charge of detecting inconsistencies can request: 1) automatic correction (following some rule), 2) request expert assessment if there are no confident correction rules, or 3) notify the corresponding author if the expert auditor is not capable of solving the problem. The autonomous agent on charge of correcting the inconsistency may request expert assessment as well if it is not sure of the correction rule applied. The expert auditor can approve or reject the proposed correction or choose another correction. The author has a final decision in case of controversy.

Human supervision of expert auditors and authors should be minimized on the measure that autonomous agents learn how accurate are its given rules. Nevertheless, minimizing human supervision might compromise the confidence on information as long as this
8.3. CAUSAL MODELING OF AIA

Table 8.1: AIA process goal.

<table>
<thead>
<tr>
<th>Name*</th>
<th>AuditedPub</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Maintain audited every publication in the repository.</td>
</tr>
<tr>
<td>Whenever (A)</td>
<td>Publication(?pub). Repository(?rep). in(?pub,?rep)</td>
</tr>
<tr>
<td>Achieve (Ω)</td>
<td>status(?pub, AUDITED)</td>
</tr>
<tr>
<td>Importance *</td>
<td>10</td>
</tr>
</tbody>
</table>

is determined in terms of how accurate are the rules applied and how much human supervision was involved. Both objectives are in conflict.

The solution proposed for this problem can be formalized in terms of autonomic computing through the following objectives:

**Self-configuration.** Adapt the process specification given by an expert to actual behaviors of internal and external participants.

**Self-optimization.** Minimize unnecessary supervision by human users and maximize the confidence in the repository consistency.

**Self-protection.** Detect inaccurate expert knowledge represented by inconsistency and correction rules.

**Self-healing.** Switch between auditing strategies if conditions change along time.

This lead us to coin the term *Autonomic Information Auditing (AIA)* for this approach we propose to information auditing.

8.3 Causal modeling of AIA

Following the methodology described in Section 7.2 we specified and automated the auditing process. AIA objectives form an essential part in the specification of the organizational process.

8.3.1 Organizational goals and metrics

We started defining the main goal in the auditing process as illustrated in Table 8.1. This goal doesn’t specify when or how to audit the information, it only establishes that every piece of information must be audited (indicated through the property *status*).
Self-optimization objectives of AIA are used to define organizational metrics. The objective of maximizing the repository confidence is formalized as a process metric that qualifies the achievement of the main goal. The objective of minimizing unnecessary human revision is formalized as a general organizational metric. Both organizational metrics are illustrated in Tables 8.2 and 8.3, respectively.

In the first metric are indicated positive preferences in different degrees privileging the highest confidence evaluation. Meanwhile with the second metric we penalize unnecessary human revision with a negative preference.

### 8.3.2 The AuditNewPub process

One way of auditing the information consists on verify every new publication added to the repository. This function is modeled as an organizational process. The main attributes of this process are shown in the table 8.4. Note that the triggering condition \( C \) contains the precondition of the organizational goal AuditedPub, which implies that AuditNewPub can satisfy AuditedPub, in certain way.
Table 8.4: Organizational process: Audit new publications.

The original process

Following the CMAS methodology I modeled the process AuditNewPub just as it was originally performed by a human auditor. The modeled process is shown in Figure 8.4. The initial and final nodes are annotated with the precondition and final condition of AuditNewPub. The expert auditor is represented by the Auditor class, meanwhile actions audit and correct are declared as his competencies. Web interfaces are identified as material causes, meanwhile inconsistency and correction rules represent formal causes in the process.

I introduced a quantifier in the audit action indicating that this should be performed in every publication using every inconsistency rule available in the repository. Likewise, it was indicated that the correct action should be done over every inconsistency detected.

Finally observe that the SCN has been adjusted for modeling every inconsistency rule and correction rule. This is represented labeling the corresponding annotation with the
variable name, \( ?\text{rule}\) and \( ?\text{rule}\) respectively.

### The automated process

Then I introduced software agents for performing some of the tasks originally developed by the human auditor and indicated his supervision. The SCN illustrating the automated process with organizational metrics is shown in Figure 8.5.

The automated functions are the following: 1) monitoring new publications in the repository, 2) detecting inconsistencies, and 3) applying corrections. These functions were delegated respectively to the following internal agent types: \( \text{LogMonitorAg} \), \( \text{AuditorAg} \) and \( \text{CorrectorAg} \). The \( \text{LogMonitorAg} \) was endowed with a function that it was not originally considered, but that it was necessary for detecting the feeding of new publications in the repository.

On the other hand, \( \text{AuditorAg} \) and \( \text{CorrectorAg} \) replaced the participation of the human auditor, but motivated the incorporation of a supervision phase. In this way, the human auditor is on charge of revising the correction made by the \( \text{CorrectorAg} \), being able of rejecting its automatic correction. The \( \text{CorrectorAg} \) can avoid requesting such supervision if its confidence on its correction is high enough. Skipping auditor’s supervision is indicated by the annotation \( \text{Ann}(Z_{16} = D) \). Likewise, is modeled the participation of the corresponding author in the process. Annotations \( \text{Ann}(Z_{16} = G) \) and \( \text{Ann}(Z_{17} = D) \) represent a request neglected by the human user. In this specification, author’s participation can be requested only by the expert auditor.

I included an annotation for the negative value \( (Z_i = \bot) \) of intermediate conditions and organizational metric variables. Intermediate conditions need to be instantiated in order to proceed with process execution. The use of negation as failure doesn’t allow waiting the response of other participants regarding the result of their actions. Such variables are \( Z_{4}, Z_{7}, Z_{13}, Z_{16}, Z_{17} \) and \( Z_{18} \). \( \text{Ann}(Z_{13} = \bot) \) is reused in \( Z_{16}, Z_{17} \) and \( Z_{18} \), as long as the negative realization of these variables is consequence of an incapacity for correcting an inconsistency. Variables representing organizational metrics and the final state, \( Z_{20} \) and \( Z_{19} \), must be capable of representing that the desired condition was not reached.

The \textit{PubConfidence} metric is incorporated in the final node \( (Z_{19}) \) of the automated model by introducing three possible final states. Each final state is identified by a different label as long as they are valued distinctly w.r.t. this organizational metric. The \textit{MinHumanIntervention} metric is incorporated in the process adding the node \( Z_{20} \) which represents unnecessary human supervision of the auditor or the author. Arcs indicate the causal dependency between \( Z_{16}, Z_{17} \) and \( Z_{20} \), which will later be translated into immediate causal rules.

Before that, agents were only able to determine if their actions contributed or not to the goal of the process. Now agents can mediate their decisions based on these organizational metrics. For instance, the decision of the \( \text{CorrectorAg} \) for requesting or
8.3. CAUSAL MODELING OF AIA

Figure 8.5: The automated auditing process.
CHAPTER 8. A CASE STUDY ON AIA

not the supervision of the expert auditor now can consider both conflicting metrics.

A simplified version of the automated process is shown in Figure 8.6. In this diagram we only represent actions (in boxes) and intermediate states (rounded boxes). The initial and final state are distinguished in the same way, with a thick line and double line in the outline, respectively. Action causes are removed for making the diagram clearer. Annotations are replaced by textual descriptions; in consequence arc labels are removed too.

In this diagram can be observed clearly the type of workflows that constitute the process. Actions $X_1$ and $X_2$ are executed sequentially, meanwhile $X_3$ through $X_8$ can or cannot be executed conditioned on $Z_7$. Action $X_9$ (expert assessment) is considered optional as long as there is an alternative path from $Z_{13}$ to $Z_{16}$. Finally, $X_5$ and $X_6$ are mutually excluding, conditioned on $Z_{16}$.

The augmented process

A new agent class called $RepGuardianAg$ was designated as administrator (PMA) of the auditing process, meanwhile another new agent class called $PubCarrierAg$ was designated as the monitor (PMO) of the process. The augmented process specification is shown in Figure 8.7.

As can be seen, variables $Z_{PMA}$ and $Z_{PMO}$ were introduced as causes of actions, repre-
Figure 8.7: The AuditNewPub augmented process.
senting that their participation is required for executing such actions. For the PMA and PMO, this means that they are responsible for enabling such actions. The contextual condition was recalculated and now is $C = \{Z_1 = \text{True}, Z_2 = \text{True}, Z_{\text{PMA}} = \text{True}\}$.

**AuditNewPub Process Views**

Process views for *RepGuardianAg* and *PubCarrierAg* were generated from the augmented process model as indicated in section 7.2.7. The *RepGuardianAg* enables repository monitoring ($X_1$) whereas the *PubCarrierAg* participates in the process until the new publication is detected ($Z_4 = \text{True}$).

The rest of the agent process views consisted mostly of SCNs with a single action, with a custom initial condition ($C$) and heading to the process goal ($F$). Variables $Z_{19}$ and $Z_{20}$ were included as unobservable variables as long as they represent organizational metrics and are not direct consequence of the action of none of the participants.

The *CorrectorAg* was the only one agent controlling two actions in the process (see Figure 8.8). Its contextual condition considers only variables $Z_7$ and $Z_{\text{PMO}}$ because $Z_9$ and $Z_{10}$ do not have parents in the augmented SCN.

**AuditNewPub Strategies**

Figure 8.9 shows the strategies identified in the process. Its calculation took approximately four hours and produced five valid strategies according to the SCN.
Similarly, strategies of each partial view were calculated. According to them, human roles Author and Auditor are allowed to act or not in the process. Their participation is conditioned to certain conditions of the process, identified by the respective \( W_k \) sets obtained on plan identification.

Strategy identification didn’t throw any valid strategy containing \( X_5 = True \) and \( X_6 = True \), as long as they are mutually excluding with respect to \( Z_{16} \). This validates Theorem 1 which says that is possible identifying strategies in combinations of workflows, including alternative choices. The only valid strategy for LogMonitor and AuditorAg is acting, as long as the actions they control (\( X_1 \) and \( X_2 \)) are mandatory in the process. The CorrectorAg, in the other hand, controls two actions, \( X_3 \) and \( X_6 \), both of them considered optional in the general process. Nevertheless, the partial view of this agent has \( Z_7 = B \) as contextual condition, which becomes \( X_3 \) mandatory, as reflected on its strategies; see Figure 8.10.

The communication protocol

In order to design the communication protocol we built the OCDD of each process and its respective process views. The OCDD of the whole process is shown in Figure 8.11. The OCDD starts with variables included in the contextual condition, \( Z_1 \) and \( Z_2 \), ordered sequentially. Descending from it, each causal chain goes down to \( Z_{19} \), which identifies
the goal of the process.

The exchange of messages obtained from partial view OCDDs is shown in Figure 8.12. There are represented two parallel fragments introduced by multi-valued variables $\text{?pub}$, $\text{?rule1}$ and $\text{?inc}$. Labels on arrows represent the possible messages the sender can inform. If the message is labeled with a variable name $(Z_i)$ means that the message can contain any possible value $z_i \in \text{Dom}(Z_i)$.

The participation of the $\text{RepGuardianAg}$, the $\text{PubCarrierAg}$ and the rest of the participant agents in the process is illustrated by the corresponding lifelines. For instance, the $\text{LogMonitorAg}$ only initiates the process execution notifying to the $\text{RepGuardian}$ of the appearance of a new publication in the repository. The $\text{PubCarrier}$, on the other hand, directs the process execution informing to other agents about conditions that motivate their participation in the auditing process. As well, the $\text{CorrectorAg}$ participates actively through $X_3$ and $X_6$, being the second conditioned to $Z_{16} = C$.

Parallel execution, optional blocks and choices are represented in the sequence diagram through UML combined fragments. For instance, the fork produced by the detection, or not, of inconsistencies $(Z_7)$ is represented by an alternative fragment. In this case, parallel fragments are produced by multi-valued variables, as indicated in the header of the corresponding blocks.

Having identified the messages produced in each side of the protocol, we defined ACL messages for representing the SCN annotations used in the diagram. A partial list of equivalences is shown in the Table 8.5. In messages marked with *, $\text{Auditor}$ and $\text{Author}$ types were replaced by the upper class $\text{Agent}$.

8.3.3 The AIA Organizational Ontology

The relation $\text{types}$, compiled during the modeling of the process, shown in the Table 8.6, is used to define the essential forms used in the application domain. Figure 8.13 shows
Figure 8.11: The AuditNewPub OCDD.
Figure 8.12: The UML sequence diagram for *AuditNewPub*. 
Table 8.5: Some equivalences between SCN annotations and ACL messages.

<table>
<thead>
<tr>
<th>V=s</th>
<th>Semantic annotation</th>
<th>Message declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z₅=B</td>
<td>consistentOn(?pub, ?rule)</td>
<td>ConsistentOn(Publication, InconsistencyRule)</td>
</tr>
<tr>
<td>Z₆=A</td>
<td>Inconsistency(?inc).incType(?inc, ?rule).hasInconsistency(?pub,?inc)</td>
<td>InconsistentOn(Publication, InconsistencyRule,Inconsistency[])</td>
</tr>
<tr>
<td>Z₁₆=A</td>
<td>aprovedBy(?cor, ?aud).status(?cor, C)</td>
<td>Approved(Correction, Agent) *</td>
</tr>
<tr>
<td>Z₁₆=B</td>
<td>rejectedBy(?cor, ?aud).status(?cor, U)</td>
<td>Rejected(Correction, Agent) *</td>
</tr>
<tr>
<td>Z₁₆=C</td>
<td>undoBy(?cor, ?aud).status(?cor, P)</td>
<td>Undo(Correction, Agent) *</td>
</tr>
<tr>
<td>Z₁₆=E</td>
<td>toAuthor(?cor, ?aud).Status(?cor, P)</td>
<td>ToAuthor(Correction, Agent) *</td>
</tr>
<tr>
<td>Z₁₇=A</td>
<td>aprovedBy(?cor, ?aut).status(?cor, C)</td>
<td>Approved(Correction, Agent) *</td>
</tr>
<tr>
<td>Z₁₇=B</td>
<td>rejectedBy(?cor, ?aut).status(?cor, C)</td>
<td>Rejected(Correction, Agent) *</td>
</tr>
<tr>
<td>Z₁₇=C</td>
<td>undoBy(?cor, ?aut).status(?cor, U)</td>
<td>Undo(Correction, Agent) *</td>
</tr>
<tr>
<td>Z₁₉</td>
<td>undoBy(?cor, ?corAg)</td>
<td>Undone(Correction)</td>
</tr>
</tbody>
</table>

The hierarchy of classes rooted in the class *Entity*, visualized through the OWLViz plugin for Protege.

The hierarchy of classes includes the intelligent organization class (*DSIIP*), as well as organizational agent roles (*AuditorAg, LogMonitorAg, CorrectorAg, PubCarrierAg, RepGuardianAg*) and organizational human role (*Auditor* and *Author*). Classes *SpecializedAgent, PMA* and *PMO* included in the hierarchy of software agents are used for classifying agents in layers (see Section 7.1.8).

The definition of the class *DSIIP* is given in Equation (8.1). On it can be identified the efficient cause, the implemented process, observed metrics, the pursued organizational goal, controlled resources, potential participants and the type of information used on it.

---

Table 8.6: Relation types from the process AuditNewPub.

Figure 8.13: The Entity hierarchy of classes.
DSIIP \sqsubseteq\text{IntelligentOrganization} \sqcap \text{hasEfficientCause} = \text{hceballos} \sqcap
\text{implements} = \text{AuditNewPub} \sqcap \text{observes} = \text{MinHumanIntervention} \sqcap
\text{observes} = \text{PubConfidenceHIGH} \sqcap \text{observes} = \text{PubConfidenceLOW} \sqcap
\text{observes} = \text{PubConfidenceMEDIUM} \sqcap \text{pursues} = \text{AuditedPub} \sqcap
\geq 0 \text{ controls.}(\text{Repository} \sqcap \text{WebInterface}) \sqcap
\geq 0 \text{ hasParticipant.}(\text{Auditor} \sqcap \text{AuditorAg} \sqcap \text{Author}) \sqcap
\geq 0 \text{ hasParticipant.}(\text{CorrectorAg} \sqcap \text{LogMonitorAg}) \sqcap
\geq 0 \text{ hasParticipant.}(\text{PubCarrierAg} \sqcap \text{RepGuardianAg}) \sqcap
\geq 0 \text{ uses.}(\text{Correction} \sqcap \text{CorrectionRule} \sqcap \text{Inconsistency}) \sqcap
\geq 0 \text{ uses.}(\text{InconsistencyRule} \sqcap \text{Publication}) \quad (8.1)

Actions, their causes and effects were defined as well. For instance, the action Audit receives as arguments one Publication, uses one InconsistencyRule, on its execution accesses a Repository, is performed by one AuditorAg and may produce Inconsistencies; its definition is shown in Equation (8.2).

\begin{align*}
\text{AuditAction} \sqsubseteq \text{Action} \sqcap &= 1 \text{ auditedPublication} \\
\sqsubseteq &= 1 \text{ using.InconsistencyRule} \sqcap \geq 0 \text{ accessing.Repository} \\
\sqsubseteq &= 1 \text{ effCause.AuditorAg} \sqcap \geq 0 \text{ causes.Inconsistency} \quad (8.2)
\end{align*}

In the other hand, the action Revise is performed by an Author, over one Correction and using a WebInterface; in this case, the action doesn’t produce (cause) new entities; its definition is shown in Equation (8.3).

\begin{align*}
\text{ReviseAction} \sqsubseteq \text{Action} \sqcap &= 1 \text{ supervised.Correction} \\
\sqsubseteq &= 1 \text{ using.WebInterface} \sqcap 1 \text{ effCause.Auditor} \quad (8.3)
\end{align*}

Table 8.7 shows the initial organizational assets declared in the organizational ontology. Assets indicated as located in the organizational ontology were declared as individuals, meanwhile that Authors, Auditors and Publications were mapped to the repository SIIP_PubsDB, already defined as an individual of type Repository. SIIP_D2RMMapping contains a reference to the XML mapping file used by the D2R Server for retrieving information from the repository.

Finally, the AuditNewPub SCN was stored in the Process Repository. In the probabilistic distribution it was indicated that the confidence on correction rules is near to zero, i.e. \( P(Z_{16} = A | Z_{13}, X_4) = 0.001 \), making the probability of unnecessary auditor revision near to zero as well, since \( P(Z_{20} | Z_{16} \neq A) = 0 \). Likewise, the probability of detecting an inconsistency applying a new inconsistency rule is initialized to 50%, i.e. \( P(Z_7 = A | Z_3 = \text{rule1}) = 0.5 \).
8.3.4 AIA Organizational Protocols

In our case study we only modeled an organizational protocol. This protocol is used for requesting the participation of agents and is called AgRequest. We assumed that resources are fully controlled by some agent and that information and knowledge is known a priori in each agent.

The AgRequest protocol causal diagram is shown in Figure 8.14. It is introduced a role DispatcherAg on charge of keeping track of agents in the system and notify which of them are available. Besides, the DispatcherAg is endowed with the capacity of instantiating agents based on the organizational role they can play, indicated by the variable ?role. In the SCN shown in Figure 8.14, the requester agent (?req) specifies the role that should play the required agent and a required attribute.

The PMA role in this protocol is played by the dispatcher agent, which we represent using the variable name ?PMA. A PMO is not required in this process. The communication protocol for this organizational process is shown in Figure 8.15.

8.4 Implementing AIA through Electronic Institutions

The process AuditNewPub was specified and implemented using the Electronic Institution (EI) formalism to demonstrate the compatibility of the CMAS methodology with current robust approaches like this [19]. AuditNewPub’s ontology, agent classes and pro-
8.4. IMPLEMENTING AIA THROUGH ELECTRONIC INSTITUTIONS

Figure 8.14: Causal diagrams for $AgRequest$.

Figure 8.15: Communication protocol for $AgRequest$. 
tocols were used for generating the EI specification. Decision making of agents implementing agent types PubCarrier and Corrector was simulated. Finally, organizational protocols were implemented through an outer performative structure.

The EI was specified and run using the software platform EIDE [5].

8.4.1 Expressing the AuditNewPub process in EIs

In the EI dialogical framework, organizational roles (Auditor and Author) were defined as external roles, whereas organizational agent classes (RepGuardianAg, PubCarrierAg, LogMonitorAg, AuditorAg and CorrectorAg) were defined as internal roles. In our implementation all the messages were codified through ACL INFORM messages.

In the EI ontology, essential forms were defined as datatypes whereas constants used in essential form definitions were declared as enumerations. SCN annotations used in the definition of messages (see Table 8.5) were declared as functions.

The AuditNewPub SCN (Figure 8.5) and its communication protocol (Figure 8.12) were used for constructing the AuditingPS performative structure, shown in Figure 8.16. In the SCN, actions along with its respective causes and effects were used for identifying d-separated subgraphs that would represent scenes. In this way we identified scenes NewPub (monitor), Auditing (audit) and Correction (correct, supervise, revise and undo). The Correction scene requires by the participation of the same CorrectorAg in actions correct and undo.

Additionally, the scene Audited is used by the PubCarrierAg for reporting the results of auditing to the RepGuardianAg, and the scene AudWaiting is introduced for direct-
8.4. IMPLEMENTING AIA THROUGH ELECTRONIC INSTITUTIONS

The AuditNewPub communication protocol is used for determining which agents participate in each scene. For instance, in the AuditNewPub scene participate RepGuardianAg, LogMonitorAg and PubCarrierAg agents, whereas in the Correction scene participate PubCarrierAg, CorrectorAg, AuditorAg and AuthorAg.

Multiple action executions produced by ALL quantifiers are represented through cyclic movements between scenes. For instance, a PubCarrierAg moves from an Auditing scene to a Correction scene and then it can create a new Auditing scene or it can conclude with the process moving to the Audited scene.

EI scenes are modeled using the AuditNewPub protocol. Messages illustrated in the sequence diagram are used as valid transitions/ilocutions between scene states. RepGuardianAg and LogMonitorAg are enabler agents of AuditNewPub, hence they must remain in the institution until the termination of the process. This constraint provoked that NewPub and Audited scenes were modeled with the continued presence of both agents. See for instance the NewPub scene (Figure 8.17), where both agents remain in the scene meanwhile PubCarrierAg agents enter at state end and exits at states commited or w4PC.

In the case of the Correction scene, the participation of CorrectorAg, Auditor and Author agents is optional. This is represented in the corresponding scene states through labels indicating their entrance and exit of the scene. Intermediate results of the process are informed to the PubCarrierAg by other participants, as represented by illocutions labeled Transf2Exp and Transf2Aut.

Another characteristic of EIIs that was exploited was the use of timeout transitions. In the same Correction scene, a timeout is used for limiting the time that the PubCarrierAg should wait for human assessment. Once that time limit is exceeded, the scene finishes and the PubCarrierAg determines which agent unfulfilled its organizational commitment neglecting the supervision or revision request.

EI norms used for validating movements and illocutions in performative structures and scenes, respectively, were defined using the sequence diagram of the AuditNewPub pro-
tocol. Sequence diagram expressions (guards) conditioning alternative and optional fragments are declared as constraining norms. Illocutions exchanged by participant agents set the institutional variables used in these norms.

8.4.2 Implementing Organizational Protocols

The organizational protocol AgRequest was adapted to Els specifications. This and other organizational protocols were specified through scenes or protocols collocated with AuditingPS in a main performative structure. In this way, the access of agents to organizational services is granted.

The agent request protocol for Els

Active EI scenes delimit the context of a process execution. Whereas the PMA and the PMO agents are responsible for the creation of these scenes, the rest of the participants must be invited to them in order to play a role in the process. In this way, the design of the communication protocol considered, additionally, the scene on which the request occurs and the number of agents requested (ONE, ALL).

The number of requested agents is given by those action quantifiers included in the process/plan SCN. If the process or plan establishes that action $X_i$ must be performed by every (ALL) agent Type(?ag), the request is issued using the quantifier ALL. Otherwise,
the quantifier $ONE$ is used.

The communication protocol of $AgRequest$ is illustrated in Figure 8.19. Table 8.8 shows the equivalences between SCN annotations and ACL messages. The EI role $Requester$ is introduced in the protocol. The datatype $Properties$ encapsulates a set of property-value tuples.

Additionally a $Dispatcher$ agent implements three additional protocols for accomplishing its purpose: 1) keeps track of agents entering into the institution (LogIn), 2) keeps track of agents leaving the institution (LogOut), and 3) addresses agents to the corresponding scenes (Invitation). More details on the implementation of these protocols are given in [20, 21].

### The organizational protocols layer

In order to access organizational services, participant agents must coincide in some scene. For this reason, scenes produced by organizational protocols are allocated in
a main performative structure, named \textit{MainPS}. Performative structures produced by organizational processes (e.g. \textit{AuditingPS}) are allocated in MainPS as well. Figure 8.20 shows the \textit{MainPS} of our implementation, which allocates \textit{AuditingPS} (\textit{Auditing}), and the scenes required for implementing \textit{AgRequest} described above.

### 8.4.3 AIA agents’ implementation in EIs

The agent class \textit{RepGuardianImplAg} extends organizational agent types \textit{RepGuardianAg}, \textit{Requester} and \textit{Dispatcher}, and establishes the repository to manage through the constraint \textit{controls} = \textit{SIIP-PubsDB}. The agent class \textit{PubCarrierImplAg} extends \textit{PubCarrierAg} and \textit{Requester}, and states that it is capable of auditing a single publication at once, represented by the constraint \(< 1 \\text{knows.Publication}.

The agent class \textit{AuditorImplAg} extends \textit{AuditorAg} and \textit{Requester}. This kind of agent receives an \textit{InconsistencyRule} as initialization parameter and its action \textit{audit} is simulated. Likewise, \textit{CorrectorImplAg} extends \textit{CorrectorAg} and \textit{Requester}, and its action \textit{correct} is simulated as well. Nevertheless, the efficiency of the correction for each correction rule is controlled by a parameter in the simulation: \(P_{\text{CRO1}}\) and \(P_{\text{CRO2}}\).

\textit{LogMonitorImplAg} extends \textit{LogMonitorAg} and \textit{Requester}, and establishes a monitoring frequency given by the constraint \textit{monitorsEveryNumSec} = 60, i.e. checks the SIIP_DB log every minute. This agent class also implements a version of the action \textit{monitor} on which a single new publication is detected on every checking.

Finally, the implemented \textit{UserImplAg} extends organizational agent types \textit{UserAgent}, \textit{Requester}, \textit{Auditor} and \textit{Author}, indicating that this agent class can act on behalf of users playing the role of expert auditor or publication author. Actions \textit{revise} and \textit{supervise} are implemented through functions that simulate the response of a real user. The response time for auditors is generated randomly in the range \(T_{\text{Aud}} = [5, 10]\), in seconds, whereas author response is given by \(T_{\text{Aud}} = [7, 15]\).
8.5. IMPLEMENTING THE AIA THROUGH CAUSAL AGENTS

8.4.4 Simulating AIA through EIs

The ABuilder tool generated the agent templates for modeled organizational agent types. Decision making and further details of implementation were programmed manually in the obtained Java classes. Decisions making was simulated in order to test all the possible cases in the modeled scenario.

Simulations of the EI through Ameli allowed to detect some elements missing in our process specification. For instance, if there is no available InconsistencyRule, the PubCarrierAg should go directly to the end of the process ($Z_4 \rightarrow Z_{19}$). Besides, on the unavailability of CorrectionRules for a given inconsistency the CorrectorAg should produce a correction request to the human Auditor ($Z_7 \rightarrow X_4$).

8.5 Implementing the AIA through Causal Agents

Agent roles defined in AuditNewPub and AgRequest were extended in the Agent Repository as specified in the last phase of the CMAS methodology. New agent classes were complemented with additional constraints for avoiding multiple interpretations and simplifying the instantiation of agents. For instance, the role AuditorAg was extended by two classes:

- DuplicityAudAg $\subseteq$ AuditorAg $\cap$ knows = IR01.DuplicatedRecord
- StatusAudAg $\subseteq$ AuditorAg $\cap$ knows = IR02.YearStatusInconsistent

overriding the constraint $\geq 1$ knows.InconsistencyRule in AuditorAg.

I defined a set of agents similar to those generated for the EI implementation (see section 8.4.1). Unlike EI agent implementation, with Causal Agents I just customized plans produced as partial views, complemented them with rules and implemented actions through actuator classes.

I also modified the goals associated to process views, in order to control plan instantiation. For instance, given a goal $G = (A, \Omega)$ such that $satisfies(G, P)$, setting $A = \bot$ causes the plan $P$ be only instantiated through subgoaling.

Additionally to process views of AuditNewPub and AgRequest inherited by agent classes it was necessary to design other essential plans. For instance, the PubCarrierAg needed a plan for qualifying the confidence on an audited publication. This plan is triggered by a predicate asserted when certain messages are received by the PubCarrier, for instance, $Z_{17} = \{A, B, C, D\}$.

The RepGuardianAgImp agent, on charge of managing the process AuditNewPub, sets the conditions for the process by instantiating a LogMonitor agent when a new publication is detected and provides the service of agent dispatching through the organizational protocol AgRequest. The RepGuardian agent also keeps track of every case observed by PubCarrier agents.
The subgoaling mechanism was proved successfully in the instantiation of agents. For instance, the condition \( Z_1 \) in the \( \text{AuditNewPub} \) process view controlled by the \( \text{RepGuardianAg} \) motivated the instantiation of a \( \text{LogMonitorAg} \). Knowing \( Z_2 \) (the precondition of the process view), and in order to enable \( X_1 \), the \( \text{RepGuardianAg} \) considered \( Z_1 \) a missing condition. Being \( \text{Ann}_{\text{AuditNewPub}}(Z_1 = T) \subseteq \text{Ann}_{\text{AgRequest}}(Z_5 = T) \) and \( Z_5 = T \in F_{\text{AgRequest}} \), subgoaling was initiated. Similarly was instantiated the \( \text{PubCarrierAg} \) when a new publication was reported by the \( \text{LogMonitorAg} \).

As long as the \( \text{RepGuardianAg} \) played both roles in the \( \text{AgRequest} \) protocol, it decided to play the role requester first and then the role dispatcher. This decision was based on a simple formula for ranking available plans (process views) on which there are privileged plans containing fewer actions controlled by other agents. The formula is \( xs - dxs \), where \( xs \) is the number of actions in the plan and \( dxs \) is the number of actions directly controlled by the agent. It is selected the plan with the lowest value.

In the \( \text{AgRequest} \) process, the generated protocol was modified due to the limitations in the policy followed with this purpose. For instance, the \( \text{PMA} \) role was already defined in the process and it was not introduced a PMO. Besides, the result of the process, \( Z_5 \), was directly observed by the \( \text{PMA} \) but it was no way of stating the it should be known by the requester too. For this reason the messages were codified manually; even so they were grounded on realizations of \( \text{AgRequest} \) variables.

The \( \text{PubCarrierAg} \) was capable of deciding not to execute \( X_5, X_4, X_3 \) and \( X_6 \) whenever the publication was consistent (\( Z_7 = B \)). Trials with this characteristic didn’t requested the participation of any \( \text{CorrectorAg} \), \( \text{Author} \) or \( \text{Auditor} \) for ?ruleI. In the other hand, when \( Z_7 = A \) the \( \text{PubCarrierAg} \) requested the participation of the \( \text{CorrectorAg} \) (\( X_3 = \text{True} \)) and then decided between requesting expert assessment (\( X_4 = \text{True} \)) or approving the correction made by the \( \text{CorrectorAg} \) (\( X_4 = \text{False} \)). Through a rule triggered by the omission of \( X_4 \), i.e. choosing \( X_4 = \text{False} \), was asserted the predicate \( \text{approvedBy}(?\text{cor}, ?\text{PMO}) \), which produced \( Z_16 = D \) and lead the trial to its conclusion.

After choosing \( X_4 = \text{True} \) the decisions made by the \( \text{PubCarrierAg} \) are driven by the decisions of the human \( \text{Auditor} \). If the \( \text{Auditor} \) approves or rejects the correction (\( Z_16 = \{A, B\} \)), actions \( X_5 \) and \( X_6 \) are omitted by the \( \text{PubCarrierAg} \), as indicated in the probabilistic distribution given by the expert. If the \( \text{Auditor} \) request undoing the correction (\( Z_16 = C \)), the action \( X_6 \) (undo) is performed by the \( \text{CorrectorAg} \). And if the \( \text{Auditor} \) requests the revision of the corresponding author (\( Z_16 = E \)) or ignores the request (\( Z_16 = G \)), the action \( X_5 \) is performed. In conclusion, the unique decision controlled by the \( \text{PubCarrierAg} \) that can change the selection of strategy in \( \text{AuditNewPub} \) is on \( X_4 \).

The \( \text{LogMonitor} \), the \( \text{AuditorAg} \), and the \( \text{CorrectorAg} \) have not choices, i.e. once they are informed of the contextual condition of the process view they control they must perform their action: \( X_1 = \text{True} \), \( X_2 = \text{True} \) and \( X_3 = \text{True} \), respectively. The \( \text{PubCarrierAg} \) controls the process and provides to these agents the information they require.

8.6 Experiments

I performed some experiments using the CMAS implemented with Causal Agents. Simulating the response of human participants and using parametric and structural Bayesian learning I tested the ability of Causal Agents for refining expert knowledge and for adapting to organizational and environmental changes.

Experiments setup includes an initial description of the process given by an expert and a set of distinct participant behaviors. Observing the response of agents and the qualification of audited publications, the PMA will refine the AuditNewPub process. I'll perform three different experiments, each one with a different set of simulated behaviors and using the probabilistic model updated in the previous experiment.

In the first experiment I will simulate a behavior of the human participants extremely different to the one assumed by the designer of the auditing process. In this scenario parametric learning is considered enough for adapting the response of the PubCarrierAg to these behaviors, considered environmental conditions.

In the second experiment, the organizational metric of minimizing unnecessary human intervention is introduced. I'll observe the response of the agents with different importance factors for the new metric.

Finally, in the third experiment, I will use structural learning for refining the probabilistic model. I will simulate a different response by human participants on each type of correction rule. I expect to observe a different treatment for them, i.e. a different strategy selection by the PubCarrierAg.

8.6.1 Set up

I start with a description of the AuditNewPub process given by the expert. Such description includes the expected behavior of participants and their interventions (actions), encoded in the probabilistic distribution of the process.

Participants' action efficacy is not altered, sensors and actuators are not considered noisy. The decision of the human participants is simulated using a random numbers generator and assigning a probability to each possible response they can give. On each experiment, it is given a table with the behavior of the Auditor and the Author.

I only consider two inconsistency rules with a single corresponding correction rule for one of them. The detection of inconsistencies is also simulated. The probability of detecting an inconsistency of type IR-01 is 55%, meanwhile it is 45% for inconsistencies of type IR-02. Each inconsistency is successfully corrected by the CorrectorAg producing a single Correction. In this way, the PubCarrierAg generates only two trials for each publication and controls their execution.

When the execution of the plan finishes for all the generated trials, the PubCarrierAg evaluates them and assigns an accrued confidence to the publication. For each trial it
calculates a score and their average is used for setting the qualification. The score for the trial is:

- 0 - if the detected inconsistency is not corrected or if the correction exists and even when is requested the revision of the corresponding author, he/she ignores the request.
- 1 - if the correction is undone by the CorrectorAg due to a request of the human Auditor.
- 2 - if the publication is considered consistent w.r.t. the inconsistency rule, if exists a correction and this is approved automatically by the PubCarrierAg, or if the correction is approved or rejected by the human Auditor.
- 3 - if exists a correction and this is approved, rejected or undone by the corresponding author.

The average score \( \text{avg} \) determines the confidence on the publication auditing: \( 0 < \text{avg} \leq 1 \) gives a LOW confidence, \( 1 < \text{avg} \leq 2 \) gives a MEDIUM confidence, \( \text{avg} > 2 \) gives a HIGH confidence, and \( \text{avg} = 0 \) sets an INCONSISTENT status to the publication. The obtained average score is set on the publication, setting the variable \( Z_{19} \) on the two trials. This aggregation introduces certain noise in the result of the process as long as the result on one trial affects the result in the another.

Likewise it is calculated the value for \( Z_{20} \). Unnecessary human intervention (\( Z_{20} = True \)) is set in both trials if at least one of them contains \( Z_{16} = A \) or \( Z_{17} = A \).

On each round of simulation is simulated the auditing of 50 new publications with the participation of one RepGuardian, one LogMonitor, two CorrectorAg agents, two AuditorAg agents, one PubCarrier for each publication, a single expert auditor and a different corresponding author for each publication. The feeding rate is controlled to avoid conflicts of agents insufficiency. For results on this kind of conflicts please refer to [20].

The set of strategies the PubCarrierAg can follow in AuditNewPub is shown in Table 8.9. According to the expert, the most trustworthy decision when an inconsistency is detected (\( Z_7 = A \)), is requesting the supervision of the Auditor, which can derive on the strategies \( S_3, S_4 \) or \( S_5 \), according to the decision of the human Auditor.

### 8.6.2 Learning configuration

In order to obtain a rapid learning rate on parametric learning I used a combination of 500 instances generated from the current probabilistic distribution and 500 instances observed during simulation. I only considered instances (SCN closed cases) obtained by PubCarriers as long as this SCN contains most of the variables of the process. More details of the learning procedure are given in section 4.4.
Regarding structural learning, I used the Bayes Net classifier implemented in Weka with 5,000 instances generated from the current probabilistic distribution. Learning was configured with a simple estimator and used the IC* implementation of Weka as search algorithm. In order to select a score that provides a more accurate prediction I evaluated four local score functions: Bayesian Dirichlet (BD) [42] and its variation BDeu [13], Minimal Description Lenght (MDL) [51], and the Akaike Information Criterion (AIC) [1].

Figure 8.21 shows the results of the comparison. In the first place I considered valid arcs those arcs $V_i \rightarrow V_j$ on which $V_i \in Ar(V_j)$, reducing the number of arcs to less than the half in all the cases. Then I counted how many of these arcs were already present in the original structure; the difference were new arcs. Next I dismissed arcs connecting control variables obtaining a set of candidate arcs.

![Figure 8.21: Comparison of score functions in the AuditNewPub SCN.](image)

Between these candidates there were several arcs connecting a precondition of a control variable $X_i$ with the effect of $X_i$. Despite the fact these arcs can provide a more accurate prediction on the outcome of an action, I didn’t considered it relevant as long as actuators were considered not noisy. In other domains these nodes would be very helpful. Finally, I considered remaining candidate arcs as those that provide more predictive benefit as long as they connect nodes with longer causal chains between them, w.r.t. the original structure.
I didn’t found an important difference between their rate of correct classification (around 73.6%), so I privileged the gain in predictive power. This was calculated as the proportion of long arcs between candidate arcs. In this way I selected the Minimum Description Length (MDL) score. The list of arcs founded with the MDL score are shown in Table 8.10.

The arcs 8, 9 and 10 are the long arcs found. The arc 9 \((Z_3 \rightarrow Z_{19})\) connects the inconsistency rule with the target variable, which makes me think that it will improve decision making for the PubCarrier, as long this is capable of observing both variables in its process view. I added all the arcs, except by 8 and 10; even when they are long arcs, variables \(Z_w\) (the presence of a Corrector Ag) and \(Z%\) (the presence of an Auditor Ag) do not vary in most of the cases. Besides, this arcs would have increased even more the size of the CPT for \(Z_w\), which passed from 210 entries in the original model to 840 with the addition of the arc 9. In total I added only 8 arcs to the structure.

Finally, for calculating the probabilistic distribution of the new structure I ran parametric learning with the same sample used in structural learning.

### 8.6.3 Experiment 1. Self-configuration of process specification

In the first experiment I test if my approach allows refining the process specification given by the expert. Given that simulated behaviors substantially differs from those represented originally, the organization (represented by the decision of the PubCarrier Ag) must adapt to the actual scenario.

The simulated behavior of Auditors and Authors is shown in Figure 8.22. In this experiment, the expert auditor approves only a 50% of the corrections made by agents and authors ignore the requests 60% of the times. The process designer expected a higher approval of automatic corrections and authors to be more cooperative.

I ran four rounds of simulation, applying parametric learning between each one of them and observing the decisions made by the agents. Between each round the probabilistic
distribution was updated and the new parameters of the Bayesian networks were used for recalculating the strategies of the process and the agents. I will observe changes on these strategies and will compare the overall performance on each round. The overall goal achievement is calculated using the formula:

\[
Score(Z_{19}) = 10 \cdot O(Z_{19} = \text{High}) + 7 \cdot O(Z_{19} = \text{Mid}) + 3 \cdot O(Z_{19} = \text{Low})
\]

where \(O(Z_i = z_i)\) gives the number of closed cases having \(Z_i = z_i\). Factors of this formula correspond to the importance or preference expressed in the organizational metric \(\text{PubConfidence}\) (see Figure 8.2).

### 8.6.4 Experiment 2. Optimization on conflicting goals

Next I tested the ability of internal agents for adapting to organizational changes, represented by the introduction of the metric \(\text{MinHumanIntervention}\). This time, the simulated behavior of human participants is similar to the expected by the process designer. I tried different configurations of preferences on organizational metrics and observed the behavior of agents on each one of them.

The simulated behavior of Auditors and Authors in this experiment is shown in Figure 8.23. In this case, the behavior of both Auditors and Authors is different for each correction rule. CR01 has a better acceptance on human revision, meanwhile CR02 still has a low rate of acceptance. Furthermore, CR02 is mainly rejected by corresponding authors.

The configurations on organizational metrics considered in this experiment are shown in Table 8.11. In the baseline configuration the preference on \(Z_{20}\) is neutral, whereas in the rest of them it has a negative preference indicating that is a non-desired effect. Being specified with a negative preference, the new metric is in conflict with the original one, associated to \(Z_{19}\) with positive preferences, as demonstrated by the correlation \(P(Z_{20}, Z_{19}) > 0\).

In the first configuration the penalty for unnecessary human intervention is high, meanwhile in the second is lower. For the third configuration I selected a balanced combination that let me observe the selection of multiple strategies.
CHAPTER 8. A CASE STUDY ON AIA

Table 8.11: Configurations for experiment 2

<table>
<thead>
<tr>
<th>Metric</th>
<th>Baseline</th>
<th>Configuration 1</th>
<th>Configuration 2</th>
<th>Configuration 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I(Z_{19} = \text{High})$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$I(Z_{19} = \text{Medium})$</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>$I(Z_{19} = \text{Low})$</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>$I(Z_{20} = \text{True})$</td>
<td>0</td>
<td>-5</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

The first simulation round sets the baseline for comparing the performance with the three given configurations. All the agents incorporate the new organizational metric in their process views. The $OFFn()$ of participants is modified with the inclusion of the new metric. In consequence, I observed changes in the behavior of the agents due to a different evaluation of available strategies.

At the end of this simulation phase I selected the metric configuration that maximizes the confidence on the repository and minimizes unnecessary human intervention. The first objective is calculated by the aggregation $Score(Z_{19})$, described in the previous experiment. The second objective is calculated by the aggregation:

$$ Score(Z_{20}) = I(Z_{20} = \text{True}) \times O(Z_{20} = \text{True}) $$

where $I(Z_{20} = \text{True})$ is the importance factor assigned to the metric $\text{MinHumanIntervention}$ in the given configuration and $O(Z_{20} = \text{True})$ is the number of cases on which the publication auditing required unnecessary human intervention. The sum of both scores will give us the punctuation obtained in the respective configuration.

8.6.5 Experiment 3. Self-protection of inaccurate expert knowledge

In the third experiment I use structural learning for giving a differentiated treatment to each correction rule. Using a probabilistic distribution that reflects the actual behavior of human auditors and authors I applied structural learning. After adding new edges to the original graph I performed three rounds of simulation with parametric learning.
between each one of them. I expect agents learn to distinguish between correction rules and make different decisions on each case.

The simulated behavior of human auditors and author is similar to the one of the previous experiment. These are shown in Figure 8.24. This time CR01 is more trustworthy and CR02 is much more lousy.

<table>
<thead>
<tr>
<th>Auditor</th>
<th>CR01</th>
<th>CR02</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z16=A Approve</td>
<td>85%</td>
<td>20%</td>
</tr>
<tr>
<td>Z16=B Reject</td>
<td>0%</td>
<td>15%</td>
</tr>
<tr>
<td>Z16=C Undo</td>
<td>0%</td>
<td>10%</td>
</tr>
<tr>
<td>Z16=E Send to author</td>
<td>10%</td>
<td>50%</td>
</tr>
<tr>
<td>Z16=G Ignore</td>
<td>5%</td>
<td>5%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Author</th>
<th>CR01</th>
<th>CR02</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z17=A Approve</td>
<td>70%</td>
<td>5%</td>
</tr>
<tr>
<td>Z17=B Reject</td>
<td>15%</td>
<td>80%</td>
</tr>
<tr>
<td>Z17=C Undo</td>
<td>10%</td>
<td>5%</td>
</tr>
<tr>
<td>Z17=D Ignore</td>
<td>5%</td>
<td>10%</td>
</tr>
</tbody>
</table>

Figure 8.24: Simulated behaviors for (a) Auditors and (b) Authors in experiment 3.

After obtaining a baseline behavior with the original structure, I applied the structural learning algorithm described in section 4.4.2. With the new structure I performed three more rounds of simulation. I expect that unnecessary revisions made by expert auditors and by corresponding authors drops down on CR01 and increases on CR02.

8.7 Results

Next I present and analyze the results obtained in the previously introduced experiments.

8.7.1 Results on Self-configuration of process specification

Overall performance results of the first experiment are shown in Table 8.12. On it there are illustrated a summary of the chosen strategies and the overall performance obtained in the auditing of the 50 publications (100 trials).

This information is also shown in Figure 8.25. Strategies and results are shown in percentiles in these charts. As can be seen agents stop requesting human participation as soon as they notice that they do not provide good feedback or even ignore the requests. After the first parametric learning phase, between the training and the first round, the updated strategies produced an odd phenomenon: the PubCarrier and the user agent representing the Auditor differed on their decision about executing $X_4$. According to the PubCarrier, $X_4$ should be executed whenever an inconsistency is detected. Nevertheless the agent representing the Auditor decided not executing $X_4$ as long as utility of $X_4 =$
Table 8.12: Performance evolution on self-configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Baseline</th>
<th>Round 1</th>
<th>Round 2</th>
<th>Round 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(S_{\text{None}})$</td>
<td>47</td>
<td>45</td>
<td>54</td>
<td>48</td>
</tr>
<tr>
<td>$O(S_{\text{Auto}})$</td>
<td>0</td>
<td>0</td>
<td>46</td>
<td>52</td>
</tr>
<tr>
<td>$O(S_{\text{Auditor}})$</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$O(S_{\text{Author}})$</td>
<td>20</td>
<td>55</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$O(S_{\text{Undo}})$</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$O(Z_{19} = \text{False})$</td>
<td>2</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$O(Z_{19} = \text{Low})$</td>
<td>26</td>
<td>34</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$O(Z_{19} = \text{Medium})$</td>
<td>68</td>
<td>44</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>$O(Z_{19} = \text{High})$</td>
<td>4</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Score($Z_{19}$)</td>
<td>594</td>
<td>510</td>
<td>700</td>
<td>700</td>
</tr>
</tbody>
</table>

Figure 8.25: Strategy selection and overall performance in experiment 1.

*False* was higher than the utility of $X_4 = \text{True}$. Figures 8.26 and 8.27 show the calculation made by the *PubCarrier* and the *Auditor*, respectively.

Both actions $X_4 = \text{True}$ and $X_4 = \text{False}$ are considered in valid strategies of both agents. Nevertheless, the process view used by the user agent contains less variables than the process view of the *PubCarrier*. Besides, the *PubCarrier* has more information that improves the accuracy of its decision. For the user agent, in the other hand, the uncertainty introduced by the absence of such information produces a wrong decision, or at least one different to that made by the *PubCarrier*.

When the user agent decides omitting $X_4$, a rule is triggered asserting the predicate *ignoredBy(?cor, ?aud)* that sets $Z_{16} = E$ in the trial. In consequence, as indicated by the designer of the process, the corresponding author revision is requested. This rule is an implementation decision out of the control of the process designer; this, as well as programming bugs are examples of behaviors not foreseen during the design phase.

Anyway, the system corrects the problem in round 2, after parametric learning, when the *PubCarrier* decides that the safest choice is no longer $X_4 = \text{True}$, given that the author didn’t responded to the requests. If the author would answered the request instead of
8.7. RESULTS

Figure 8.26: Decision making for the PubCarrierAg.

| x4     | A priori | A posteriori, i.e. \( P(z_{19}|do(x_4),s_2) \) |
|--------|----------|-----------------------------------------------|
|        | \( P(x_4|s) \) | \( Z_{19}=\text{High} \) | \( Z_{19}=\text{Medium} \) | \( Z_{19}=\text{Low} \) | \( FFn(x_4) \) |
| X4=True| 0.7353   | 0.1012 | 0.4906 | 0.2274 | 5.1280 |
| X4=False| 0.2647  | 0.1786 | 0.3571 | 0.2500 | 5.0357 |

* \( s = \text{AuditNewPub}_Z_{12} = \text{True}, \text{AuditNewPub}_Z_{15} = \text{True}, \text{AuditNewPub}_Z_{27} = A, \text{AuditNewPub}_Z_{10} = \text{True}, \text{AuditNewPub}_Z_{13} = \text{True}, \text{AuditNewPub}_Z_{11} = \text{True} \)

Figure 8.27: Decision making for the Auditor.

ignoring them, the safest choice for the PubCarrier would remain \( X_4 = \text{True} \).

On the other hand, the implemented protocol didn’t allowed to recognize the differed decisions made by the PubCarrier and the user agent. Whereas the PubCarrier registered \( X_4 = \text{True} \), the user agent registered \( X_4 = \text{False} \). Besides, learning was made using the information compiled by the PubCarrier only. In consequence, parametric learning taught the PubCarrier that the outcome of doing \( X_4 = \text{True} \) was \( Z_{16} = E \) most of the times.

Finally, round 3 didn’t show any change on the strategies selected by the PubCarrier. In this way I can conclude that the system reconfigured itself and reached a stable operation in the third round.

8.7.2 Results for optimization on conflicting goals

Table 8.13 shows the results of tree configurations of importance factors for organizational metrics, denoted \( I(Z_i = z_i) \). This table also shows the times a strategy \( S_i \) was chosen. Scores on \( Z_{19} \) and \( Z_{20} \) are summed up for selecting the best configuration. The performance obtained in the baseline configuration is 718. This score is result of the new behaviors codified for this experiment. In the baseline configuration we can observe that all the available strategies are selected, except for the Auto strategy.

With the first configuration, given that the penalty for unnecessary intervention is high
(−5), the safest strategy is Auto, i.e. perform the auditing without human intervention. Agents confidence on correction rules is high enough. In the second configuration the PubCarrier requested human intervention all the time, since the introduced penalty was not so high (−2). Nevertheless, once again I observed the same phenomenon of the first experiment where the user agent representing the Auditor decided not performing \(X_4\) and redirected all the requests to the respective author.

Meanwhile in the first configuration unnecessary human intervention was not present, in the second configuration it was observed in 34% of the observed instances. This provoked a total score in the second configuration lower than the one obtained in the baseline and the first configuration.

The third configuration was set for overriding the conflict between PubCarriers and Auditors decisions. With a penalty equal to -1 both agents decided to perform \(X_4\). In this way we could observe the selection of multiple strategies again. The balance between requesting or not human intervention produced a higher total score this time (750).

Even when the total score obtained is calculated in terms of the same metrics used for guiding agent decisions, the administrator can determine which behavior is more effective in its domain. For instance, in the third configuration the expert auditor supervised each correction made by the agents, but only in 20% of them the author made a second revision. In that sense, charts shown in Figure 8.28 provide a visual aid for the administrator.

In conclusion, the different configurations allowed to observed different behaviors of the system and provided valuable information for deciding which configuration to use in the system. Conflicting goals, improving confidence in the auditing process and minimizing unnecessary human intervention, can be mediated through the inclusion of organizational metrics. I showed how agents’ behavior is affected by the inclusion of the new metric.
8.7. RESULTS

Table 8.14 shows the strategies used in the baseline configuration and in the three rounds that followed structural learning (see section 8.6.2). Strategies are divided according to the correction rule used in each case. A graphical representation is shown in Figure 8.29.

Once again a disagreement was observed between the PubCarrier and the Auditor regarding $X_4$ in the second round of simulation. That explains why in the second round prevails the strategy Author. Even so, in that round the strategy Auto is used for CR02 but not for CR01, indicating that there was a different treatment for each rule.

Finally in the third round the strategy Auto is selected for both rules as a consequence of the excessive enquire to authors. In one side, authors mainly approved the corrections, producing unnecessary intervention. In the other, even when authors mainly rejected corrections made with CR02, the accrued calculation of publication confidence didn’t allow to distinguish the efficacy of this strategy.
Taking a closer look to the decisions made by agents we can observe that the distinction between both correction rules is actually made by the PubCarrier. Table 8.15 shows the finality functions calculated by the PubCarrier for CR01 and CR02.

Table 8.15 shows two scenarios, one on which the corresponding author of the publication is not present and another on which he is. The second scenario occurs when the first correction was revised by the author and there is a second correction in the same publication. Given that all the trials associated to this publication are updated, the evaluation of $FFn(X_4)$ considers this fact.

We can observe that in the absence of the author, the PubCarrier decides requesting the supervision of the auditor for CR01 but not for CR02. And when the PubCarrier notice that the author is already present it changes its decision for CR02, requesting auditor’s supervision, expecting the author being asked to participate as well.

In this experiment I conclude that the system actually learned to distinguish between the two correction rules but the way on which $Z_{19}$ and $Z_{20}$ are calculated introduces excessive noise and forces the system to adopt the same strategy in both cases (Auto).

### Discussion on results

The system configured itself with the response of auditors and authors to the rules codified by the expert. As shown in the first experiment, parametric learning permitted adapting the specification given by the expert to actual environmental conditions, rep-
represented by human participant behaviors. The finality function \((FFn)\), allowed agents to mediate between the causal effect of their actions on the goal and the preferences expressed by the administrator through organizational metrics.

In the second experiment, optimization with conflicting goals was possible thanks to the use of preferences with negative values. Agents modified their decisions with the introduction of a new metric in order to maximize their contribution to the final causes of the system. Simulation of different configurations allows the administrator to measure the utilization of resources (e.g. human supervision) and choose the best set of preferences for organizational metrics.

Structural learning allowed distinguishing between the response of authors and auditors to the given correction rules. The disagreement between the \textit{PubCarrier} and the \textit{Auditor} prevented the \textit{PubCarrier} of adopting the safest strategy for the inaccurate rule, but there was an attempt in that sense. Besides, it was possible to observe the different decisions made by the \textit{PubCarrier} with different conditions (e.g. the presence or absence of the corresponding author).

The system showed its capability for self-recovering from unexpected behaviors produced by implementation decisions. Even when the response of the user agent representing the Auditor was not foreseen, the system was capable of selecting the strategy that better satisfied the preferences given by the administrator.

Finally, experiments evidenced differed decisions between agents that were not captured correctly due to the implemented protocol. Besides, given that agents can only choose between the initial strategies, they are not capable of discovering new strategies; they are limited to select the best one on each case. Nevertheless, this is a desirable behavior in a controlled organizational process.

8.8 CMAS applications and limitations

In this section I propose further applications of the CMAS framework in other application domains. I also elaborate on scalability issues and on the limitations of the current implementation.

8.8.1 CMAS in other domains

The CMAS framework can be applied to other application domains where:

- **Processes are constituted by finite sequences of actions.** In contrast, robot navigation, usually modeled through Markovian Decision Problems (MDPs), is an example of applications where the goal is reached after an indefinite number of actions.
• **There are multiple ways of doing things.** If the plan is only a very long sequence of actions, and in consequence contains a single strategy, there is no much gain in using these formalisms.

• **Creativity is not allowed.** The possible strategies are given by the designer and agents must stick to one of them. Nevertheless, agents are free to decide which strategy is better in every scenario.

I recommend using the CMAS framework for analyzing and structuring processes and then determine if the solution can be implemented through agents. This framework, as well as Electronic Institutions, consider the participation of human users in the process, which allows reinserting the human in the loop, as it was recently proposed by Human Centered Robotics.

Specialized tasks like scheduling and constraint satisfaction problems (CSPs) must be treated with most suitable formalisms, like Genetic Algorithms or model checkers. Nevertheless, such tasks can be incorporated in this framework through an actuator containing the implementation of such algorithms. The solution to these problems can be represented in triplets and then be used by the agent somehow. Classes and properties used for defining the problem and the solution would serve as integration interface.

I can also recommend applying the CMAS approach to domains on which Case-Based Reasoning (CBR) has being successfully applied. Additionally to the capacity for selecting strategies for each case, Bayesian learning allows incorporating new data to the model and improve the prediction of the model. Besides, the agent architecture modularizes and inserts this type of inference in an event-driven workflow.

### 8.8.2 Scalability

**Given the agent-based nature of this approach, I consider it scalable to large scenarios.** I support this claim in: 1) the handling of large amounts of information through SCN Cases, and 2) the hierarchical decomposition of goals and creation of agents.

SCNs used for specifying organizational processes clearly delimit the information to use. Agents create as many SCN cases as the process specification requires, indicated through quantifiers. Through auxiliary protocols, agents obtain information from the repositories following a given workflow. In this way, the access to information is mediated through these protocols and the competencies of agents delimit how much information they can handle.

Every agent is aware of its creator and the agents it created, as well as the goals other agents delegate him. Additionally, the MAS does not start with a fixed number of agents, but they are spawned only if they are invoked. In the opposite way, agents without active goals should leave the system, freeing resources. Anyway, it can be
constrained the maximum number of agents and goals an agent can handle in order to avoid bottlenecks.

In this chapter it was described a Dispatcher agent charge of creating and referring agents. In this case, the solution can become distributed if the Dispatcher agent has a maximum limit of created agents and it is allowed to create other Dispatcher agents in turn.

An agent and the group of agents created by him can reside in different servers. Nevertheless, it would be necessary to dictate a policy (organizational metric) for motivating the distributed creation or mobility of agents.

Nevertheless, as in any other MAS frameworks or middleware, the risk of a crash is still present. In this sense, redundancy can be implemented through clusters of agents providing the same service. For instance, instead of having a unique agent managing all the printers in a company it would exist an agent per each printer. In this way, if an agent needs to send a work to a color printer it might have many available options (the subset of color printer agents) from which it will choose one based on previous experience.

Finally, this approach proposes that a single agent is the pursuer of the main goal of the system. A crash in this agent constitutes the biggest risk for my approach. For this reason I separated the function of managing and monitoring processes in two roles: Process Manager (PMA) and Process Monitor (PMO). The latter is who has the highest load represented by the selection of strategies and delegation of tasks, meanwhile the former only sets the conditions for the process and compiles the results. The former agent plays the PMA role of the system process and controls the lesser as possible actions. As well, the former agent should create as fewer agents as redundancy permit. These constraints can be expressed in the agent class of this agent.

### 8.8.3 Limitations of the current CMAS implementation

The main limitations of the current CMAS implementation are the following:

- **The poor expressiveness of conjunctive queries (CQs).** CQs do not allow using negation or comparison operators hence modeling goals and plans requires using the same tricks that First Order Logic (FOL) systems. For instance, I used the property \textit{status} for indicating on which stage of the process a publication or correction was.

- **The limited set of quantifiers.** Actual SCN quantifiers (ONE and ALL) allow indicating that only one or that all the possible actions must be performed in the process. Nevertheless it cannot be indicated that the process requires exactly, at least or at most N executions of a given action.

- **Temporal considerations cannot be expressed in the plan.** Subsumption of temporal predicates requires further inference that is not expressed in actual DL
constructors. For instance, it cannot be expressed \textit{minutesAfter}(N, T), where N is the number of minutes and T represents a point in time.

- **The inability for trying again dropped strategies.** Once that an agent chooses a strategy for a given scenario, it will not choose a different strategy in future cases. This prevents verifying if a dropped strategy is valid again.

I present some alternatives to overcome these limitations in section 9.9.

### 8.9 Summary

I presented a case study in which the task of information auditing is automated through the design and implementation of a CMAS. I described the auditing process originally performed by expert auditors and then propose an automation solution. Next, the CMAS methodology was applied for modeling the auditing of new publications in the repository. In order to compare our approach, it is documented an implementation through Electronic Institutions and another through a CMAS conformed by Causal Agents. I conclude documenting some experiments and their results.

I focused in the auditing of a publication's repository of professors at a university. Empirically, expert auditors have developed rules for verifying the internal and overall consistency of information. The main objective of automating this process is the automatic detection and correction of inconsistencies with the supervision of expert auditors and authors.

I proposed a solution called Autonomic Information Auditing (AIA) given that is expressed in terms of autonomic goals: 1) self-configuring process specification, 2) minimizing unnecessary participation of human users meanwhile the confidence in the repository consistency is maximized (self-optimizing), 3) self-healing of inaccurate inconsistency and correction rules, and 4) self-protection of uncommitted users.

The process passed through the first four phases of the CMAS methodology, producing an organizational ontology, an automated process specification, process views for each organizational role, a communication protocol and an auxiliary protocol for requesting agents.

Then, this process specification was used for implementing a multiagent system using the EIDE platform for Electronic Institutions [5]. The organizational ontology was used for defining the EI dialogical framework and its ontology. The automated process specification was decomposed in scenes, constituting the EI performative structure. The communication protocol was used for specifying scenes in terms of valid illocutions. Likewise, the auxiliary protocol was adapted to EIs. Finally the ABuilder tool was used for generating reactive agents that allowed simulating the process.

Agents implemented in the fifth phase of the CMAS methodology demonstrated the capabilities of the Causal Agent architecture for guiding plan execution through SCNs,
for enabling plans through subgoaling and for instantiating new agents using a DL agent class. In the prototype I used Java functions for emulating DL reasoning during agent profiling and instantiation.

Finally I tested the autonomic capabilities of the AIA CMAS. First, I verified the self-configuration of the process specification through a training phase. Then it was simulated organizational changes through the introduction of a new organizational metric and observed changes on the strategies chosen by internal agents. Finally I tested CMAS adaptation to environmental changes, represented by human participation.

Experimental results demonstrated that the system reconfigured itself in response to external stimuli and reached a stable operation. The introduction of the new organizational metric affected the behavior of internal agents, which adopted different strategies on the different scenarios. Finally, structural learning allowed making a different treatment of both correction rules, based on the response of auditors and authors to their application. Agents were even capable of selecting a different strategy when actual conditions changed.

Besides experiments showed two agents (PubCarrier and Auditor) making different decisions for the same case. Even when the plan controlled by the Auditor is a subset of the PubCarrier’s plan, when both agents calculated the causal effect of their actions on the target variables $Z_{19}$ and $Z_{20}$, the different information possessed by them makes them differ in their decision.

Finally I proposed other application domains where the CMAS approach can be used. I also discussed actual limitations of my approach and elaborated on scalability issues.
Chapter 9

Conclusions

In this research I presented an approach for endowing large organizations with organizational intelligence through the automation of their processes. In order to do so, several techniques of Artificial Intelligence (AI) were integrated in a framework and a methodology was developed. The proposed framework borrows notions from classical philosophy in order to integrate modern AI approaches inspired in Causality and Metaphysics.

In order to endow organizational processes with intelligence I proposed the *Causal Artificial Intelligence Design (CAID)* theory which I used for defining *intelligent entities*. I extended this definition to *intelligent organizations* and *intelligent software agents*. Besides it allowed me identifying rational aspects of human users relevant in their interaction with a software system.

The CAID theory is grounded in both: 1) an ontological framework inspired in classical philosophy and formalized through Description Logics (DL) formalisms, and 2) a set of formalisms that integrate probabilistic, logic and semantic components for representing networks of causal relations (SBCMs and SCNs). These formalisms provide a novel formalization of organizational processes and plans, on which there are identified the material resources, information, knowledge, participants (agents), goals, actions, events and constraints involved in the process.

In order to incorporate the participation of intelligent agents in the development of organizational intelligence, I proposed a methodology called *Causal MultiAgent Systems (CMAS)*, used for modeling organizational processes through the introduced ontological framework and semantic-probabilistic formalisms. This methodology introduces two roles, the *Process Manager* (PMA) and the *Process Monitor* (PMO), played by software agents responsible for managing the process and guiding its execution.

I developed a tool that consumes this process specification and produces a description of the software agents and human profiles required for carrying out the process. This tool produces partial views of the process specification that represent plans for each participant, and provides a preliminary communication protocol for the process.

The definition of intelligent agent given according to the CAID theory is materialized
in the *Causal Agent* architecture. This architecture incorporates a goal-driven BDI inference engine that consumes the plans and protocols produced by the decomposition of the process, and uses an agent description for loading dynamically the actuators, rules and plans that will require. Besides, the explicit representation of agents in the process specification allows requesting the participation of an agent with certain characteristics in a process execution, or spawning a new one if none is available. This architecture is also used for encapsulating the participation of human users in the process.

Besides, given that the specification of processes and plans is built upon the Bayesian causal network formalism [71], we can use available learning algorithms for: 1) refining the specification given by the expert, and 2) adapting the process to environmental changes observed by the agents. The process specification is complemented with organizational policies that allow expressing preferences over events that occur as consequence of the process execution.

The incorporation of an organizational policy that punishes unnecessary human intervention, a schema of human supervision encoded in the process specification and the use of parametric learning, served for transferring knowledge from the expert to autonomous agents. In a controlled scenario, agents learned to carry out the process without the participation of human participants once they verified the effectiveness of rules given by experts and procedures programmed by the developer.

To demonstrate the applicability of my proposal, I used the CMAS methodology and the Causal Agent architecture for modeling and automating a process of information auditing. Furthermore, the process specification produced with my methodology was used for developing an implementation of the process through the use of the Electronic Institution formalism and tools, demonstrating the validity of my approach as MultiAgent Systems modeling tool. Experiments demonstrated the capabilities of the *Causal MAS* and causal agents for adapting to environmental and organizational changes.

Finally, I consider that this work has two main contributions. Due to its broad impact on the design of intelligent systems, I consider the presented causal theory of design as a significant theoretical contribution. On the other hand, the novelty and further applications of the Semantic Causal Network makes this formalism be an important contribution to AI formal tools. The relevance of both contributions is further explained in section 9.2 and section 9.4, respectively.

In the next section I answer the research questions of this work. Then I detail the main theoretical, formal and methodological contributions of this work and compare it with similar approaches. I conclude proposing some further extensions to this work.

### 9.1 Organizational Intelligence through Causality

In order to model organizations as intelligent entities I proposed an ontological framework based on Description Logics and a causal theory of Artificial Intelligence design, the
CAID theory. An intelligent organization is modeled semantically in terms of the goals it pursues, the processes it implements, its participants, its information and knowledge, its material resources and the metrics it uses for qualifying its results.

Organizational processes are specified as goal-oriented causal networks thanks to two semantic-probabilistic formalisms: the Semantic Bayesian Causal Network (SBCM) and the Semantic Causal Network (SCN). They use causal relations to represent causal dependencies between agents, knowledge, resources, goals, actions and events involved in the execution of the process.

The individual component of Organizational Intelligence is represented by Causal Agents, a probabilistic goal-driven Belief-Desire-Intention (BDI) agent architecture presented in Chapter 6. This architecture was used for introducing autonomous software agents and for representing the participation of human users in organizational processes.

These agents are capable of choosing the options that most contribute to their own goals and to those posed by the organization, mediating between them. To do so, agents consider a set of goal preferences given by the organization and the causal effect of its actions on these goals. Additionally, Bayesian learning algorithms are used for refining the probabilistic model underlying in the SCN formalism. In this way, agents improve the accuracy of the causal effect of its actions on its goals based on cases previously observed. This means that agents learn through experience.

Individual decisions made by participant agents constitute process strategies that can be used for carrying out the process. These decisions are limited by the process according to the specification given by the expert. Changes on environmental conditions, that include the response of human participants, are incorporated in the process representation through periodic Bayesian learning. These changes modify the estimation of the causal effect of agent actions and produces changes on their decisions. In consequence, agents and the organization itself adapt to environmental conditions.

On the other hand, the Causal MAS methodology describes how this framework can be used for automating organizational processes and transferring knowledge from human experts to autonomous software agents. Knowledge, represented in the ontological framework by URIs denoting procedures or rules, is applied by autonomous agents under the supervision of human experts. Additionally, an organizational metric punishing unnecessary human intervention is incorporated in the specification of processes.

Observation of human response teaches agents which pieces of knowledge can be applied confidently and which not. In this way agents choose which strategy to adopt on each case. Initially, agents prefer the safest strategy, i.e. that on which human supervision might be unnecessary but the achievement of the process goal is assured. Once that confidence on that rule is high enough, agents adopt a strategy with minimal or null human participation.

The Causal Agent and the Causal MAS architectures provide a scalable approach for large organizations. Scalability is founded on the hierarchical decomposition of goals and creation of agents. Additionally, large amounts of information are handled through
delimited process occurrences, i.e. SCN Cases. Nevertheless, the performance of the actual implementation of Causal Agents still need be improved. Additionally, in order to warrant a continuous operation of the system this approach requires a conflict resolution protocol and a general solution for deadlocks. A proposal for all these drawbacks is given in section 9.9.

In conclusion, the CMAS framework can be used for implementing organizational intelligence in large organizations. As discussed in the next section, Causality theory was used for making a coherent integration of AI techniques with this purpose.

9.2 Contributions of Aristotelian Causality

In this section I would like to point out the contributions that the use of Aristotelian Causality theory allowed me to do in MultiAgent Systems and to propose further applications. In the first place, I used Causality as a theoretical framework for modeling organizations, agents and human users, as well as their interactions. I proposed a formal representation of finality, in terms of goals and preferences, that is used instead of utility theory for making rational decisions. This allowed me to define intelligence, learning and adapting in causal terms and providing a computational implementation of them in my framework. Additionally, finality alignment is used for justifying the hierarchical decomposition of goals and motivating cooperation between agents.

The four main categories of causes (efficient, final, formal and material) proposed by Aristotle allowed me to identify the relevant elements in the design of agents. I listed the actions an intelligent agent performs (perceiving, acting, cooperating, deciding and learning) and proposed an agent definition in terms of the causes that intervene in such actions. Then I abstracted this definition for defining intelligent entities and specialized it later for intelligent organizations and human users. The result of this exercise was the CAID theory, constituted by a set of principles that guides the design and implementation of intelligent entities.

The final cause of an agent (intelligent entity) is not necessarily the same of its designer, but it is aligned with it. This means that the former is a mean for the latter. Similarly to Singh’s commitment obligations [84], the final cause of the agent is represented by those goals the agent must pursue by design. These goals are called essential goals and are defined in potency, indicating that the agent will try to achieve them on any chance it has. Additionally, it was expressed a preference on essential goals using a scale of goodness, i.e. how good or bad should an agent consider to be an actual goal derived from one of these essential goals. In my approach such preferences are dictated by the organization and participants cannot change them. The finality of an agent is expressed by its essential goals and their goodness.

Similarly, the final cause of an intelligent organization was defined through a set of organizational goals and organizational metrics. The achievement of these goals is specified through organizational processes which in turn are expressed through causal networks.
9.2. CONTRIBUTIONS OF ARISTOTELIAN CAUSALITY

I incorporated semantic annotations in the Bayesian Causal Networks approach for: 1) representing the types of causes (knowledge, agents and resources), 2) identifying action executors or enablers, and 3) representing the different ways of achieving the process goal.

Semantic annotations of SCNs are used for building the ontology of the system. These definitions are grounded in a given set of concepts (e.g. goals, agents, resources, knowledge, etc.) with a specific meaning in the framework. In this way, system's ontology definition is based on the purpose of the system.

The framework allows representing hard and soft goals. Process goals are considered hard goals as long as they motivate the execution of the plan and its achievement motivates agent action. Additionally, soft goals are represented by organizational metrics that identify events that are desired or undesired by the organization. In both cases, the administrator expresses a degree of preference for both types of goals.

Additionally, Pearl's causal effect [71] allowed me to quantify the likelihood of achieving a goal or causing a side effect through an agent intervention (action). This causal effect and the preference on the goal or side effect represent truth and goodness of the action. The sum of them constitute the contribution of the action to the finality of the agent, which it is formalized in my framework through the Finality Function (FFn), see Definition 79.

Finality is an alternative to the utility theory. In essence, the FFn has the same components that the expected utility: a weight (preference) and the probability of achieving a goal; see influence diagrams [45] and the Independence Choice Logic (ICL) [74] for instance. But the notion of finality is collective rather than individual and self-interested, thanks to the possibility of identifying (un)observable effects of my own actions in others' goals. Besides, even when agents are pursuing individual goals, these goals are aligned with the goals of the system. And lastly, the notion of cost used in utility theory can be represented causally as negative side effects.

Agents exist in the system due to their contribution to the achievement of some system goal. In my approach I motivated the creation of an agent with this principle, nevertheless I didn't deal with the disposal or destruction of an agent when it is not longer required. In this sense, once that an organizational goal is achieved, the pertinence of the agent should be revised. Translating this principle to open systems, it can be stated that an agent shouldn't be allowed to enter in the system if it doesn't contribute to some organizational goal; this can be validated through an agent class definition that exposes the capabilities of the agent.

Agent's rationality is based on its essential goals and the selection of the action that best contributes to the finality of the agent. In my framework, agents are aware of the actions and plans that control and they can choose the plan that best suits in a given scenario. All the possible strategies identified in a plan are evaluated simultaneously and the action with the highest FFn is chosen. In the evaluation of the FFn there are also considered organizational side effects, hence the decision of the agent mediates between
CHAPTER 9. CONCLUSIONS

individual and collective goals.

Learning can be expressed as the capacity of the agent for improving the accuracy of its predictions on the causal effect of its actions. I am assuming that preferences do not change along time. In this sense it was applied batch parametric and structural learning in the experiments described in section 8.6. Pearl’s IC* algorithm was used for doing structural learning. This algorithm makes use of causality assumptions for discovering dependencies between variables. Experimentally I observed that IC* discovered some of the relations actually stated in the process’ SCN. I used some discovered arcs for refining the original causal network, based on the considerations described in section 4.4.2. Results indicated that agents actually selected different strategies in different scenarios. Bayesian learning produced a change on the causal effect of actions, which in turn modified the FFn and the selection of strategies.

Bayesian learning uses past experiences of agents (closed SCN Cases), hence changes on their probabilistic models are result of environmental response to their actions. Experimentally, I simulated the response of human users and confirmed that agents adopted different strategies based on these simulated behaviors and the preferences of the organization. In this way can be stated that agents adapted to environmental changes, represented by human behaviors.

Planning can be defined as the capability of the agent for identifying chains of events and actions that lead to the achievement of a goal. This procedure is already provided by CCalc [34], but my framework still doesn’t incorporate this functionality. Causal descriptions of actions could be extracted from the SCNs using the semantic annotations of causes and effects. Nevertheless, CCalc doesn’t support probabilistic inference, so it would be necessary to adapt the CCalc algorithm in this framework.

Goal alignment does not prevent deadlocks. Further research must be done in this sense. Each agent knows its goals and the subgoals derived of them, but these goals can be result of a request made by another agent that in turn is pursuing another goal. In the end, goal tracing can be implemented through some protocol and once that the ultimate goal be identified the owner of the goal could be asked to solve the conflict. This resolution could be automated using goal preferences, if every essential goal of an agent, and respectively an organization, has a different value.

This and other kinds of conflicts can be resolved by the efficient cause of the system. Given that every goal can be traced back to an initial essential goal associated with the purpose of the system, the owner or pursuer of such goal, i.e. the system administrator, should be capable of solving these conflicts.

Cooperation between agents can be expressed as result of the alignment of their goals. If agent A knows the roles an agent B is playing it can deduce the essential goals B is pursuing. In this sense, A can inform B of certain condition that triggers a goal in B. Even if agent B is not playing a role in the organization, its agent class describes all the roles it can be asked to play in the organization.

The agent class (agent’s formal cause) also exposes certain information (properties) that
other agents might consider before asking him to participate or that can be considered before instancing an agent. Agent's essential information can be used to determine if the agent will be helpful in a given scenario. This functionality is hard-coded in the actual implementation but it can be implemented using a DL reasoner and creating all the possible interpretations of an agent class in order to identify one that satisfies (subsumes or contains) an actual agent request.

Finally, I can describe how causality can be used for solving disagreement between agents. In experiments I detected a conflict on which an agent $A$ asked an agent $B$ to perform certain action $\alpha$ in order to achieve a goal. Both agents were controlling plans extracted from the same process specification oriented towards the achievement of an organizational goal. Nevertheless, agent $B$ decided to omit $\alpha$ as long as he believed that its omission contributed more to its finality than its execution. In this case, $A$ had more information than $B$, which allowed him to make a better assessment.

In general, disagreement occurs when two agents have different information that bias the causal effect of their actions on a common goal. Missing information can be classified as: 1) unobserved preconditions or 2) unknown side effects. Both of them can be represented by variables and semantic annotations in their plans, and be associated as causes or effects, respectively, of actions or events contained in these plans. In this sense, agreement would be met through the incorporation, in the plan of one or both agents, of the minimal number of variables and causal relations that produce the same (or a similar) FFn for the action that motivated the conflict.

A conflict resolution protocol can be devised in this sense. Unlike in open systems, I'm assuming that: 1) agents assign the same meaning to the vocabulary they use (ontological commitment [38]), 2) individual plans derive from a process specification, and 3) agents know the common goal they are pursuing and the organizational metrics they are affecting. This protocol would start with the exchange of variables and relations involved in the calculation of the causal effect of the conflicting action. An agent would incorporate missing variables and arcs in its plan if this achieves the agreement of the parts in all possible cases. The solution would be optimal if the number of incorporated variables and relations is minimal. Additionally, the most informed agent should commit to inform the other agent about former missing causes. Given the aforementioned assumptions the protocol would be complete.

### 9.3 Ontological modeling of domains

I presented an ontological framework inspired in classical philosophy for modeling intelligent entities in terms of resources, knowledge and goals that intervene on its action. This framework uses Description Logics (DL) formalisms for defining these basic concepts and for defining domain-specific beings grounded in a causal classification of entities.

Meanwhile Object-Oriented paradigms consider causes as parameters of an action, I consider causes as necessary conditions on which it can be identified the entities involved
in its execution. Nevertheless these conditions are not enough for the execution of the action. Instead, it must be a final cause (an intention) that motivate its execution. My dynamic representation of actions is inspired in C+ [34] and DL-actions [6].

Rather than defining broad classes and properties [30, 89, 36, 23, 77], I keep the domain-dependent orientation of ontologies providing basic concepts that forces the user to model the domain in terms of actions and entities involved in a goal oriented task. The resulting ontology provides a formal specification of the intelligent entities required for automating the modeled organizational processes.

An essential goal represents an intention for changing a state of affairs, rather than merely reaching or avoiding one. Essential goals are represented by two conjunctive queries (CQ), the former representing the initial state of affairs and the latter the target state of affairs. The creation or modification of entities is codified through the use of variables and predicates in both queries.

The possibility of calculating subsumption of conjunctive queries incorporated interesting properties in the framework. For example, it is possible to verify if a plan satisfies a goal or if a missing condition can be enabled through the execution of a plan (triggered by a subgoal). Nevertheless, the constraint on the use of negations made difficult to express certain goals. For instance, the absence of an agent could not be expressed in the initial condition of the goal but it was expressed in the plan using negation as failure.

9.4 A semantic and probabilistic plan representation

I presented a novel representation of plans based on Bayesian Causal Networks and Description Logics formalisms. This representation allows: 1) describing a process from a global perspective, 2) identifying its participants and their capabilities, 3) expressing actions probabilistically in term of causes and effects, 4) generating the domain ontology, 5) generating plans and a preliminary communication protocol for each participant, 6) identifying and selecting strategies, 7) guiding plan execution and evaluating available options, 8) hiding implementation details, 9) identifying the effect of actions on (un)observable effects, 10) performing automatic subgoaling, 11) learning through experience and 12) mediating between conflicting goals. Additionally, semantic annotations can be further exploited for improving plans through the incorporation of causal rules and for devising new learning algorithms.

The Semantic Bayesian Causal Model (SBCM) formalism is the result of the integration of semantic annotations on a Bayesian causal network [71]. In order to represent organizational processes and plans, the SBCM was further extended for representing multiple occurrences of partial subnetworks and unobservable effects. The resulting formalism, which I called Semantic Causal Network (SCN), allowed representing processes in terms of participants, resources, knowledge, goals, metrics, actions, participant attributes, and
9.4. A SEMANTIC AND PROBABILISTIC PLAN REPRESENTATION

Events, all of them arranged in combinations of sequential and parallel/choice workflows.

The closest approach I found in literature was the Multi-Entity Bayesian Network (MEBN) approach [53]. This formalism also integrates logic and probabilistic inference but it is intended for representing probabilistic knowledge bases as an extension of Bayesian belief networks. Unlike MEBN, I represent beliefs through ABoxes and make the probabilistic inference in the plan representation. Besides, meanwhile MEBN uses quantifiers for aggregating evidence, I use them for eliminating unnecessary trials. Beyond, both approaches cannot be compared.

Concepts and predicates used in semantic annotations are used for defining the ontology of the system. In addition, due to the use of conjunctive queries it is possible to identify containment relationships between annotations of SBCMs/SCNs. The annotation mechanism also opens the possibility for grouping multiple interpretations of actual events and adapting the model to changes in the ontology through query rewriting.

The SBCM formalism captures the actions a participant can perform and the requirements imposed to him by the process. Furthermore, the Causal MAS methodology uses this description for extracting automatically the plans that participants must follow.

Ordered Binary Decision Diagrams (OBDDs) [2] were used for representing strategies in a compact format and use them in the decision model. It was shown that a single query to the probabilistic model can simultaneously evaluate multiple strategies. I also developed the Ordered Causal Decision Diagram (OCDD), a graphical representation complementary to the OBDD, on which there are identified the conditions on which the participants act upon the process goal. This representation allows identifying message exchange between process participants.

The OCDD and the OBDD obtained from process views of the same SCN reflect the very same strategies identified in the process. Strategies depend on the structure of the network and its probabilistic distribution. The modification of one or both of them may produce a different set of strategies, and as a consequence learning can modify the OBDD (decision model) and OCDD (communication protocol) of every participant.

SCN cases are represented by a set of semantic and probabilistic values associated to a SCN. This representation allows representing historical information of a plan or process, enabling learning from experience. It is also used to represent the actual execution of the plan or process, and along with the OBDD guides agent decisions.

Composite actions can in turn be represented by SCNs. This would allow a progressive specification of the process on which implementation details are hidden for other participants.

These formalisms allowed the implementation of a goal-driven Belief-Desire-Intention inference engine extended with semantic and probabilistic capabilities. Semantic annotations were used for implementing an automatic subgoaling mechanism. Options are evaluated considering possible worlds, derived from unobserved action preconditions. Missing preconditions are set through the instantiation of new goals that in turn trig-
CHAPTER 9. CONCLUSIONS

gers the execution of plans. The plan $P$ is said to enable the condition $C$ if the target condition, $Ann_P(F)$, of $P$ contains $C$, i.e. $C \subseteq Ann_P(F)$.

I introduced the notion of unobservable effects to the Bayesian Causal Network formalism. Keeping the Markovian condition, the causal effect of actions on unobservable variables is considered on decision making. Unlike unobservable causes, unobservable effects do not require the Do calculus introduced by Pearl. Additionally, predictive inference (from causes to effects) was optimized using the causal assumptions proposed by Pearl.

Built upon Bayesian Networks, the SBCM formalism can use well known Bayesian parametric and structural learning algorithms. In this way, past experiences, codified through closed SCN cases, are used for learning through experience.

This framework provides an explicit representation of conflicting goals. Conflicting goals are represented by SCN covariates and negative-positive preferences. Unlike other BDI [46, 11] and goal-driven BDI [47, 73] agent architectures, the finality function allows mediating between conflicting goals, which can be associated to the process target (hard goal) or to organizational side effects (soft goals) [88]. Furthermore, goals can be associated to indirect (and probably unobservable) effects.

Additionally, conflicts between plans [58], detected and solved by the system administrator, can be further prevented. This would be done by introducing inhibiting action causes or negative organizational side effects. The former would represent certain conflicts meanwhile the latter would represent probable conflicts. In the first case the agent would be incapable of selecting the action, meanwhile in the second, the agent would try to avoid the conflict selecting some strategy where the causal effect on the side effect would be minimized.

The SBCM formalism can further be used for performing data mining in RDF repositories [67]. A SBCM can represent an initial hypothesis that can be refined using Bayesian learning or some kind of generalization or specialization on semantic annotations.

9.5 Causal modeling of intelligent entities

I introduced the Causal Artificial Intelligence Design (CAID) theory, which is constituted by four principles that guide the design and implementation of a kind of intelligent entities. I applied such principles in the specification of Intelligent Agents and Intelligent Organizations, and for modeling the participation of human users in organizational processes.

According to CAID, intelligent entities are defined by four main causes that intervene on its action (formal, material, efficient and final causes). In addition, intelligent entities are obliged to act according to the purpose of its design. This purpose includes a clear identification of an individual purpose aligned with a common goal given by the designer of the system for which the intelligent entity is developed. Insofar as the intelligent entity
9.6. A CAUSAL GOAL-DRIVEN BDI AGENT ARCHITECTURE

is aware of the consequences of its actions, its decisions will be guided by its individual purpose and by goals adopted by its participation in a collectivity.

This definition of intelligent entities was extended to Intelligent Organizations, Intelligent Agents and Human Users. The definition of an intelligent organization was used for modeling top-down an agent-based system. In the opposite way, the definition of intelligent agent was used for considering the minimal elements that should be considered in the agent architecture for enabling cooperation towards a common goal. Finally, human user definition proposes that human participants cannot be forced to act or be punished directly in the system, but instead they can be modeled for identifying uncompliant behaviors and avoiding their participation if this is harmful for the system.

9.6 A causal goal-driven BDI agent architecture

I designed and implemented an agent architecture called Causal Agent, which is based on CAID principles and uses the introduced semantic causal models. This architecture has a goal-driven BDI inference engine. Beliefs and perceptions are represented in a format of triplets (subject-predicate-object) compatible with recent industrial standards for representing and enquiring ontologies and data (OWL, RDF, SPARQL). Inference rules are used for indicating the incorporation of perceptions into beliefs and for the revision of the last. Desires (goals) and intentions (options) are represented causally through the ontological framework and the SCN formalism.

Like in SOAR [66] and unlike other goal-driven BDI architectures [47, 73], subgoaling is triggered automatically. This is possible thanks to a preliminary discovery phase of enabling plans inferred from semantic descriptions of plans and goals. Subgoaling is triggered by a definition similar to SOAR's impasse: actions requiring to set missing conditions initiate auxiliary plans through subgoals. Subgoaling is motivated by action decisions represented by covariates \( X_i \), but is not triggered on the absence of pending actions.

Plan representation allows selecting actions from different plans alternatively. JADEX [73], on the other hand, creates a thread for each plan, which usually produces concurrency problems. The Causal Agent architecture reduces this risk to action scope. The developer must assure that atomic actions do not access shared resources; otherwise actions should be classified as composite and be described through another SCN.

The probability of success of plans is expressed probabilistically \( P(F|C) \) and it is updated thanks to Bayesian learning. This solves the limitation of JADEX and other goal-driven architectures of selecting the most appropriate plan for a given goal when multiple choices are available.

Developing a Causal Agent consist on selecting one or more compatible roles specified in one or more organizational processes, complementing the agent specification with additional constraints and implementing the code for actions in Java. Incompatible
roles can be detected through a concept reasoner [39, 70], by checking the consistency of agent classes.

Message reception and delivery is associated to plan representation and carried out automatically. The communication protocol is grounded in process specifications and each agent builds a view of each protocol on which it can participate. Message reception is controlled by perception rules whereas message delivery is controlled by the execution of the respective trials associated to goals (SBCM instances).

Similarly to agents in the Electronic Institution framework, the behavior of Causal Agents is guided by control structures. But meanwhile in Els the developer must codify in Java the decision for following a path (strategy) or another, the SCN Case allows the agent to choose a strategy automatically. This decision is based on process quantifiers and the finality of the agent and the system (finality function). Quantifiers overcome the limitation of not having cycles in Bayesian Networks.

Using this agent architecture, I implemented three types of agents: 1) Specialized Agents, focused in simple tasks where they intervene directly, 2) User Agents, which represent the intervention of a human user in the system, and 3) Manager Agents, which monitor and facilitate organizational processes on behalf of human users.

9.7 A methodology for automating organizational processes

I introduced the Causal MultiAgent System (CMAS) methodology for modeling organizational processes with the participation of intelligent agents and human users. My methodology starts modeling the original process through a SCN, considers automation of the process through the introduction of intelligent agents supervised by human users, produces a set of models compatible with the Causal agent architecture for each role in the process, and finishes with an ontological specification of the process and its participants.

I developed a tool in Java that automatically generates the plan of each participant in the organizational process. Additionally, this tool generates the OBDD, the OCDD, and a preliminary communication protocol for the hole process and for the corresponding process views. This tool also generates automatically graphical representations of the generated models through Graphviz [31] and imports/exports the Bayesian network from/to the BIF format used in Weka. In this way, subsequent modifications of the process specification are propagated automatically to most of the submodels used by agents, except for the communication protocol.

Meanwhile methodologies like Gaia and Prometheus use multiple types of diagrams for refining the specification, I propose the use of a single process/plan representation (SCN) and the use of ontological constraints with this purpose. Further submodels are derived from them automatically using the CMAS Tool. Besides, concept reasoners can
be used for verifying the consistency of the process specification.

All the models are expressed using OWL and RDF, hence their management and modification can be done by current open-source or commercial editors. Besides, thanks to the use of DL formalism, inference tasks can be expressed as subsumption tests and be performed by concept reasoners [39, 70].

9.8 Applications on autonomic processes

I formulated Autonomic Information Auditing (AIA) as an organizational process on which auditing an information repository is carried out by intelligent agents and human users with the following objectives: 1) self-configuring populations of agents, 2) minimizing unnecessary supervision of human users while maximizing the confidence in the repository consistency (self-optimization), 3) discarding inaccurate expert knowledge introduced in the system (self-protection), and 4) choosing different strategies when environmental conditions change (self-healing).

This process was modeled through the CMAS methodology producing SCN specifications of organizational process. The CMAS tool probed the feasibility of automatically generating models from a SCN process specifications. Besides, the refinement of the specification was fostered through the graphical representation of SCNs, which allowed introducing events and on considering new conditions in current nodes.

The CMAS specification provided the elements necessary for the implementation of the application using the Electronic Institutions formalism and tools developed at the IIIA-CSIC. Agent classes derived of process specifications were concretized in agent implementations using the ABuilder tool [5].

Then I developed Causal Agent implementations that made use of the models derived from process specifications. The inheritance mechanism of DL allowed to assign goals and plans to groups of agents in cascade. Agent classes and plans were customized with additional constraints and rules. The use of constraints allowed defining concrete classes of agents. Causal agents were instantiated calling an organizational protocol through subgoaling.

Even when the developed tool is capable of generating automatically the communication protocol for the two modeled processes, it makes strong assumptions that does not allow generalizing this procedure. More information must be considered in order to have a complete definition of information production and awareness.

Finally I performed a set of experiments using the Causal Agent implementations. I measured changes on the strategies and the communication protocol of the participants on every scenario and compared the overall efficiency in terms of goal achievement. I observed the effects of introducing organizational metrics in agent decisions, and the adapting of internal agents to human users response and inaccurate automatic correction rules.
Experimental results showed that the system reconfigured itself in response to external stimuli and reached a stable operation. The introduction of the new organizational metric affected the behavior of internal agents, which adopted different strategies on different scenarios. Finally, structural learning allowed making a different treatment of both correction rules, based on the response of auditors and authors to their application.

I also observed how two agents (PubCarrier and Auditor) made different decisions for the same case. Even when the plan controlled by the Auditor is a subset of the PubCarrier’s plan, when both agents calculate the causal effect of their actions on the target variables \( Z_{19} \) and \( Z_{20} \), the different information possessed by them produced different decisions.

### 9.9 Future Work

This research can be extended in several ways. The most relevant aspects are the following:

- We need additional information for generating the communication protocol. The actual policy used for generating it makes strong assumptions that avoid generalizing its use. For instance, the effects of an action performed by an agent \( A_1 \) may not being directly perceived by \( A_1 \), rather than, effects may be perceived by another agent \( A_2 \). In this case the agent \( A_2 \) should inform \( A_1 \) the results of its action, which is opposite to what currently dictates the policy. As can be seen we need to know which agent is aware of the event in order to decide the message direction.

- Incoming messages can be used for knowing that certain condition will be informed by other agent, avoiding subgoal on such conditions. Or in the opposite way, it can be specified that certain conditions are informed by other agents, which can be used for determining the direction of a message during the protocol generation phase.

- The communication protocol must also allow recognizing differed decisions. If an agent \( A_1 \) believes certain action \( X_i \) was performed and other agent \( A_2 \) believes it was not \( (X_i = False) \), the belief of the agent that directly executed \( X_i \) should prevail. Otherwise, as observed in experiments, learning is biased.

- In the same way, once that an agent requesting an action from another agent is informed of the refusal, it could start a negotiation protocol. Such protocol would be based on causal relations and causal effects. In the end, one of the agents would incorporate additional variables and causal relations in its plan and the other agent would commit to deliver such information regularly. The negotiation protocol proposed in [60] can be adapted with this purpose.
9.9. FUTURE WORK

• Expressiveness of the ontological framework can be improved with 1) the introduction of unions of conjunctive queries (UCQs), and 2) the incorporation of two additional CQs in the representation of goals: one for representing conditions not holding in the initial state of affairs and another representing undesired states of affairs. Goal representation can also be improved by representing cancelation and termination conditions.

SCN Quantifiers can also be extended. We can use the actual cardinality constraints proposed in DL constructors: at least N, at most N, and exactly N.

• The SBCM formalism can be extended by considering temporal predicates, but a special containment operation must be provided for this purpose. A similar extension can be done for geographical references. Extensions require a containment operation and a distinction of these concepts and predicates in the ontology.

• We need to explore the capabilities of the CMAS specification in other application domains, and verify extensively the robustness of agents to specification changes. For instance, sensor and actuator implementations can be validated using their semantic description and process specifications.

• A visual tool for the specification and generation of models would be useful in the adoption of this framework by system developers.

• Learning capabilities of the SCN formalism can be further exploited. For instance, structural learning can make use of semantic annotations and subsumption relations among them. Besides, the use of a sequential learning algorithm [29, 52] would allow to recalculate the probabilistic distribution every time a new case is observed. We must also consider failed or incomplete SCN cases which would be considered instances with missing information.

• The figure of Process Managers (PMAs) and Process Monitors (PMOs) proposed in the CMAS methodology allowed implementing a centralized learning approach. In contrast, in the MultiAgent Causal Model approach [59] it was proposed a distributed learning algorithm [64] that can be incorporated on the CMAS architecture.

• In order to try again a dropped strategy it can be injected artificially created cases with this strategy, until that its efficiency surpasses the efficiency of the actual strategy. This operation can be limited to a given context of the plan, represented by a set of variable realizations.

• Composite actions can be implemented through a nested plan expressed as a SCN. In this way, the execution of a composite action would be equivalent to generating a subgoal.

• We could generate a coupling diagram like the one proposed on the Prometheus Methodology [68] for selecting organizational roles to group in concrete agent classes.
• Causal diagrams and goal dependencies can be used for detecting the cause of a system deadlock. As long as all the goals in the system are aligned with respect to a global goal, the possible reasons for a conflict would be a failure in a protocol, concurrent access to or depletion of a resource, among others. These causes would be represented by missing conditions in actual plan executions.

The actual implementation of Causal Agents consumes so many resources but can be optimized in the following ways:

• Using an algorithm for pruning ontologies like [22], each agent can build a subset of the organizational ontology that contains only those concepts and properties considered on the annotations of the plans it controls and the rules it knows. This would optimize the belief representation by consuming less memory and processing time (used in evaluating entailment rules).

• We can incorporate a more efficient library for calculating posterior probabilities on the Bayesian model or implement a cache mechanism for remembering previous calculations.

Modeling of AIA through the CMAS methodology can be improved in the following ways:

• Considering publication confidence and human intervention with respect to the correction rather than to the publication.

• Representing the correction and its supervision \((X_3, X_4, X_5, X_6\) and the corresponding covariates) as a composite action controlled by the CorrectorAg. In this way, annotations of the selected variables wouldn't include references to publications without inconsistencies, reducing the complexity of the model.

• Likewise, the action Audit can be decomposed in several auxiliary plans, one for each type of inconsistency rule. This would allow describing the participation of other agents in such task.
Bibliography


[21] Hector Ceballos, Pablo Noriega, and Francisco Cantu. Requesting agent participation in electronic institutions (extended abstract). In *Proceedings of the Ninth*


