ABSTRACT

TELANG, PANKAJ RAMESH. Multiagent Business Modeling. (Under the direction of Dr. Munindar P. Singh.)

Cross-organizational business processes are common in today’s economy. Of necessity, enterprises conduct their business in cooperation to create products and services for the marketplace. Thus business processes inherently involve autonomous partners with heterogeneous software designs and implementations. The existing business modeling approaches that employ high-level abstractions are difficult to operationalize, and the approaches that employ low-level abstractions lead to highly rigid processes that lack business semantics. We propose a novel business model based on multiagent abstractions. Unlike existing approaches, our model gives primacy to the contractual relationships among the business partners, thus providing a notion of business-level correctness, and offers flexibility to the participants. Our approach employs reusable patterns as building blocks to model recurring business scenarios. A step-by-step methodology guides a modeler in constructing a business model. Our approach employs temporal logic to formalize the correctness properties of a business model, and model checking to verify if a given operationalization satisfies those properties. Developer studies found that our approach yields improved model quality compared to the traditional approaches from the supply chain and healthcare domains.

Commitments capture how an agent relates with another agent, whereas goals describe states of the world that an agent is motivated to bring about. It makes intuitive sense that goals and commitments be understood as being complementary to each other. More importantly, an agent’s goals and commitments ought to be coherent, in the sense that an agent’s goals would lead it to adopt or modify relevant commitments and an agent’s commitments would lead it to adopt or modify relevant goals. However, despite the intuitive naturalness of the above connections, they have not yet been studied in a formal framework. This dissertation provides a combined operational semantics for goals and commitments. Our semantics yields important desirable properties, including convergence of the configurations of cooperating agents, thereby delineating some theoretically well-founded yet practical modes of cooperation in a multiagent system.

We formalize the combined operational semantics of achievement commitments and goals in terms of hierarchical task networks (HTNs) and show how HTN planning provides a natural representation and reasoning framework for them. Our approach combines a domain-independent theory capturing the lifecycles of goals and commitments, generic patterns of reasoning, and domain models. We go beyond existing approaches by proposing a first-order representation that accommodates settings where the commitments and goals are templatic and may be applied repeatedly with differing bindings for domain objects. Doing so not only leads to a more perspicuous modeling, it also enables us to support a variety of practical patterns.
Multiagent Business Modeling

by
Pankaj Ramesh Telang

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science
Raleigh, North Carolina
2013

APPROVED BY:

______________________________  ________________________________
Dr. Jon Doyle                  Dr. Thom J. Hodgson

______________________________  ________________________________
Dr. John F. Madden             Dr. Emerson Murphy-Hill

______________________________  ________________________________
Dr. Tao Xie                    Dr. Munindar P. Singh
Chair of Advisory Committee
DEDICATION

To my wife, Anjali Telang for making numerous sacrifices to support me in this work.
BIOGRAPHY

The author was born in Amravati, a town in the state of Maharastra, India. He received his first M.S. in Industrial Engineering with Operations Research focus from the University of Cincinnati, Ohio in 1998. He received his second M.S. in Computer Science from the NC State University in 2010. Additionally, he has been working in the software industry since 1998, initially as an IT Engineer, and later as an IT Architect.
ACKNOWLEDGEMENTS

I am deeply indebted to my advisor Professor Munindar Singh for his teachings and relentless support during this work. Without his guidance, this work would have been impossible. Beyond the skills for effective research, I have learnt the qualities of a good teacher from him. I wish to be a researcher and a teacher like him.

I would like to especially thank Neil-Yorke Smith, Felipe Meneguzzi, and Anup Kalia for extensively collaborating with me on portions of this work. I learnt a great deal about goals from Neil, which enabled the development of joint operational semantics on goals and commitments. I learnt extensively about AI planning from Felipe, which enabled the development of HTN planning formalism for goals and commitments. Anup has supported me in the developer evaluations, and implemented Protos modeling tool.

Several other individuals have reviewed and provided critical suggestions to improve my work. I would like to thank, in no particular order, Scott Gerard, Amit Chopra, Chung-Wei Hang, Derek Sollenberger, Nirmit Desai, Prashant Kediyal, Pradeep Murukannaiah, Christoph Bussler, and M. Birna van Riemsdijk.

I am deeply grateful to my advisory committee members Dr. Jon Doyle, Dr. Thom J. Hodgson, Dr. John F. Madden, Dr. Emerson Murphy-Hill, and Dr. Tao Xie for advising me during this work. Their valuable suggestions have significantly improved the quality of this work.

Finally, the sacrifices made by my wife and my kids during this work overwhelm me. They patiently allowed me to work during countless weekends and evenings. Their unwavering support enabled me to undertake and finish this endeavor.
# TABLE OF CONTENTS

LIST OF TABLES ............................................................... ix

LIST OF FIGURES ........................................................... x

Chapter 1 Introduction ......................................................... 1
  1.1 Motivating Examples .................................................. 2
    1.1.1 Order-To-Cash ................................................. 2
    1.1.2 Breast Cancer Screening and Diagnosis ..................... 4
  1.2 The Proposed Approach .............................................. 5
    1.2.1 Metamodel and Patterns ..................................... 5
    1.2.2 Methodology and Evaluation .................................. 6
    1.2.3 Relating Commitments and Goals ............................ 7
    1.2.4 Hierarchical Planning about Commitments and Goals ....... 7
  1.3 Contributions ....................................................... 7
  1.4 Organization of the Dissertation ................................... 8

Chapter 2 Background ....................................................... 9
  2.1 Achievement Commitments .......................................... 9
    2.1.1 Lifecycle of a Commitment ................................ 9
  2.2 Achievement Goals ................................................ 12
    2.2.1 Lifecycle of an Achievement Goal ......................... 13
  2.3 Maintenance Goals ................................................ 14
    2.3.1 Lifecycle of a Maintenance Goal ............................ 15
  2.4 Computation Tree Logic ............................................ 16
  2.5 NuSMV Model Checking Tool ....................................... 17
  2.6 Classical and HTN Planning ....................................... 19
    2.6.1 Logic Language .............................................. 19
    2.6.2 Classical Planning ......................................... 20
    2.6.3 Hierarchical Task Network (HTN) Planning ................. 21
  2.7 UML 2.0: Sequence Diagrams ...................................... 21
  2.8 RosettaNet .......................................................... 22
  2.9 HL7 ................................................................. 23

Chapter 3 Metamodel, Patterns, and Verification ..................... 24
  3.1 Metamodel ........................................................... 24
  3.2 Business Model Patterns .......................................... 25
    3.2.1 Conditional Offer Pattern ................................ 26
    3.2.2 Commercial Transaction .................................... 26
    3.2.3 Outsourcing ................................................. 27
    3.2.4 Standing Service Contract .................................. 27
  3.3 Applying the Patterns ............................................. 28
    3.3.1 Quote-To-Cash Process .................................... 29
  3.4 Verifying Business Models ........................................ 32
3.5 Patterns Formalized .................................................. 33
  3.5.1 Conditional Offer Pattern Formalization .................. 33
  3.5.2 Commercial Transaction Pattern Formalization .......... 39
  3.5.3 Outsourcing Pattern Formalization ....................... 39
  3.5.4 Service Contract Pattern Formalization ................ 39
3.6 Applying Verification .............................................. 40
3.7 Runtime Violations ................................................ 42
3.8 Conclusion .......................................................... 43
3.9 Related Work ....................................................... 44

Chapter 4 Comma: Modeling Methodology .................................. 47
  4.1 Methodology: Comma ............................................. 48
  4.2 Design of the Study ............................................. 49
    4.2.1 Study Solution ............................................ 50
    4.2.2 Study Mechanics and Threat Mitigation ................. 52
    4.2.3 Dependent Variables .................................. 53
  4.3 Study Results ................................................ 54
    4.3.1 Quality .................................................... 54
    4.3.2 Difficulty ............................................... 56
    4.3.3 Time .................................................... 58
  4.4 Conclusion ..................................................... 60
  4.5 Related Work ................................................ 60

Chapter 5 Modeling Healthcare Processes .................................. 62
  5.1 Dialectical Commitments ....................................... 62
    5.1.1 Lifecycle of Dialectical Commitments .................. 63
  5.2 Organizational Context ....................................... 68
  5.3 Commitment Patterns .......................................... 69
    5.3.1 Service Provider with Correctness .................... 69
    5.3.2 Escalate ................................................ 69
    5.3.3 Chained Service Providers with Joint Correctness .... 70
  5.4 Applying the Patterns ........................................ 71
    5.4.1 Breast Cancer Screening and Diagnosis ............... 71
    5.4.2 Comparison with the Traditional Approach ............ 73
  5.5 Applying Verification ......................................... 74
  5.6 Empirical Evaluation of Comma compared to HL7 ............ 77
    5.6.1 Managing Threats ....................................... 78
    5.6.2 Dependent Variables .................................. 79
  5.7 Results of Empirical Evaluation .............................. 80
    5.7.1 Flexibility and Objective Quality ....................... 80
    5.7.2 Subjective Quality ..................................... 81
    5.7.3 Time and Difficulty .................................... 81
    5.7.4 Hypothesis Testing ..................................... 82
  5.8 Conclusion ..................................................... 82
  5.9 Related Work ................................................ 83
<table>
<thead>
<tr>
<th>Chapter 6</th>
<th>Commitments and Goals</th>
<th>86</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Operational Semantics</td>
<td>86</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Assumptions</td>
<td>87</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Formalization</td>
<td>89</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Practical Rules</td>
<td>91</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Nested Commitments</td>
<td>99</td>
</tr>
<tr>
<td>6.2</td>
<td>Illustrative Application</td>
<td>100</td>
</tr>
<tr>
<td>6.3</td>
<td>Establishing Convergence Properties</td>
<td>104</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Proof by Model Checking: The NuSMV Model</td>
<td>110</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Output of Model Checking showing Convergence</td>
<td>113</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Output of Model Checking showing Lack of Convergence</td>
<td>114</td>
</tr>
<tr>
<td>6.4</td>
<td>Maintenance Commitments</td>
<td>115</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Lifecycle of a Maintenance Commitment</td>
<td>115</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Formalization including Maintenance Commitments and Goals</td>
<td>117</td>
</tr>
<tr>
<td>6.5</td>
<td>Operational Semantics including Maintenance Commitments and Goals</td>
<td>119</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Maintenance Goal to Achievement Commitment</td>
<td>120</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Maintenance Goal to Maintenance Commitment</td>
<td>121</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Maintenance Commitment to Maintenance and Achievement Goals</td>
<td>121</td>
</tr>
<tr>
<td>6.5.4</td>
<td>Nested Achievement Commitment to Achievement Goals</td>
<td>122</td>
</tr>
<tr>
<td>6.5.5</td>
<td>Achievement Goal to Maintenance Commitment</td>
<td>123</td>
</tr>
<tr>
<td>6.5.6</td>
<td>Achievement Goal to Nested Maintenance Commitment</td>
<td>123</td>
</tr>
<tr>
<td>6.6</td>
<td>Conclusion</td>
<td>124</td>
</tr>
<tr>
<td>6.7</td>
<td>Related Work</td>
<td>125</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 7</th>
<th>Hierarchical Planning about Commitments and Goals</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Formalization within HTN</td>
<td>129</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Goal and Commitment Dynamics</td>
<td>129</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Commitment Axioms and Operators</td>
<td>130</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Goal Axioms and Operators</td>
<td>131</td>
</tr>
<tr>
<td>7.1.4</td>
<td>Methods Relating Commitments and Goals</td>
<td>133</td>
</tr>
<tr>
<td>7.1.5</td>
<td>Bringing it all Together</td>
<td>133</td>
</tr>
<tr>
<td>7.2</td>
<td>Applying the HTN Formalization</td>
<td>134</td>
</tr>
<tr>
<td>7.3</td>
<td>First Order Formalization in HTN</td>
<td>138</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Commitment Dynamics</td>
<td>139</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Goal Dynamics</td>
<td>141</td>
</tr>
<tr>
<td>7.4</td>
<td>Formalizing the Patterns</td>
<td>143</td>
</tr>
<tr>
<td>7.5</td>
<td>Conclusion</td>
<td>146</td>
</tr>
<tr>
<td>7.6</td>
<td>Related Work</td>
<td>147</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 8</th>
<th>Discussion</th>
<th>149</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Conclusions</td>
<td>149</td>
</tr>
<tr>
<td>8.2</td>
<td>Future Directions</td>
<td>150</td>
</tr>
</tbody>
</table>

REFERENCES | 152 |
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1</td>
<td>Semantic postulates of Computation Tree Logic</td>
<td>17</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>Comma methodology steps.</td>
<td>48</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>RosettaNet model PIPs for $S_i$.</td>
<td>51</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Study exercises.</td>
<td>52</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Quality measures, as judged by experts.</td>
<td>54</td>
</tr>
<tr>
<td>Table 4.5</td>
<td>Hypothesis testing for model and MSC development times.</td>
<td>59</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>Subscenarios and patterns from the breast cancer diagnosis scenario.</td>
<td>84</td>
</tr>
<tr>
<td>Table 5.2</td>
<td>Hypothesis testing via Student’s t-test.</td>
<td>85</td>
</tr>
<tr>
<td>Table 6.1</td>
<td>Syntax for beliefs, commitments, and goals in Backus-Naur Form.</td>
<td>89</td>
</tr>
<tr>
<td>Table 6.2</td>
<td>Goals and commitments from the aerospace scenario.</td>
<td>102</td>
</tr>
<tr>
<td>Table 6.3</td>
<td>Progression of configurations in the aerospace scenario.</td>
<td>103</td>
</tr>
<tr>
<td>Table 6.4</td>
<td>Syntax for beliefs, commitments, and goals.</td>
<td>119</td>
</tr>
<tr>
<td>Table 6.5</td>
<td>Practical rules: From m-goal to a-commitment.</td>
<td>121</td>
</tr>
<tr>
<td>Table 6.6</td>
<td>Practical rules: From m-commitment to m-and a-goals.</td>
<td>123</td>
</tr>
<tr>
<td>Table 7.1</td>
<td>Goals and commitments from the purchase example.</td>
<td>135</td>
</tr>
<tr>
<td>Table 7.2</td>
<td>Domain-specific axioms</td>
<td>136</td>
</tr>
<tr>
<td>Table 7.3</td>
<td>Domain-specific methods and operators.</td>
<td>137</td>
</tr>
<tr>
<td>Table 7.4</td>
<td>Goals and commitments for the patterns.</td>
<td>144</td>
</tr>
<tr>
<td>Table 7.5</td>
<td>Methods and operators for the patterns.</td>
<td>145</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1.1</td>
<td>Order-To-Cash business process</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Breast cancer screening and diagnosis process</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>Commitment lifecycle as a state transition diagram</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>Goal lifecycle as a state transition diagram</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>Maintenance goal lifecycle as a state transition diagram</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>TRANS constraint for a simple sequence diagram</td>
<td>22</td>
</tr>
<tr>
<td>2.5</td>
<td>TRANS constraint for a sequence diagram with an OPT</td>
<td>23</td>
</tr>
<tr>
<td>3.1</td>
<td>A commitment-based business metamodel</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Conditional offer pattern</td>
<td>26</td>
</tr>
<tr>
<td>3.3</td>
<td>Commercial transaction pattern</td>
<td>27</td>
</tr>
<tr>
<td>3.4</td>
<td>Outsourcing pattern</td>
<td>28</td>
</tr>
<tr>
<td>3.5</td>
<td>Standing service contract pattern</td>
<td>29</td>
</tr>
<tr>
<td>3.6</td>
<td>Commercial transaction: Customer-Reseller</td>
<td>30</td>
</tr>
<tr>
<td>3.7</td>
<td>Outsourcing: Reseller-Distributor</td>
<td>31</td>
</tr>
<tr>
<td>3.8</td>
<td>Outsourcing: Distributor-Shipper</td>
<td>32</td>
</tr>
<tr>
<td>3.9</td>
<td>Commercial transaction: Distributor-Cisco</td>
<td>33</td>
</tr>
<tr>
<td>3.10</td>
<td>Outsourcing: Cisco-Shipper</td>
<td>34</td>
</tr>
<tr>
<td>3.11</td>
<td>Service contract: Customer-Cisco</td>
<td>35</td>
</tr>
<tr>
<td>3.12</td>
<td>Quote-To-Cash business model</td>
<td>36</td>
</tr>
<tr>
<td>3.13</td>
<td>Verification approach</td>
<td>37</td>
</tr>
<tr>
<td>3.14</td>
<td>Verifying the conditional offer pattern</td>
<td>38</td>
</tr>
<tr>
<td>3.15</td>
<td>Operational interactions in a Quote-To-Cash implementation</td>
<td>40</td>
</tr>
<tr>
<td>3.16</td>
<td>NuSMV output indicating satisfaction</td>
<td>43</td>
</tr>
<tr>
<td>3.17</td>
<td>NuSMV output showing violation</td>
<td>44</td>
</tr>
<tr>
<td>4.1</td>
<td>Message sequence charts for outsourcing</td>
<td>47</td>
</tr>
<tr>
<td>4.2</td>
<td>Comma model for $S_i$</td>
<td>50</td>
</tr>
<tr>
<td>4.3</td>
<td>Example Comma MSCs for $S_i$</td>
<td>51</td>
</tr>
<tr>
<td>4.4</td>
<td>Example RosettaNet MSCs for $S_i$</td>
<td>52</td>
</tr>
<tr>
<td>4.5</td>
<td>Our approach of grouping the subjects</td>
<td>53</td>
</tr>
<tr>
<td>4.6</td>
<td>Quality of the models of $S_i$</td>
<td>55</td>
</tr>
<tr>
<td>4.7</td>
<td>Flexibility of the MSCs produced for $S_i$</td>
<td>55</td>
</tr>
<tr>
<td>4.8</td>
<td>Difficulty of modeling, as percentage of responses by the subjects</td>
<td>56</td>
</tr>
<tr>
<td>4.9</td>
<td>Time in hours expended in creating models, as reported by subjects</td>
<td>57</td>
</tr>
<tr>
<td>5.1</td>
<td>Dialectical commitment life cycle as a state transition diagram</td>
<td>64</td>
</tr>
<tr>
<td>5.2</td>
<td>Dialectical commitments in radiologist and pathologist interactions</td>
<td>65</td>
</tr>
<tr>
<td>5.3</td>
<td>Service provider with correctness pattern</td>
<td>69</td>
</tr>
<tr>
<td>5.4</td>
<td>Escalate pattern</td>
<td>70</td>
</tr>
<tr>
<td>5.5</td>
<td>Chained service providers with joint correctness pattern</td>
<td>70</td>
</tr>
<tr>
<td>5.6</td>
<td>Breast cancer screening and diagnosis business model</td>
<td>72</td>
</tr>
</tbody>
</table>
Chapter 1

Motivation

Real-world organizations seldom operate in isolation. To conduct business, organizations enter into complex contractual relationships with each other. As an example, in a supply chain, a seller may outsource the manufacturing of goods to a contract manufacturer, and their shipping to a shipper. In another example, an individual may purchase a health insurance policy from an insurance provider. The insurance provider may have a network of physicians. In such scenarios, to conduct the business, the organizations and individuals carry out subtle interactions with one another. The problem of engineering IT systems for such cross-organizational interactions is challenging, and the existing solutions are in their infancy.

With only a few notable exceptions, much of the existing work on business models emphasizes low-level or operational details of the interactions. Existing approaches specify the control and data flow among the business partners, but they fail to specify their business relationships. Such business relationships are known to business analysts and motivate the specification of the processes. Therefore, losing track of the relationships in the operational perspective is unfortunate: losing the notion of business correctness by which to judge operational execution complicates the creation and maintenance of each partner’s software systems. To address the shortcomings of existing approaches, we propose a novel multiagent business modeling approach based on commitments and goals.

The concepts of commitments and goals are intuitively complementary. A commitment describes how an agent (a business partner) relates with another agent, whereas a goal describes a state of the world that an agent is motivated to bring about. An achievement commitment from one agent (the debtor) to another (the creditor) states that the debtor promises to achieve the consequent if the creditor (first) achieves the antecedent. It carries normative force in terms of what an agent would bring about for another agent, whereas a goal describes an agent’s proattitude toward some condition. Developing a unified theory of commitments and goals would be significant for the following two reasons. First, it would close the theoretical gap in present understanding between the organizational (commitments) and individual (goals) perspectives, which are both essential in a comprehensive account of rational agency in a social world.
Second, it would provide a basis for a comprehensive account of the software engineering of multiagent systems going from interaction-orientation (with commitments) to agent-orientation (with goals).

Further, consider an agent that wishes to maintain a condition, such as a home owner who wishes to maintain his lawn using a contracted lawn mower. Such situations highlight the need for understanding maintenance commitments, and how they relate to achievement commitments, and achievement and maintenance goals.

To operationalize a multiagent business model that is declaratively specified in terms of goals and commitments, the agents (participants) need to be able to develop cost effective and feasible plans. There is a natural hierarchy in the plans that an agent requires to achieve its goals and to satisfy its commitments. For example, a goal may be decomposed into multiple conjunctive or disjunctive subgoals. The agent may satisfy a subgoal on its own, or may create a commitment toward another agent to satisfy it. If the commitment expires or is violated, then the agent may create a commitment toward some other agent, and sanction the violating agent. Not only can operations on commitments and goals feature within plans, but also to satisfy a commitment or a goal may require another hierarchical plan.

1.1 Motivating Examples

To further motivate the problem, we now describe two examples.

1.1.1 Order-To-Cash

Consider the so-called Extended Order-To-Cash process flow, which is specified as an important scenario in the RosettaNet eBusiness Process Scenario Library [RosettaNet, 2010]. Such a flow is representative of traditional business modeling approaches.

Figure 1.1 shows the Order-to-Cash business process scenario. The participants of this process are a supplier, a customer, and a shipper. The customer orders products from the supplier. Later, the customer may request a change to the order. The supplier engages a shipper for shipping the goods to the customer. Additionally, the shipper periodically sends an inventory report to the supplier. Prior to shipping the goods, the shipper notifies the customer. The customer notifies the supplier upon receiving the goods. Subsequently, the supplier sends an invoice to the customer. The customer validates the invoice, and thereafter sends a remittance advice to the supplier. Figure 1.1 shows the RosettaNet PIPs (messages) that the participants employ for the above interactions.

A model such as the above specifies the process as a workflow at an operational level. It shows the messages that the participants exchange, and their temporal ordering. But it completely ignores the business relationships among the participants. Thus, it leaves several important questions unanswered.

- What happens if the shipper fails to send a shipment notice? At a business-level is it mandatory for the shipper to send such a notice?
• Why must the supplier create an invoice only after the customer sends the receiving information? At a business-level, it may be acceptable for the supplier to create an invoice before the customer sends the receiving information.

• Why must the shipper process a shipping order? Is the shipper contractually bound to processing a shipping order that the supplier sends?

• Is it mandatory for the shipper to send an inventory report? Sending an inventory report is arguably not as important. In that case, the process model is being unnecessarily rigid in mandating the shipper to send the inventory report.

Thus, the model in Figure 1.1 leaves such important questions regarding the business meanings of the interactions to human interpretation. Each group of analysts and developers working in the partner organizations would negotiate such considerations between themselves, but doing so has the effect of introducing idiosyncratic constraints through which the partners become inadvertently tightly coupled. Further, there is no basis to judge if the model is correct. For example, the model in Figure 1.1 lacks an interaction in which the supplier makes a payment to the shipper. Should the model contain such an interaction?
1.1.2 Breast Cancer Screening and Diagnosis

Figure 1.2 shows another example from the healthcare domain. This is a simplified scenario adapted from ASPE [2010] that deals with breast cancer screening and diagnosis. The model shows four main participants, but omits the patient who receives the treatment, and the tumor board that serves as an authority to resolve any disagreements. A patient visits a primary care physician who detects a suspicious mass in her breast; he sends her over to a radiologist for mammography (imaging). The radiologist notices calcifications of about 0.50 mm, which are borderline suspicious. She forwards her report to the primary care physician recommending he order a biopsy, which he does. The radiologist performs a biopsy and sends a specimen to a pathologist, who analyzes the specimen, potentially performing additional studies. The pathologist and radiologist might hold a conference to ensure their results are concordant, and provide an integrated report to the primary physician for discussion with the patient. The physician reviews the integrated report with the patient to create a treatment plan. The pathologist sends a report to the registrar who adds the patient to the cancer registry.

The above scenario is representative of other diagnosis problems. With suitable modifications, virtually every healthcare situation involves such back-and-forth between multiple healthcare providers such as physicians and laboratories, for instance, in diabetes care [Glasgow et al., 2005]. Since the model from Figure 1.2 ignores business relationships, it suffers from serious shortcomings. In particular, it
cannot answer the following questions.

- What is the radiologist committing to in her report, and in the tissue specimen she sends to the pathologist?

- Can a pathologist easily deviate from the normal process by asking questions about how the specimen was taken?

- What happens if the patient does not arrive for the follow up visit for the biopsy?

- Can a service fulfillment be delegated? For example, can the radiologist’s report be produced by an off-shore radiologist based on images provided from the laboratory?

- Is it acceptable in an emergency setting for the primary care physician to concurrently request the radiologist for mammography and biopsy?

The above workflow expresses a business process captured in terms of the interlacing activities of the participants involved. Such representations are common in traditional business settings [OMG, 2010] and are best suited to highly regimented efforts by clerical workers. However, they are inadequate for modern cross-organizational settings. In particular, because traditional representations fail to capture business meaning, they preclude the following essential elements:

- Flexible enactment, essential for dealing with exceptions and corner cases, because without a proper meaning, there is no justification for any deviations from the stated flow.

- Model evolution, essential for accommodating requirements changes, because without a proper meaning, there is no justification for any modifications to the stated flow in light of a new requirement.

- Verification, essential for correctness of an operationalization, because without a proper meaning, it is difficult to correlate low-level operational events with their potential meanings.

### 1.2 The Proposed Approach

This section summarizes our proposed approach.

#### 1.2.1 Metamodel and Patterns

Our approach contains a simple business metamodel using which we can express a model for a particular cross-organizational scenario. Since the participants in a cross-organizational scenario are real-world organizations and thus autonomous and heterogeneous, our metamodel represents them computationally
as *agents* [Singh and Huhns, 2005]. An agent has *goals*, and executes business *tasks*. A *role* is an abstraction over an agent. For each business relationship in which an agent participates, it enacts one or more roles.

The business relationships among the roles are expressed in terms of their *commitments* to one another. Each role specifies the commitments expected of the agents who play that role along with the tasks they must execute to function in that role. A goal is a state of the world that an agent desires to be brought about. An agent achieves a goal by executing appropriate tasks or by negotiating with other agents to have them execute appropriate tasks. A task here is a business activity viewed from the perspective of an agent.

A *practical commitment* is an element of a contractual business relationship. A commitment $C(\text{debtor}, \text{creditor}, \text{antecedent}, \text{consequent})$ denotes that the debtor commits to the creditor to bring about the consequent if the antecedent begins to hold [Singh, 1999]. A business relationship is a set of interrelated commitments among two or more roles that describe how they carry out the given business process.

A *dialectical commitment* pertains to an agent making an assertion about certain state of affairs. Note how dialectical commitment is in contrast to a practical commitment which involves a debtor bringing about certain condition typically by performing an action. As an example, by providing diagnosis report, a doctor makes a dialectical commitment to a patient for the correctness of that report.

A *pattern* is a recipe for modeling recurring business scenarios. Our approach seeks to develop a key set of patterns that could seed a potential business model pattern library. A few examples of the patterns are: commercial transaction and outsourcing.

Our approach verifies an operational model specified as a set of UML 2.0 sequence diagrams with respect to a business model. We employ NuSMV, an existing model checker. We map the business model to a temporal logic specification regarding the progression of the states of the relevant commitments, and the UML 2.0 sequence diagrams to a model in NuSMV input language. NuSMV takes the temporal logic specification and the model as input, and reports if the model satisfies the specification.

### 1.2.2 Methodology and Evaluation

We develop Comma, a methodology for multiagent business modeling. Comma methodology guides a designer in producing a formal model starting from an informal description of the business scenario. The methodology employs the business patterns. We have conducted rigorous developer studies comparing our approach with traditional approaches from the supply chain and healthcare domains. The studies compared the modeling time, the modeling difficulty, and the model quality. The design of our studies addressed the internal and external threats to the validity of the study results, and employed well-known statistical tests of difference such as the t-test for comparing the study observations.
1.2.3 Relating Commitments and Goals

We present the lifecycles of commitments and goals as state transition diagrams. Next we develop the combined operational semantics based on guarded rules. We term these practical rules because they capture patterns of practical reasoning that an agent may adopt. We propose and prove convergence properties for agents that adopt our practical rules. We employ model checking over CTL formulas to establish the proof.

We propose a lifecycle of maintenance commitments. We extend the work on relating achievement commitments and goals to maintenance commitments and goals. Further, we detail how achievement and maintenance commitments and goals can be composed.

1.2.4 Hierarchical Planning about Commitments and Goals

We formalize the combined operational semantics of achievement commitments and goals in terms of hierarchical task networks (HTNs) and show how HTN planning provides a natural representation and reasoning framework for them. Our approach combines a domain-independent theory capturing the lifecycles of goals and commitments, generic patterns of reasoning, and domain models. Specifically, our approach shows how each agent may take into account its capabilities, costs, and preferences as it plans its interactions (captured as operations on commitments) with other agents to attempt to achieve its goals.

We enhance the HTN formalization to accommodate first-order representations and reasoning. Although we confine ourselves to finite domains, the first-order representation leads to a more natural encoding for domain scenarios involving commitments. Our proposed approach further accommodates patterns for reasoning about commitments and goals that previous approaches do not tackle cleanly.

1.3 Contributions

We now summarize the key contributions of this dissertation.

**Metamodel, Patterns, and Verification:** This dissertation proposes a simple, yet expressive declarative way to specify business models at a high level based on the notion of commitments. In addition to practical commitments, it considers dialectical commitments. It shows how a commitment-based model maps to a conventional operational model. It provides a basis for verifying the correctness of the operational representations with respect to the declarative business model using existing temporal model checking tools. This dissertation validates the above claims on scenarios from the supply chain and healthcare domains.

**Methodology and Evaluation:** This dissertation proposes Comma, a business modeling methodology, that builds upon our proposed metamodel and patterns. We conduct developer studies comparatively evaluating Comma with respect to RosettaNet [2008], a supply chain industry standard, and HL7
[2007], a healthcare industry standard. Our results confirm the relative effectiveness of Comma for the quality of modeling cross-organizational processes, and some benefits in ease of modeling and time expended. Further, the results yield insights for future improvements.

**Joint Operational Semantics for Commitments and Goals:** This dissertation studies the complementary aspects of commitments and goals by establishing an operational semantics of the related life cycles of the two concepts. We distinguish the purely semantic aspects of their lifecycles from the pragmatic aspects of how a cooperative agent may reason. We establish a set of convergence properties that follow from our semantics, and prove them by model checking. We illustrate our semantics with a well-known insurance scenario [Browne and Kellett, 1999]. Further, this dissertation proposes maintenance commitments as a construct in their own right and presents a novel operational semantics linking maintenance and achievement goals and commitments. We illustrate our semantics including maintenance on a scenario in the domain of aerospace aftermarket services [van Aart et al., 2007].

**Hierarchical Planning about Commitments and Goals:** This dissertation proposes a new treatment of commitments and goals that accommodates first-order representations and reasoning via the Hierarchical Task Network (HTN) formalism. Moreover, we show how our formalization overcomes expressivity limitations found in previous work. Thus, our contribution is twofold. First, we provide a formalization of commitments and goals that enables us to use an off-the-shelf HTN planner in the verification of the realizability of a multiagent system. Second, we identify key limitations in the expressiveness of existing formalizations and reasoning mechanisms and develop reasoning patterns in our formalism that address these limitations.

### 1.4 Organization of the Dissertation

Chapter 1 describes the motivation of this dissertation, an overview of our proposed approach, and the contributions. Chapter 2 describes the necessary background. Chapter 3 describes our metamodel and a set of business patterns, and applies them to model the Quote-To-Cash scenario. Additionally, it presents our method for specifying and verifying business models, and describes its application on the Quote-To-Cash scenario. Chapter 4 describes Comma, our modeling methodology, and the developer study that compares Comma to RosettaNet. We evaluate our approach in the domain of healthcare in Chapter 5. Additionally, Chapter 5 presents the developer study that compares Comma with HL7. Chapter 6 presents a joint operational semantics for (achievement and maintenance) commitments and goals. In Chapter 7, we present hierarchical task network formalization of achievement commitments and goals. Finally, Chapter 8 discusses related literature and some future directions.
Chapter 2

Background

This chapter consolidates the salient background needed for understanding our approach.

2.1 Achievement Commitments

An achievement commitment expresses a social or organizational relationship between two agents. This sense of commitments was defined by Singh (1991) and adopted by Castelfranchi (1995): its key feature is that it relates one agent to another and thus contrasts with an agent’s commitment to its intention Bratman [1987]. Additional motivation and historical background on our view of commitments is available in Singh [2012]. Specifically, a commitment C(DEBTOR, CREDITOR, antecedent, consequent) denotes that the DEBTOR commits to the CREDITOR for bringing about the consequent if the antecedent becomes true [Singh, 1999].

Figure 2.1 shows the lifecycle of a commitment. A labeled rounded rectangle represents a commitment state, and a directed edge represents a transition; transitions are labeled with the corresponding action or event. A commitment can be in one of the following states: Null (before it is created), Conditional (when it is initially created), Expired (when its antecedent becomes forever false, while the commitment was Conditional), Satisfied (when its consequent is brought about while the commitment was Active, regardless of its antecedent), Violated (when its antecedent has been true but its consequent will forever be false, or if the commitment is cancelled when Detached), Terminated (when cancelled while Conditional or released while Active), or Pending (when suspended while Active). Active has two substates: Conditional (when its antecedent is false) and Detached (when its antecedent has held). A debtor may create, cancel, suspend, or reactivate a commitment; a creditor may release a debtor from a commitment.

2.1.1 Lifecycle of a Commitment

In detail, the lifecycle of a commitment as in Figure 2.1 is as follows:
• If the debtor creates a commitment, then the commitment becomes active; it becomes conditional or detached depending on the antecedent.

• The antecedent failure (timeout) of a conditional commitment causes it to expire.

• If the antecedent of a conditional commitment is brought about, then the commitment detaches.

• If the debtor suspends a conditional or detached commitment, then the commitment becomes pending.

• If the debtor reactivates a pending commitment, the commitment becomes active; moreover, if the antecedent holds, then the commitment becomes detached else conditional.

• If the debtor reactivates a pending commitment and the consequent holds, then the commitment is satisfied.

• If the debtor cancels a conditional commitment, then the commitment terminates.

• If the creditor releases the debtor from a conditional or detached commitment, then the commitment terminates.

• If the debtor cancels a detached commitment, or there is failure (timeout) of the consequent, then the commitment is violated.

Figure 2.1: Commitment lifecycle as a state transition diagram.
• If the debtor brings about the consequent of a conditional or detached commitment, then the commitment is satisfied.

We consider commitments whose antecedents and consequents are propositions. In mapping from informal examples, this means we would (1) parameterize the antecedent and consequent, and (2) instantiate such parameters when we instantiate such a commitment. For example, a seller’s offer to a buyer would be formalized as a commitment with parameters such as item description and price. By listing items for sale in a price catalogue, the seller creates multiple specific commitments, each for its specified item and price. The above life cycle applies to each such instantiated commitment. Such a commitment is detached when its antecedent comes to hold and satisfied when its consequent comes to hold.

Note that we specify the truth of a condition—our approach is neutral as to who brings about a condition—as opposed to the performance of an action by an agent. In general, focusing on conditions facilitates greater flexibility during enactment. Further, for some commitments, as in the payment example above, the creditor may adopt a goal to bring about the antecedent. However, in other commitments, the creditor may not. For example, an insurance company may commit to paying a car driver’s medical bills if the driver is injured in an accident: the driver would not ordinarily have a goal of getting injured! For these reasons, we keep the lifecycle of commitments general and enable agents in different settings to exercise that lifecycle in ways that best make sense to them in the appropriate context.

Second, note that, in general, commitments need not be symmetric, i.e., reciprocal. That is, in general, an agent may have a commitment to another agent without the latter having a converse commitment to the former agent. As an example, by accepting to review a paper, a conference committee member commits to a program chair to review that paper. The chair makes no converse commitment. Singh [2012] discusses such properties of commitments at length.

Third, note that we do not include penalties within a commitment but would capture them separately. In the literature on commitments, a penalty is customarily handled from the organizational context [Singh et al., 2009; Bulling and Dastani, 2011]. The organizational context refers to the organization within whose scope the given commitment arises [Singh, 1999]. A classical example is the eBay online marketplace within whose scope a buyer and a seller enter into commitments (namely, the seller to ship the goods in question and the buyer to pay the seller). If the buyer does not pay for an auction she won, then eBay can penalize her in various ways, including closing her account. Our chosen setting of autonomous agents contrasts with a conceptually centralized, regimented system wherein the “system” can prevent the violation of the applicable commitments (see the discussion in, e.g. Boella et al. [2008]). A pertinent example of a regimented system is the use of proxy agents in AMELI/ISLANDER [Sabater-Mir et al., 2007] that actually prevent agents from performing forbidden actions. In effect, the bad action never occurs and therefore no compensation or penalty is needed to undo or mitigate the effects of the bad action.
2.2 Achievement Goals

An achievement goal expresses a state of the world that an agent wishes to bring about. Our view of goals follows Thangarajah et al. [2011]. Goals differ from both desires and intentions. An agent’s desires represent the agent’s proattitudes [Rao and Georgeff, 1992]; an agent may concurrently hold mutually inconsistent desires. By contrast, it is customary to require that a rational agent believe its goals are mutually consistent [Winikoff et al., 2002]. An agent’s intentions are its adopted or activated goals.

Specifically, a goal \( G = G(x, p, q, s, f) \) of an agent \( x \) has a precondition (or context) \( p \) that must be true before \( G \) can become Active and some intention can be adopted to achieve it, and a post-condition (or effect) \( q \) that becomes true if \( G \) is successfully achieved [Winikoff et al., 2002]. The success condition \( s \) defines the success of \( G \), and the failure condition \( f \) defines its failure. A goal is successful if and only if \( s \) becomes true prior to \( f \): that is, the truth of \( s \) entails the satisfaction of the goal only if \( f \) does not intervene. Note that \( s \) and \( f \) should be mutually exclusive.

As for commitments, the success or failure of a goal depends only on the truth or falsity of the various conditions, not on which agent brings them about. The antecedent and consequent of a commitment are respective analogues of the precondition and post-condition of a goal. For example, if a buyer has a commitment to pay if the seller ships the items, then the buyer may consider a goal to pay with the precondition of the items being shipped.

Often, the post-condition \( q \) and the success condition \( s \) coincide, but they need not. Since the pre- and post-conditions of goals do not have a direct bearing on our semantics and are not relevant for the discussion (see also [Günay et al., 2012]), after this point we omit them and simply write \( G(x, s, f) \). In fact, one can see \( p \) as a guard on the activate and reinitialize transitions in the goal lifecycle, and can fold \( q \) into \( s \) by rewriting \( G(x, p, q, s, f) \) as \( G(x, s \land q, f) \).

Figure 2.2 simplifies Thangarajah et al.’s (2011) lifecycle of an achievement goal. A goal can be in one of the following states: Null, Inactive\(^2\), Active, Suspended, Satisfied, Terminated, or Failed. The last three collectively are terminal states: once a goal enters any of them, it stays there forever. Note how both commitment and goal lifecycles have Satisfied and Failed states, and also have Null and Active states.

Before its creation, a candidate goal is in state Null. Once considered by an agent (its “goal holder”), a goal commences as Inactive. Upon activation, the goal becomes Active; the agent may pursue its satisfaction by attempting to achieve \( s \). If \( s \) is achieved, the goal transitions to Satisfied. At any point, if the failure condition of the goal becomes true, the goal transitions to Failed. At any point, the goal may become Suspended, from which it may eventually return to an Inactive or Active state appropriately. Lastly, at any point the agent may drop or abort the goal, thereby moving it to the Terminated state. Note the difference between the drop and abort operations is that the former omits any clean-up that might be

---

\(^1\)We do not follow their inclusion of an in-condition that is true once a goal is Active until its achievement.

\(^2\)Renamed from Pending to avoid conflict with commitments nomenclature. Although further alignment of goal and commitment state names could be attempted, we have made the minimal change: we retain all other names unchanged to agree with the literature.
appropriate for plans part-way through execution for the goal—such as canceling an order made to a supplier—and simply discards the goal and any associated plans [Thangarajah et al., 2011].

2.2.1 Lifecycle of an Achievement Goal

In detail, the lifecycle of a goal as in Figure 2.2 is as follows:

- If an agent creates and considers a goal, then the goal becomes inactive.
- If an agent activates an inactive goal, then the goal becomes active.
- If an agent suspends an active or inactive goal, then the goal becomes suspended.
- If an agent reconsider a suspended goal, then the goal becomes inactive.
- If an agent reactivate a suspended goal, then the goal becomes active.
- If an agent drops or aborts an inactive, active, or suspended goal, then the goal terminates.
- If the agent begins to believe the failure condition of an inactive, active, or suspended goal, then the goal fails.
• If the agent begins to believe the satisfaction condition of an inactive, active, or suspended goal, then the goal is satisfied.

It is worth remarking on a subtle but important point. Although we represent commitment and goal lifecycles using the same notation, they are conceptually quite different in that a goal represents an element of a private state whereas a commitment represents an element of a social state. Thus a commitment comes into being in state Active the moment it is created (typically due to communications and based on the social norms in play) whereas for a goal, an agent may mull it over. In other words, a commitment is created via an atomic public event whereas a goal, being private, can be created Inactive and then transition to Active. We return to this point in Section 6.1.2.

We also remark that the careful analysis of states and transitions of a goal lifecycle, demanded by our coupled commitment—goal operational semantics, has the side benefit of clarifying minor points in the operational semantics for goals themselves. For example, in some cases the literature has ambiguity about the possibility of the simultaneous truthhood of the success and failure conditions of a goal, and its semantic implication (if permitted). Our semantics is explicit in disallowing states in which both the success and failure conditions of a goal simultaneously become true.

As discussed earlier, a conceptual relationship is established between a goal and a commitment when they reference each other’s objective conditions. Even when related in such a manner, however, a goal and a commitment independently progress in accordance with their respective lifecycles. For example, when agent \( y \) brings about condition \( s \)—an objective condition—a commitment \( C = C(x, y, s, u) \) detaches, and a goal \( G_{end} = G(x, s, f) \) is satisfied. That is, \( C \) has \( s \) as a condition (its antecedent) and \( G_{end} \) has \( s \) as a condition (its success condition), but \( C \) and \( G_{end} \) progress independently through their lifecycle when \( s \) comes about.

### 2.3 Maintenance Goals

We define a maintenance goal and its lifecycle based on Duff et al. [2006]. Let \( M = M(x, m, R, P, s, f) \) be agent \( x \)’s maintenance goal for condition \( m \). \( M \) persists until either its success condition \( s \) or its failure condition \( f \) become true. \( B_x \models \neg m \) means that \( x \) believes that \( m \) is false: thus, \( x \) adopts a recovery achievement goal \( R \). Let \( \pi_x \) capture \( x \)'s lookahead mechanism, independent of \( M \). Then, \( B_x \models \pi_x(\neg m) \) means that \( x \) believes that \( m \) will become false unless it acts appropriately: thus, \( x \) adopts a preventive achievement goal \( P \). Here \( x \) deliberates as normal to achieve \( R \) or \( P \). We omit the latter parameters of \( M \) unnecessary for exposition.

Fig. 2.3 depicts a state-based lifecycle for a maintenance goal \( M \). The state Waiting distinguishes maintenance from achievement goals. Once the agent considers it, \( M \) starts in the Inactive state; upon its activation, \( M \) transitions to the Waiting state. Here, the agent monitors the (predicted) truth status of condition \( m \). Should \( x \) believe that \( m \) is not true or will not remain true in the future, it transitions the
maintenance goal $M$ to Active, where $x$ creates and adopts goals $P$ or $R$ as explained above. $M$ remains Active until $m$ is restored or is no longer predicted to become false, whereupon $M$ returns to Waiting. Should $P$ or $R$ fail, $x$ retries them. If $x$ believes that it cannot prevent $\neg m$ or restore $m$, it may fail $M$. The failure condition $f$ could incorporate this requirement or include the failure of $P$ or $R$, i.e., $x$ tries once to prevent $\neg m$ or restore $m$, and fails $M$ if it cannot. We note that an agent so equipped may plan to find other ways to achieve the success condition of $P$ or $R$ [de Silva et al., 2009]. When $x$ transitions $M$ to the Suspended state, it no longer monitors $m$ and aborts any active achievement goals $P$ or $R$ generated by $M$ [Thangarajah et al., 2008].

2.3.1 Lifecycle of a Maintenance Goal

In detail, the lifecycle of a maintenance goal as in Figure 2.3 is as follows:

- If an agent considers a goal, then the goal becomes inactive.
- If an agent activates an inactive goal, then the goal becomes waiting.
- If an agent suspends a waiting goal, then the goal becomes suspended.
- If an agent reactivates a suspended goal, then the goal becomes waiting.
• If the agent believes that the maintenance condition of a waiting goal is not true or will not remain true in future, then the goal becomes active.

• If the agent restores the maintenance condition of an active goal, then the goal becomes waiting.

• If an agent drops or aborts an inactive, active, or suspended goal, then the goal terminates.

• If the agent begins to believe the failure condition of an inactive, active, or suspended goal, then the goal fails.

• If the agent begins to believe the satisfaction condition of an inactive, active, or suspended goal, then the goal is satisfied.

2.4 Computation Tree Logic

Computation Tree Logic (CTL) is a temporal logic that conceptualizes time as branching into the future [Clarke et al., 1999]. The different future branches or paths emanating from a state can indicate the possible ways in which the future may evolve. In our setting, the different paths will correspond to the combinations of choices made by the various agents.

The syntax of the CTL language in Backus-Naur Form is:

$$\phi ::= \bot | \top | p | (\neg \phi) | (\phi \land \phi) | (\phi \lor \phi) | (\phi \rightarrow \phi) | AX\phi | EX\phi | AF\phi | EF\phi | AG\phi | EG\phi | A[\phi U \psi] | E[\phi U \psi]$$

where $p$ is an atomic proposition, and $\neg, \land, \lor$ and $\rightarrow$ are logical operators for negation, conjunction, disjunction, and implication, respectively. The symbols $\bot$ and $\top$ mean false and true, respectively. Each temporal operator of CTL is a pair of symbols. The first symbol of this pair quantifies paths, and can be either $A$, meaning along all paths, or $E$, meaning along at least one path. The second symbol is a linear-time operator: it can be $X$, meaning in the next state, $G$, meaning in all future states, $F$, meaning in a current or future state, or $U\psi$, meaning until $\psi$ holds. Note that $U$ is strong, i.e., $\phi U \psi$ holding requires that $\psi$ does hold.

The semantics of CTL formulas is defined over a transition system $M = (S, \tau, L)$ in which $S$ is a set of states, $\tau \subseteq S \times S$ is a transition relation, and $L$ is a labeling function that maps a state to a valuation of propositions. A path $\pi$ in the model $M$ is an infinite sequence of states $\langle s_0, s_1, s_2, \ldots \rangle$ such that $\forall i \geq 0, (s_i, s_{i+1}) \in \tau$. We represent the $i^{th}$ state on a path $\pi$ as $\pi_i$, and the set of all paths emanating from a state $s$ as $\Pi(s)$. We write $M, s \models \phi$ to mean that a model $M$ satisfies a CTL formula $\phi$ at some state $s$, where $s \in S$, and define satisfaction using structural induction. The semantic postulates for CTL are given in Table 2.1.
Table 2.1: Semantic postulates of Computation Tree Logic [Clarke et al., 1999].

1. \( M, s \models \top \) and \( M, s \not\models \bot \)
2. \( M, s \models p \) iff \( p \in L(s) \)
3. \( M, s \models \neg \phi \) iff \( M, s \not\models \phi \)
4. \( M, s \models \phi_1 \land \phi_2 \) iff \( M, s \models \phi_1 \) and \( M, s \models \phi_2 \)
5. \( M, s \models \phi_1 \lor \phi_2 \) iff \( M, s \models \phi_1 \) or \( M, s \models \phi_2 \)
6. \( M, s \models \phi_1 \rightarrow \phi_2 \) iff \( M, s \not\models \phi_1 \) or \( M, s \models \phi_2 \)
7. \( M, s \models \Box \phi \) iff \( \forall \pi \in \Pi(s), M, \pi_1 \models \phi \)
8. \( M, s \models \Diamond \phi \) iff \( \exists \pi \in \Pi(s), M, \pi_1 \models \phi \)
9. \( M, s \models \Box \phi \) iff \( \forall \pi \in \Pi(s), \forall i \geq 0, M, \pi_i \models \phi \)
10. \( M, s \models \Diamond \phi \) iff \( \exists \pi \in \Pi(s), \forall i \geq 0, M, \pi_i \models \phi \)
11. \( M, s \models A [\phi_1 U \phi_2] \) iff \( \forall \pi \in \Pi(s), \exists i \geq 0, M, \pi_i \models \phi_2, \forall j < i, M, \pi_j \models \phi_1 \)
12. \( M, s \models E [\phi_1 U \phi_2] \) iff \( \exists \pi \in \Pi(s), \exists i \geq 0, M, \pi_i \models \phi_2, \forall j < i, M, \pi_j \models \phi_1 \)

2.5 NuSMV Model Checking Tool

Model checking is a formal verification technique to determine whether a model satisfies certain desired properties. NuSMV [NuSMV, 2012] is a contemporary, well-known tool that implements model checking [Clarke et al., 1999]. NuSMV supports CTL for specifying the properties to be verified. We now describe the features of the NuSMV input language that we will use to construct a computational model.

- **VAR**: declares a model variable. We employ variables of two types: Boolean and enumeration. For each transition from the goal lifecycle and the commitment lifecycle, the model contains a Boolean variable. An enumeration variable models the state of a goal or a commitment. For example,

```plaintext
VAR
    create: boolean;
    status: {NULL, CONDITIONAL, DETACHED, EXPIRED,
            PENDING, TERMINATED, VIOLATED, SATISFIED});
```
declares `create` as a Boolean variable, which represents the `create` transition of a commitment from NULL state to CONDITIONAL state, and `status` as an enumeration variable, which models the status of a commitment.

- **INIT**: declares initial values of the model variables. For example,
  
  ```
  INIT
  create = FALSE & status = NULL;
  ```
  
  assigns initial values of FALSE and NULL to `create` and `status`, respectively.

- **INVAR**: declares an invariant, i.e., a Boolean constraint that is TRUE in each state on all the paths that NuSMV considers. For example,
  
  ```
  INVAR
  cancel = FALSE & release = FALSE;
  ```
  
  constrains the model paths to only those in which in all states `cancel` and `release` are FALSE.

- **MODULE**: declares a reusable unit containing various NuSMV constructs. For example,
  
  ```
  MODULE goal()
  VAR
  status: {NULL, INACTIVE, ACTIVE, SUSPENDED, FAILED, TERMINATED, SATISFIED};
  ```
  
  declares a goal module containing a `status` variable.

  Then we can make use of the goal module by writing, for example,
  
  ```
  MODULE main()
  VAR
  g: goal;
  ```
  
  to specify a main module that declares a variable `g` of kind goal.

- **ASSIGN**: declares assignment of a value to a model variable. For example,
  
  ```
  ASSIGN
  next(status) :=
  case
  status = NULL & consider : INACTIVE;
  status = CONDITIONAL & ant & con : {DETACHED, SATISFIED};
  TRUE : status;
  esac;
  ```
assigns the value of status variable in the next state using a case block with two clauses. The first clause assigns status the value of INACTIVE in the next state, provided that in the current state status is NULL, and consider is TRUE. The second clause assigns status the value of either DETACHED or SATISFIED, provided that in the current state status is CONDITIONAL, and ant and con are TRUE. Effectively, this clause produces two execution branches from the current state: on one branch the clause sets status as DETACHED, and on the second branch the clause sets status as SATISFIED. The third clause is a default clause that assigns the value of status in the next state to be the value of status in the current state.

• CTLSPEC: declares a CTL specification that NuSMV verifies. For example,

\[
\text{CTLSPEC} \quad \begin{align*}
\text{AG}( & g\text{-end.g.status} = \text{ACTIVE} \land c\text{-status} = \text{NULL} \rightarrow \\
& \text{AF c\text{-status} = CONDITIONAL});
\end{align*}
\]

declares a CTL specification: on all paths globally, in some state \( s \) if the end goal status \( (g\text{-end.g.status}) \) is ACTIVE and the commitment status \( (c\text{-status}) \) is NULL, then on all paths emanating from \( s \) in future the commitment status will be CONDITIONAL.

• FAIRNESS: declares a Boolean constraint that is true infinitely often on all paths that NuSMV considers. For example,

\[
\text{FAIRNESS} \quad \begin{align*}
g\text{-status} \neq \text{NULL} \land g\text{-status} \neq \text{INACTIVE} \land g\text{-status} \neq \\
\text{ACTIVE};
\end{align*}
\]

declares a fairness constraint that means that NuSMV considers only those paths on which the status of goal \( g \) is not infinitely often null, inactive, or active. We employ a fairness constraint to encode the assumption of some of our theorems that no infinite sequence of a certain type of states occurs.

2.6 Classical and HTN Planning

2.6.1 Logic Language

We use a first-order logic language consisting of an infinite set of symbols for predicates, constants, functions, and variables, obeying the usual formation rules of first-order logic and following its usual semantics when describing planning domains [Ghallab et al., 2004].

**Definition 1 (Term).** A term, denoted generically as \( \tau \), is a variable \( w, x, y, z \) (with or without subscripts); a constant \( a, b, c \) (with or without subscripts); or a function term \( f(\tau_0, \ldots, \tau_n) \), where \( f \) is a \( n \)-ary function symbol applied to (possibly nested) terms \( \tau_0, \ldots, \tau_n \).
Definition 2 (Atomic formula). A (first-order) atomic formula, denoted as $\varphi$, is a construct of the form $p(\tau_0, \ldots, \tau_n)$, where $p$ is an $n$-ary predicate symbol and $\tau_0, \ldots, \tau_n$ are terms. A first-order formula $\Phi$ is recursively defined as $\Phi := \Phi \wedge \Phi' | \neg \Phi | \varphi$.

We assume the usual abbreviations: $\Phi \lor \Phi'$ stands for $\neg(\neg \Phi \land \neg \Phi')$; $\Phi \rightarrow \Phi'$ stands for $\neg \Phi \lor \Phi'$ and $\Phi \leftrightarrow \Phi'$ stands for $(\Phi \rightarrow \Phi') \land (\Phi' \rightarrow \Phi)$. Additionally, we also adopt the equivalence $\{\Phi_1, \ldots, \Phi_n\} \equiv (\Phi_1 \land \cdots \land \Phi_n)$ and use these interchangeably. Our mechanisms use first-order unification [Apt, 1997], which is based on the concept of substitutions.

Definition 3 (Substitution). A substitution $\sigma$ is a finite and possibly empty set of pairs $\{x_1/\tau_1, \ldots, x_n/\tau_n\}$, where $x_1, \ldots, x_n$ are distinct variables and each $\tau_i$ is a term such that $\tau_i \neq x_i$.

Given an expression $E$ and a substitution $\sigma = \{x_1/\tau_1, \ldots, x_n/\tau_n\}$, we use $E\sigma$ to denote the expression obtained from $E$ by simultaneously replacing each occurrence of $x_i$ in $E$ with $\tau_i$ for all $i \in \{1, \ldots, n\}$. Unifications can be composed; that is, for any substitutions $\sigma_1 = \{x_1/\tau_1, \ldots, x_n/\tau_n\}$ and $\sigma_2 = \{y_1/\tau'_1, \ldots, y_k/\tau'_k\}$, their composition, denoted as $\sigma_1 \cdot \sigma_2$, is defined as $\{x_1/(\tau_1 \cdot \sigma_2), \ldots, x_n/(\tau_n \cdot \sigma_2), z_1/(z_1 \cdot \sigma_2), \ldots, z_m/(z_m \cdot \sigma_2)\}$, where $\{z_1, \ldots, z_m\}$ are those variables in $\{y_1, \ldots, y_k\}$ that are not in $\{x_1, \ldots, x_n\}$. A substitution $\sigma$ is a unifier of two terms $\tau_1, \tau_2$, if $\tau_1 \cdot \sigma = \tau_2 \cdot \sigma$.

Definition 4 (Unify Relation). Relation $\text{unify}(\tau_1, \tau_2, \sigma)$ holds iff $\tau_1 \cdot \sigma = \tau_2 \cdot \sigma$. Moreover, $\text{unify}(p(\tau_0, \ldots, \tau_n), p(\tau'_0, \ldots, \tau'_n), \sigma)$ holds iff $\text{unify}(\tau_i, \tau'_i, \sigma)$, for all $0 \leq i \leq n$.

Thus, two terms $\tau_1, \tau_2$ are related through the unify relation if there is a substitution $\sigma$ that makes the terms syntactically equal. In our representation and algorithms, we adopt Prolog’s convention [Apt, 1997] and use strings starting with a capital letter to represent variables and strings starting with a small letter to represent constants.

2.6.2 Classical Planning

STRIPS-style planning defines a problem in terms of an initial state and a goal state—both specified as sets of ground atoms—and a set of operators. An operator has a precondition encoding the conditions under which the operator can be used, and a postcondition encoding the outcome of applying the operator. Planning is concerned with sequencing actions obtained by instantiating operators describing state transformations. In our representation, an operator $o$ is a five-tuple $(\text{name}(o), \text{pre}(o), \text{del}(o), \text{add}(o), \text{cost}(o))$, where (1) $\text{name}(o) = \text{act}(\vec{x})$, the name of the operator, is a symbol followed by a vector of distinct variables such that all free variables in $\text{pre}(o)$, $\text{del}(o)$, and $\text{add}(o)$ also occur in $\text{act}(\vec{x})$; (2) $\text{pre}(o)$, $\text{del}(o)$ and $\text{add}(o)$ called, respectively, the precondition, delete-list and add-list, are sets of atoms where the add and delete lists are disjoint; and (3) $\text{cost}(o)$ is a numeric expression representing the cost of executing an operator. The delete-list specifies which atoms should be removed from the state of the world when the operator is applied, and the add-list specifies which atoms should be added to the state of the world when the operator is applied.
2.6.3 Hierarchical Task Network (HTN) Planning

A Hierarchical Task Network (HTN) planner generates a plan by successive refinements of sets of tasks. Tasks are classified into primitive (à la individual operators in STRIPS-style planning) and compound (abstract high-level tasks). An HTN planner recursively decomposes compound tasks by using a designer-specified library of methods until only primitive tasks remain. Methods are elements of domain knowledge that describe how a higher-level task can be decomposed into more primitive tasks, they constrain the search space making HTN planning more efficient. For example, in a purchase scenario, the customer’s higher-level task, achieveGoal, to achieve its goal for procuring the goods could be decomposed into primitive tasks of creating a commitment, \( C(\text{CUSTOMER, MERCHANT, goods, pay}) \), toward the merchant, and of paying the merchant after the merchant provides the goods.

Formally Ghallab et al. [2004], an HTN planning problem \( \mathcal{P} \) is a tuple \((d, I, D)\), where (1) \( d \) is a task network, (2) \( I \) is an initial state, and (3) \( D \) is an HTN planning domain. The planning domain \( D \) is a tuple \((A, M)\), respectively, finite sets of operators and methods. A task network \( \mathcal{H} \) is a tuple \((T, C)\), where \( T \) is a finite set of tasks (primitive and compound), and \( C \) is a set of partial ordering constraints on tasks in \( T \). A constraint specifies the order in which certain tasks can be executed, and can be either a precedes or a succeeds relation. For example, \( t_i \prec t_j \) means that \( t_i \) must be executed before \( t_j \). A task has a precondition that must hold under some substitution before the task can be executed. Corresponding to each primitive task \( t \), an operator exists in the planning domain, that is, \( A \subseteq T \). The operator specifies the effect on the world state if the task is executed. Corresponding to each compound task \( t \), a method exists in the planning domain, that is, \( M \subseteq T \). A method \( m \) is a tuple \((t, s, \mathcal{H}')\), where \( s \) is a precondition that must hold for a task \( t \) to be refined into another task network \( \mathcal{H}' = (T', C') \). Note that \( A \cup M = T \): all operators in \( A \) and all methods in \( M \) are in the task set \( T \).

2.7 UML 2.0: Sequence Diagrams

We specify an operational model as a set of UML 2.0 sequence diagrams. The UML 2.0 specification [Obj, 2004] defines the sequence diagram to model the behavior of a system. A sequence diagram represents each participant as a vertical line. A directed horizontal line between any two participants represents a message. The ordering of the horizontal lines gives the temporal order of the messages. To modularize large and complex interactions, the specification further defines operators \textit{opt}, \textit{alt}, and \textit{loop}.

Figure 2.4 shows a simple sequence diagram and the corresponding NuSMV encoding. \( R1 \) and \( R2 \) are the participants that exchange messages \( m_1 – m_n \) in that order. We map all such sequence diagrams to a NuSMV model. As an example, the NuSMV model for the sequence diagram in Figure 2.4 contains \( n \) Boolean state variables: \( m_1 – m_n \). In the initial state, \( m_1 – m_n \) are all false, that is, in the initial state, none of the messages are exchanged. The model contains a TRANS constraint that allows transitions from a state in which (a) \( m_i \) is false, \( 1 \leq i \leq n \), (a message \( m_i \) is not exchanged), and (b) \( m_j \) is true, \( 1 \leq j < i \),
The opt(ion) operator is the simplest sequence diagram operator. Figure 2.5 shows a sequence diagram with two participants $R_1$ and $R_2$, and an opt(ion) operator. The option operator has one message fragment with messages $m_1$–$m_n$. If the guard $g$ holds, then $R_1$ and $R_2$ exchange the messages. The corresponding NuSMV TRANS constraint allows transitions from a state in which: (a) the guard $g$ is true, (b) $m_i$ is false, $1 \leq i \leq n$, (a message $m_i$ is not exchanged), and (c) $m_j$ is true, $1 \leq j < i$, (all messages prior to $m_i$ are exchanged), to a state in which $m_i$ is true.

**2.8 RosettaNet**

RosettaNet [2008] is a leading industry effort, that creates standards for business-to-business integration. The RosettaNet consortium consists of over 500 organizations of various sizes, and from various industry sectors including electronics manufacturing from which domain RosettaNet began. These organizations use elements of the RosettaNet standard, named Partner Interface Processes (PIPs), to transact business that is worth billions of dollars. A PIP specifies two-party interactions for some specific business purpose. For example, a buyer requests a quote from a seller using PIP 3A1, and a seller requests financing from a financing processor, on behalf of the buyer, using PIP 3C2. A PIP specification includes a natural language document that informally describes the purpose of the PIP, any underlying assumptions, the intended outcome of executing the PIP, and the message structures as XML DTD or XML Schema.

The RosettaNet standard specifies over 100 PIPs for various business processes in the eCommerce supply chain. The standard classifies the PIPs using clusters and segments. A cluster represents a major business process of the supply chain and comprises segments that represent subprocesses of the cluster’s
business process. Each segment contains many PIP specifications. For example, Cluster 3 represents the Order Management process. Segment A of Cluster 3 represents the subprocess Quote and Order Entry. Segment A contains PIPs such as for Request Quote (3A1), Request Shopping Cart Transfer (3A3), and Request Purchase Order (3A4). RosettaNet PIPs are commonly identified through their code names such as those in parentheses above.

2.9 HL7

Health Level Seven (HL7) refers to messaging and allied standards for healthcare geared toward improving patient care and optimizing workflows by reducing ambiguity. HL7 specifies messages and trigger events to support communication. Each message has a type, which describes its purpose. For example, the ADT (Admit Discharge Transfer) type includes several messages dealing with patient admission, discharge, and transfer. HL7 specifies two categories of message types: request response, e.g., ADT followed by an ACK (Acknowledgment), and one-sided, e.g., ORM (Order Message) and RXO (Pharmacy/Treatment Order). A trigger event is initiated when there is a need for dataflows between the systems. For example, A01 (Admit) triggers a patient’s admission or visit notification. The event can cause one or more messages to be exchanged. The event can trigger additional associated events. For example, A02 (Transfer) may be followed by A08 (Update Patient Information).
Chapter 3

Metamodel, Patterns, and Verification

This chapter describes our metamodel and the reusable patterns.

3.1 Metamodel

We concern ourselves with business models that involve two or more participants. The business partners, abstracted as roles, participate in a business relationship. The participants create, manipulate, and satisfy commitments in each relationship. They execute tasks for each other that enable them to achieve their respective goals.

![Diagram of a commitment-based business metamodel.]

Figure 3.1: A commitment-based business metamodel.
Figure 3.1 describes our business metamodel. The following are the concepts that occur in it.

**Agent:** a computational representation of a business partner. An agent has goals, and executes business tasks. For each business relationship in which an agent participates, it enacts one or more roles.

**Role:** an abstraction over agents that helps specify a business relationship in templatic form. Each role specifies the commitments expected of the agents who play that role along with the tasks they must execute to function in that role.

**Goal:** a state of the world that an agent desires to be brought about. van Riemsdijk et al. [2008] define this as an achievement goal. An agent achieves a goal by executing appropriate tasks or negotiating with other agents to have them execute appropriate tasks.

**Task:** a business activity viewed from the perspective of an agent.

**Commitment:** an element of a contractual business relationship. A commitment $C(\text{DEBTOR}, \text{CREDITOR}, \text{antecedent}, \text{consequent})$ denotes that the DEBTOR commits to the CREDITOR to bring about the consequent if the antecedent begins to hold [Singh, 1999]. A commitment when active functions as a directed obligation from a debtor to a creditor. However, unlike a traditional obligation, a commitment may be manipulated, e.g., delegated, assigned, or released.

**Business relationship:** a set of interrelated commitments among two or more roles that describe how they carry out the given business process.

### 3.2 Business Model Patterns

A business pattern is a recipe for modeling recurring business scenarios in terms of manipulations of commitments. This section describes a key set of such patterns [Telang and Singh, 2009], which could seed a potential business model pattern library. Section 3.3.1 demonstrates the effectiveness of this simple set of patterns on the Quote-To-Cash business process.

We use the attributes *name, intent, motivation, implementation, and consequences* to describe our patterns [Gamma et al., 1995]. Here the *consequences* of a pattern allude to the practical consequences of applying the pattern, i.e., the assumptions underlying the model.

We express a pattern in terms of the commitments, where each commitment references roles and tasks. The patterns would be instantiated by the agents who adopt the specified roles. A set of commitments fully specify a pattern. A pattern diagram illustrates the progression of the commitments in a typical execution. The diagrams use the notation of Figure 3.1, and additionally show two directed edges for each commitment: from the debtor to the commitment and from the commitment to the creditor. The subscript on a commitment indicates its state: $C$ for conditional, $D$ for detached, $S$ for satisfied, and $P$ for pending.
The expired and violated states do not appear in the pattern diagrams since the diagrams illustrate a normal execution, that is, an execution in which none of the commitments expire or are violated.

### 3.2.1 Conditional Offer Pattern

This is the simplest possible pattern. It merely views a commitment (as described in Figure 2.1) as an offer.

![Conditional offer pattern diagram](image)

**Intent:** A proposer conditionally offers to execute a task for a client.

**Motivation:** For example, a conference committee member commits to a program chair to review a paper that the program chair requests the member to review. The chair makes no converse commitment.

**Implementation:** A commitment is created in which the proposer is the debtor, the client is the creditor, the consequent is the task that the proposer executes, and the antecedent is a condition that brings the commitment into force. Figure 3.2 shows this pattern.

**Consequences:** This presumes a benefit to the proposer from the antecedent of the commitment.

### 3.2.2 Commercial Transaction

**Intent:** This pattern expresses a value exchange between two trading partners. The trading partners negotiate and, upon agreeing, commit to executing certain tasks for each other.

**Motivation:** A typical situation would be where a seller and a buyer agree to exchange goods for payment. Similarly, if two parties barter goods or services that would fit in the same pattern.

**Implementation:** A pair of reciprocal commitments between the trading partners treated symmetrically specify the pattern. Figure 3.3 shows this pattern.
Consequences: In general, the antecedents and consequents of the commitments are composite expressions. Importantly, we need a mechanism to ensure progress by in essence breaking the symmetry, e.g., via a form of concession [Yolum and Singh, 2007].

3.2.3 Outsourcing

Intent: An outsourcer delegates a task to a subcontractor, typically because the outsourcer lacks the necessary capabilities or expects some other benefit, such as reduced costs or a lower risk of failure.

Motivation: Many business organizations outsource noncore activities. As an example, consider a customer who signs up for cable television service. The cable operator commits to the customer for installation. Instead of staffing its entire service area directly, the cable operator outsources the installation task to its local partners in various regions.

Implementation: The outsourcer has a commitment $C_1$ towards its client to execute a task. The outsourcer and the contractor negotiate, and agree that the contractor will create the commitment $C_2$ to execute the task if the outsourcer pays. Conversely, the outsourcer commits to pay the contractor if the contractor creates $C_2$. Note that the antecedent of this commitment is true ($\top$), which means that it is unconditional. We say that the commitment $C_2$ covers the commitment $C_1$. Eventually when the contractor creates $C_2$ the original commitment becomes pending. Figure 3.4 shows this pattern.

Consequences: The commitment from the outsourcer is pending and must either be discharged or reactivated depending on how the contractor performs.

3.2.4 Standing Service Contract

Intent: A provider sells a long-lived service to a consumer. The service can be bounded by a combination of duration and number of requests.
Motivation: A business service such as plumbing maintenance refers to (potentially) numerous service instances. Whenever the faucet leaks (within specified limitations), the plumber will fix it.

Implementation: The provider and consumer negotiate, and upon agreement, create a pair of commitments $C_1$ (the consumer commits to paying the service provider, if the service provider commits to provide service), and $C_2$ (the converse of $C_1$). Commitment $C_3$ is the standing service contract. In $C_3$, the provider commits to the consumer to serve each request prior to expiration up to a fixed number of requests. Figure 3.5 shows this pattern.

Consequences: Each service request should take a bounded effort.

3.3 Applying the Patterns

Our patterns capture recurring business scenarios in a natural manner. A business modeler analyzes a scenario description to identify its agents (e.g., FedEx), tasks (e.g., shipping), and business-specific roles (e.g., Shipper). The modeler develops the business model by successively applying the relevant patterns,
identifying each applicable patterns based on its specified intent. To apply each pattern, the modeler associates each business role (e.g., Shipper) with the appropriate pattern role (e.g., Contractor) and the tasks with the antecedents and consequents of the relevant commitments.

The patterns compose naturally when the business roles referenced by the patterns overlap. Our graphical representation emphasizes the composition by showing one node for each business role, thus highlighting the interrelationships among the commitments from the composed patterns.

### 3.3.1 Quote-To-Cash Process

This section evaluates our proposed methodology on the Quote-To-Cash business process loosely based on the public descriptions of Cisco System’s Quote-To-Cash implementation. Cisco is in the business of selling networking products and services. The Quote-To-Cash process encompasses all of the key business activities that begin from a customer requesting a quote, and end in Cisco receiving cash from the customer.

Some of the key partners in this process are customers, resellers, distributors, logistics providers, banks, contract manufacturers, and service providers. These participants engage in a number of complex interactions for transacting business. The resellers and the distributors serve as sales channels. A customer purchases goods either directly from Cisco, or from a reseller. In addition to selling the goods, a reseller also provides value added services of installing and configuring the goods. A reseller purchases goods either from a distributor, or from Cisco. A distributor always purchases goods from Cisco. Unlike a reseller, a distributor may purchase goods, and stock the goods in its warehouse. To build its products,
Cisco utilizes a set of contract manufacturers, and for shipping a set of transportation providers. The participants use different banks and credit companies for making payments.

Next we apply our approach to a fragment of the Quote-To-Cash process. To satisfy a business goal, a customer desires to install Cisco goods. The customer selects a reseller of Cisco for the purchase. The commercial transaction pattern models this scenario. The customer and the reseller negotiate the purchase price. Upon agreement, they create commitments \( C_1, C_2, \) and \( C_3 \), as Figure 3.6 shows. The customer commits \((C_1)\) to the reseller to pay if the reseller ships and installs the goods, and the reseller commits to the customer to ship \((C_2)\) and install \((C_3)\) the goods if the customer pays. Note that we model the commitment to ship \((C_2)\), and the commitment to install \((C_3)\) as separate commitments since the reseller outsources the shipping \((C_3)\) to a distributor, but installs the goods by itself \((C_2)\).

The reseller outsources to a distributor its commitment \( C_3 \) to the customer to ship the goods. Figure 3.7 applies the outsourcing pattern to this scenario. The reseller and the distributor negotiate, and upon agreement, create \( C_4 \) and \( C_5 \). The reseller commits \((C_4)\) to the distributor to pay if the distributor commits to \((C_6)\) ship the goods to the customer. The distributor conversely commits \((C_5)\) to create \( C_6 \) if the reseller pays the distributor. When the distributor creates \( C_6 \), \( C_5 \) discharges, \( C_4 \) detaches, and \( C_3 \) transitions to the pending state. The reseller may pay \((\text{payD})\) the distributor either before or after the distributor commits \((C_6)\) to shipping the goods to the customer. The figure shows a possible progression in which the reseller pays the distributor at a later time.

Figure 3.8 shows another scenario modeled by the outsourcing pattern. If the distributor has the goods in stock, it outsources the shipping of the goods to a shipper. The distributor commits \((C_7)\) to the shipper to pay if the shipper commits \((C_9)\) to ship the goods to the customer. The shipper conversely commits \((C_8)\) to create \( C_9 \) if the distributor pays the shipper. The shipper satisfies \( C_8 \) by creating \( C_9 \), it detaches \( C_7 \), and \( C_6 \) transitions to the pending state.
If the distributor lacks the stock of goods that it needs to ship to the customer to satisfy $C_9$, the distributor purchases the goods from Cisco. Figure 3.9 shows how the commercial transaction pattern models this purchase. Upon successful negotiation, the distributor and Cisco create the reciprocal commitments $C_{10}$ and $C_{11}$. The distributor commits ($C_{10}$) to Cisco to pay if Cisco ships the goods to the distributor. Conversely, Cisco commits ($C_{11}$) to the distributor to ship the goods if the distributor pays Cisco.

In Figure 3.10, Cisco outsources the shipping of the goods to the distributor, to a shipper. The outsourcing pattern models this scenario, and the figure shows the commitments that are created.

Figure 3.11 shows how service contract pattern applies to the scenario in which the customer buys a service contract from Cisco. The customer commits ($C_{15}$) to Cisco to pay if Cisco creates the service commitment $C_{17}$. Cisco commits to the customer to create the service commitment $C_{17}$ if the customer pays Cisco. Here $C_{17}$ means that Cisco commits to the customer to provide service on the goods if the customer requests the service prior to its expiration. The customer can request the service multiple times. To satisfy $C_{17}$, Cisco needs to provide service for each of those requests as long as the request is made.
prior to $C_{17}$’s expiration.

The above exercise shows how one can naturally construct business model for a scenario by applying the patterns. Figure 3.12 combines Figures 3.6, 3.7, 3.8, 3.9, 3.10, and 3.11 to show the complete model of Cisco’s QTC process. Note that commitments $C_1$ through $C_{17}$ fully specify the model. Figure 3.12 shows the status of each commitment in a possible progression of this model.

### 3.4 Verifying Business Models

We seek to build tools to verify if an operational model correctly supports a business model. Leading methods for verification include model checking, theorem proving, and manual testing. Theorem proving in general cannot be fully automated, and exhaustive test coverage with the manual testing is practically
Figure 3.9: Commercial Transaction: Distributor buys goods from Cisco.

C\textsubscript{10} \quad C(\textsc{distributor, cisco, shipGoodsD, payX})
C\textsubscript{11} \quad C(\textsc{cisco, distributor, payX, shipGoodsD})

infeasible. Model checking provides a happy middle since it automatically and exhaustively verifies if a model satisfies a stated property.

Figure 3.13 shows the main components of our verification approach. We map a business model to a temporal logic specification regarding the progression of the states of the relevant commitments. As explained above, we capture a business model as an aggregation of business patterns. We can map each pattern to a CTL specification, and can compose a CTL specification for a business model based on the specifications for the patterns that the model aggregates. UML 2.0 sequence diagrams capture operational interactions, which we map to FSMs specified in the NuSMV input language (as described in the appendix).

### 3.5 Patterns Formalized

We formalize a business model pattern as a set of CTL specifications. These specifications capture the essence of the pattern in terms of how the commitments evolve during the business execution. The subsections below formalize each pattern from Section 3.2.

#### 3.5.1 Conditional Offer Pattern Formalization

In a business execution, each commitment must evolve as per the life cycle that the Figure 2.1 shows. For each of the seven states in the commitment life cycle, we develop a CTL formula that specifies all legal transitions from that state. These exhaustively cover all possible transitions from the life cycle.

1. If a commitment is inactive in the current state, then in the next state it may be inactive, active, or detached.

\[ AG(inactive \rightarrow AX(inactive \vee active \vee detached)) \]
2. If a commitment is active in the current state, then in the next state it may be active, expired, satisfied, detached, or terminated.

\[ \text{AG(} \text{active} \rightarrow \text{AX(} \text{active} \lor \text{expired} \lor \text{satisfied} \lor \text{detached} \lor \text{terminated} \text{)} \) \]

3. If a commitment is detached in the current state, then in the next state it may be detached, satisfied, violated, or terminated.

\[ \text{AG(} \text{detached} \rightarrow \text{AX(} \text{detached} \lor \text{satisfied} \lor \text{violated} \lor \text{terminated} \text{)} \) \]

4. If a commitment is pending in the current state, then in the next state it may be pending, active, or detached.
Figure 3.11: Service Contract: Customer buys service contract from Cisco.

AG(pending →
AX(pending ∨ active ∨ detached))

5. A satisfied commitment remains satisfied.
AG(satisfied → AX(satisfied))

6. An expired commitment remains expired.
AG(expired → AX(expired))

7. A violated commitment remains violated.
AG(violated → AX(violated))

8. A terminated commitment remains terminated.
AG(terminated → AX(terminated))

Further, for business compliance, unless the creditor releases the debtor, the debtor must satisfy a detached commitment.

9. AG(detached → AF(satisfied ∨ terminated))

Listing 3.1 shows a fragment of the NuSMV module for this pattern.
Figure 3.12: The Quote-To-Cash business process expressed as a business model.

Listing 3.1: NuSMV module for the offer pattern.

MODULE commitment(create, antecedent,
consequent, suspend, reactivate, release,
cancel, act_timeout, det_timeout)

CONSTANTS INACTIVE,ACTIVE,EXPIRED,PENDING,
DETACHED,SATISFIED,VIOLATED,TERMINATED;

DEFINE
status :=

case
!create & !antecedent & !consequent &
!suspend & !reactivate & !act_timeout &
!release & !cancel & !det_timeout:INACTIVE;
create & !antecedent & !consequent &
Figure 3.13: Our approach in conceptual terms.


create & antecedent & !consequent & !release & ((!suspend & !reactivate) | (suspend & reactivate)) & !act_timeout & (cancel | det_timeout):VIOLATED;

create & consequent & !release & !act_timeout & !det_timeout:SATISFIED;

create & !consequent & suspend &
We use the MODULE declaration to specify the pattern in a reusable manner. Line 1 declares the module with parameters of create, antecedent, consequent, active timeout, and detached timeout. Lines 5–6 use the CONSTANTS keyword to declare the commitment states as symbolic constants. Lines 8–47 compute the state of the commitment based on the input parameters. Lines 49–51 use the CTLSPEC keyword to declare CTL specifications. Note that this is NuSMV encoding of the CTL specification 1 from the above list.

Next, we demonstrate how verification works. Consider the commitment $C(seller, buyer, pay, goods)$: a seller commits to a buyer to shipping the goods if the buyer pays. Figure 3.14 shows possible executions of an operational model. A circle depicts a state, and its label shows the status of the commitment. Above each state, the figure shows the action or event that holds in that state. Further, it labels the paths that satisfy or violate the conditional offer CTL specifications above. For example, Path 1

Figure 3.14: Example: Verifying the conditional offer pattern.
violates specification 3 from above since the commitment transitions from detached to expired state.

3.5.2 Commercial Transaction Pattern Formalization

The commercial transaction pattern consists of a pair of reciprocal commitments as Figure 3.3 shows. The CTL specifications of the conditional offer pattern individually apply to each of the commitment.

In this pattern, if one commitment is satisfied, it detaches the other commitment, and vice versa. The CTL specifications that specify this aspect are:

1. \( AG(\text{DETACHED}_1 \leftrightarrow \text{SATISFIED}_2) \)
2. \( AG(\text{DETACHED}_2 \leftrightarrow \text{SATISFIED}_1) \)

3.5.3 Outsourcing Pattern Formalization

Figure 3.4 shows the four commitments that are part of the outsourcing pattern. Among these, \( C_3 \) and \( C_4 \) are the reciprocal commitments between the outsourcer and the contractor. The CTL specifications of the commercial transaction pattern apply to these commitments, and the CTL specifications of the conditional offer pattern individually apply to commitments \( C_1 \) and \( C_2 \).

The CTL specifications unique to this pattern are:

1. If the outsourcer updates the commitment \( C_1 \) to be pending, then on all paths the contractor must eventually create the outsourced commitment \( C_2 \).
   \( AG(\text{PENDING}_1 \rightarrow AF(\neg \text{INACTIVE}_2)) \)
2. If the contractor satisfies \( C_2 \), then it satisfies the outsourcer’s commitment \( C_1 \).
   \( AG(\text{SATISFIED}_2 \rightarrow \text{SATISFIED}_1) \)
3. If the contractor violates the commitment \( C_2 \), then the outsourcer’s commitment \( C_1 \) is reactivated.
   \( AG(\text{VIOLATED}_2 \rightarrow AF(\neg \text{PENDING}_1)) \)

3.5.4 Service Contract Pattern Formalization

Figure 3.5 shows the service contract pattern consisting of three commitments. Of these, the CTL specifications of the commercial transaction pattern apply to \( C_1 \) and \( C_2 \), which are reciprocal commitments between the provider and the consumer.

The standing service commitment \( C_3 \) applies to multiple service request instances. If the consumer sends a service request prior to its expiration, and if the request count is smaller than the upper bound, then the provider creates a new detached commitment to service that request. The provider creates a new commitment for each service request. The CTL specifications of the offer commitment pattern apply to each of these commitments.
3.6 Applying Verification

This section applies the verification approach to the Quote-To-Cash business process introduced in Section 3.3.1. Figures 3.15, A.4, and A.5 show the sequence diagrams that model the operational interactions between the Quote-To-Cash participants. The participant roles shown as lifelines on these diagrams map to the model roles from Figure 3.12.

Figure 3.15: Operational interactions in a Quote-To-Cash implementation.

Figure 3.15(a) captures the quoting interactions. The customer requests a quote from the reseller to
The reseller requests a quote from the distributor for the goods. Upon receiving a quote from the distributor, the reseller sends a quote to the customer. The customer responds, and either accepts the quote or requests for a new quote with an additional discount. The reseller responds similarly to the distributor. The quote and the response interactions continue until either the customer accepts the quote, or the iteration count exceeds five. We notate the meaning of the messages below the diagram as message $\rightarrow$ create($C$), where $C$ is the commitment that the message creates. The customer’s acceptance of the reseller’s quote creates $C_2$ and $C_3$. And the reseller’s acceptance of the distributor’s quote creates $C_5$. These commitments are defined in the Quote-To-Cash business model of Figure 3.12.

In Figure 3.15(b), the customer sends an order to the reseller, and the reseller in turn sends an order to the distributor. This message sequence is guarded by the customer’s quote accept message that appears in Figure 3.15(a). The customer’s purchase order sent to the reseller creates $C_1$, the reseller’s purchase order sent to the distributor creates $C_4$, and the distributor’s confirmation sent to the customer creates $C_6$.

Figure 3.15(c) shows the quoting interaction between the distributor and the shipper. The quote sent by the shipper to the distributor creates $C_7$, the quote acceptance sent by the distributor to the shipper creates $C_8$, and the confirm ship request message sent by the shipper to the customer creates $C_9$.

In Figure 3.15(d), the shipper ships the goods to the customer. This sequence is guarded by the shipper’s confirm ship request message to the customer. By shipping the goods to the customer, the shipper satisfies $C_9$, which in turn satisfies the pending commitments $C_6$ and $C_3$. For brevity, we use the name of each task (e.g., shipGoodsE) in the business model as the name of the message (e.g., shipGoodsE) that completes the task. This is not necessary in our approach. It would be a simple matter to use different names but we would have to map each message to the corresponding task.

Similarly, Figures A.4 and A.5 (in the online appendix) show the remaining sequence diagrams for Quote-To-Cash. Figures A.4(e), (f), (g), and (h) show product quoting, ship quoting, shipping, and service quoting, respectively. Figure A.5(i) shows the customer’s request for service, and Figures A.5(j) and (k) show various payment and install interactions. These diagrams show the commitments that the messages create. In some sequence diagrams, commitments are satisfied as their consequent is brought about.

On a computer with 2.66 GHz Intel Core 2 Duo processor, and 4 GB memory, NuSMV verifies the Quote-To-Cash model in 0.1 seconds. The sequence diagrams from this section satisfy the Quote-To-Cash business model from Figure 3.12. Figure 3.16 shows a partial screen shot of NuSMV output. It shows several specifications that the model satisfies. For example, the highlighted specification is NuSMV equivalent of a specification from unilateral commitment pattern: $AG = (\text{INACTIVE} \rightarrow AX(\text{INACTIVE} \lor \text{ACTIVE} \lor \text{DETACHED}))$. The output shows that the model satisfies this specification for commitment $C_1$. We now present a few variations of these sequence diagrams that fail to satisfy the model.
Potential Violations

Because the above model is verified successfully, we consider some (imaginary) potential violation scenarios to further demonstrate our approach.

Scenario 1: Failure to confirm the order

If we remove the confirmOrderDC message from the sequence diagram of Figure 3.15(b), the distributor fails to confirm the order, thus violating these specifications:

1. $\text{AG}(\text{DETACHED} \rightarrow \text{AF} (\text{SATISFIED}))$ for $C_5$: The NuSMV counterexample shows an execution in which $C_5$ detaches since the reseller pays the distributor. However, $C_5$ never satisfies since the distributor never confirms the order, that is, never creates $C_6$. Therefore, the distributor violates $C_5$. At a business level, this situation is undesirable since the reseller pays the distributor to create a commitment to ship the goods to the customer, but the distributor fails to create that commitment.

2. $\text{AG}(\text{VIOLATED}_5 \rightarrow \text{AF} (\neg \text{PENDING}_3))$: In the operational model, the (original) commitment $C_3$ becomes pending when reseller accepts the distributor’s quote ($\text{responseRD} = \text{accept}$). By not sending the confirmOrderDC message, the distributor violates the (outsourced) commitment $C_5$. But $C_3$ stays pending, and it is not reactivated. At a business level, this is undesirable since the reseller who is responsible for satisfying $C_3$ fails to reactivate it after the outsourced commitment is violated.

Scenario 2: Failure to ship the goods

Here we remove the shipGoodsE message from the sequence diagram of Figure 3.15(d), that is, the shipper fails to ship the goods to the customer. Figure 3.17 shows a partial screen shot of the NuSMV output, highlighting that the model violates the specification $\text{AG}(\text{DETACHED} \rightarrow \text{AF} (\text{SATISFIED}))$ for commitment $C_6$. The model also violates the same specification for commitment $C_9$. In the counterexample the commitments are detached, but are never satisfied since their consequent shipGoodsE is not brought about. Note that the specification $\text{AG}(\text{DETACHED} \rightarrow \text{AF} (\text{SATISFIED}))$ is not violated for $C_3$ although it has the consequent shipGoodsE. This is because, as Figure A.5(j) shows, the customer does not pay the reseller until the goods are received. Since shipGoodsE is removed, the customer never receives the goods and, therefore, $C_3$ never detaches.

3.7 Runtime Violations

The above examples demonstrate how our approach detects commitment violations in an operational model at design time. However, a participant may violate a commitment at runtime. To handle such cases,
the business model may employ the penalty pattern [Singh et al., 2009]. For example, if the reseller ships
the goods but the customer fails to pay $10 within 15 days, that is, the customer violates a commitment.
Then a penalty applies, committing the customer to pay $15 within 30 days to the reseller.

3.8 Conclusion

This chapter presents a business metamodel, a set of modeling patterns, and an approach for formalizing
business models and verifying operations with respect to models. Using a real-life business scenario,
we evaluate both our model and patterns (Section 3.3.1), and our verification approach (Section 3.6).
Our set of business model patterns is clearly not exhaustive nor do we expect any set of patterns to be
exhaustive—hundreds of patterns exist for programming and for software architecture, and the domain
of business models appears no less complex than those domains. However, our core set of patterns
shows how we may construct additional patterns. Our approach helps a business modeler concentrate on
high-level commitments, and helps detect flaws in business process implementations. We establish the
practical usability of the approach by applying it to a real-world business process.
Figure 3.17: NuSMV output showing violation: failure to ship goods.

3.9 Related Work

Business Process Modeling

Hofreiter et al. [2006] present UN/CEFACT’s methodology UMM for modeling interorganizational business processes as global choreographies. Similar to our approach, UMM specifies a choreography at a business level, independently of the underlying implementation technology. However, unlike UN/CEFACT’s UMM model, our metamodel naturally captures business aspects by giving primacy to the commitments among the participants. Further, the well-defined semantics of a commitment and its operations provide a basis for formally verifying operational executions.

Rosenberg et al. [2008] highlight the lack of a modeling method that derives a cross-organizational business process starting from service level agreements (SLA) among the participants. They propose a methodology that extends the web services choreography description language (WS-CDL) to capture SLAs specified in the web service level agreement (WSLA) language. Unlike our metamodel, this approach fails to capture the agreements (relationships) among the participants at a business level, and lacks the flexibility that our metamodel offers.

Milosevic et al. [2006] propose a method that derives a cross-organizational business process starting from a contract between the participants. This is similar in spirit to our approach, which starts from a
business model to derive an operational model. However, Milosevic et al.’s approach models a contract using the deontic notion of an obligation. Although obligations are similar to commitments, they are more limited since they are not directed, and cannot be readily manipulated. Further, unlike our approach, Milosevic et al.’s method lacks a formal approach to verify if the derived business process satisfies the original contract.

Bodenstaff et al. [2010] present a methodology to check consistency between different models of an interorganizational scenario. Their methodology is model-independent, but their notion of consistency fails to consider the temporal progression of a model. In contrast, our method verifies the temporal progression of an operational model, with respect to the business model.

Formal Approaches

Fornara and Colombetti [2009] model an open interaction system in terms of social commitments, events, agents, roles, and norms. They specify a model using an ontology in OWL, and use SWRL and a custom Java program to monitor the temporal evolution of social commitments. We specify the desired properties of a business model using CTL, and at design time, model check operational interactions specified as UML sequence diagrams. Unlike the patterns in our approach that model a business scenario, their approach lacks principled means of creating a model.

Winikoff [2006] argues that a message-centric agent interaction model limits flexibility and autonomy of the agents. He proposes an approach using commitments for modeling agent interactions. We indeed share the same motivation, and base our business model on commitments. Winikoff further outlines a multistep process for designing an interaction model, which he applies on a simple meeting scheduler application. However, unlike our approach, Winikoff’s approach is at a lower-level, and lacks reusable patterns.

The Logic for Contract Representation (LCR) [Dignum et al., 2002b] specifies interactions in Opera [Weigand et al., 2002]. Avali and Huhns [2008] share our notion of commitments. They apply BDI\textsubscript{CTL} to relate commitments to an agent’s beliefs, desires, and intentions. LCR and Avali and Huhns’ approach support reasoning about obligations and commitments, respectively. Such reasoning would complement our work. By contrast, our approach focuses on interactions among real-world organizations and enables us to develop high-level business models, and to verify those with respect to low-level operational models. Further, we develop a set of real-world business patterns.

Fuxman et al. [2001] present an approach to check a Tropos specification for consistency and other properties that capture stakeholder expectations. Our approach checks an operational model with respect to a business model, and the semantics of a commitment and its operations yield the model specifications.

Pijpers and Gordijn [2008] present a semiformal approach for checking the consistency of a process model (UML activity diagram) with respect to an e\textsuperscript{3}-value model. Unlike our verification approach, their approach is manually intensive since it requires developing an intermediate e\textsuperscript{3}-value physical model.
Further, their notion of consistency is highly rigid. They require a process model to contain all value exchanges that an e³-value model specifies. In contrast, our approach does not mandate an execution to create and satisfy all of the model commitments. Instead, it only requires detached commitments to be satisfied prior to the detached timeout.
Chapter 4

Comma: Modeling Methodology

This chapter introduces Comma, our proposed methodology, and an extensive user study that compares Comma with a traditional methodology.

For each business pattern from Section 3.2 we develop a set of generalized (templatic) message sequence charts that operationalize that pattern. As an example, Figure 4.1 shows the MSCs for the outsourcing pattern using UML 2.0 sequence diagram [Obj. 2004] operators OPT(ion) and ALT(ernative).

![Diagram of MSCs for outsourcing]

We go beyond UML in labeling each message with its meaning. A message labeled with a proposition, usually part of the antecedent or consequent of some commitment, simply brings about that proposition.

Figure 4.1: Message sequence charts for outsourcing.

47
A message labeled $m_i$ for some $i$ means an operation on some commitment (such as its creation), which we annotate on the side. In Figure 4.1(a), the outsourcer sends $m_1$ to the client, which creates commitment $C_1$. The client sends payOut to the outsourcer upon receiving $m_1$, which detaches $C_1$ since it is $C_1$’s antecedent. In Figure 4.1(b), after receiving $m_1$, the outsourcer sends $m_2$ to the contractor, and after receiving $m_2$ the contractor sends $m_3$ to the outsourcer. Alternatively, the contractor first sends $m_3$ to the outsourcer, and after receiving $m_3$, the outsourcer sends $m_2$ to the contractor. $m_2$ creates $C_3$ and $m_3$ creates $C_4$. In Figure 4.1(c), after $m_2$ and $m_3$ are exchanged, the outsourcer sends payCon to the contractor and the contractor sends $m_4$ to the outsourcer in either order. Now payCon satisfies $C_3$ and detaches $C_4$; and, $m_4$ creates $C_2$ and satisfies $C_4$. In Figure 4.1(d), after $m_4$ is exchanged, the contractor sends task (message) to the client. This satisfies $C_1$ and $C_2$ since task is their consequent. As part of creating a model, a modeler substitutes the message labels $m_i$ with domain-specific terms.

### 4.1 Methodology: Comma

The Comma methodology begins from an informally described real-life cross-organizational scenario and produces formal business and operational models. Table 4.1 summarizes Comma.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Extract subscenarios corresponding to Comma patterns</td>
<td>Real-life cross-organizational scenario</td>
<td>Subscenarios</td>
</tr>
<tr>
<td>2</td>
<td>Identify roles from each subscenario</td>
<td>Subscenario</td>
<td>Roles</td>
</tr>
<tr>
<td>3</td>
<td>Identify business tasks from each subscenario</td>
<td>Subscenario</td>
<td>Tasks</td>
</tr>
<tr>
<td>4</td>
<td>Introduce a Comma pattern for each subscenario</td>
<td>Comma pattern, subscenario, roles, tasks</td>
<td>Business model</td>
</tr>
<tr>
<td>5</td>
<td>Introduce MSCs for each Comma pattern</td>
<td>Comma pattern MSCs, subscenario, roles, tasks</td>
<td>Operational model</td>
</tr>
</tbody>
</table>

**Step 1** A *subscenario* is a fragment of the given scenario. From the given scenario description, extract subscenarios such that each match a pattern from the Comma pattern library.

**Step 2** For each subscenario, identify its roles. A subscenario usually describes participants using a combination of generic terms (e.g., Company, Partner, and Organization) and specific names (e.g., FedEx). This step involves creating roles based on business function (e.g., Shipper) that remove
any ambiguity, such as if Partner and Organization refer to the same entity.

**Step 3** For each subscenario, identify business tasks (e.g., goods and payment) that a role executes. A scenario typically specifies the tasks as actions executed by the participants.

**Step 4** From the Comma pattern library, introduce into the business model a pattern corresponding to each subscenario. Rename the pattern characters with the roles from Step 2, and introduce the tasks from Step 3 as the antecedents and consequents of the appropriate commitments. The patterns compose naturally when the same roles are referenced by more than one pattern.

**Step 5** For each Comma pattern, introduce its MSC into the operational model. Rename the roles and messages in the MSCs to align them with those determined in Steps 2 and 3. Customize the MSCs to capture any subscenario-specific operational details, such as additional messages, guards, and loops.

### 4.2 Design of the Study

Our study used an initial scenario based on real-life cross-organizational business processes, inspired by the Oracle Quote-To-Cash process [Oracle, 2009; Telang and Singh, 2012], and two modifications of the scenario.

$S_1$, the initial scenario, involves MedEq, a company that sells medical equipment. MedEq designs the equipment in house, and out-sources manufacturing to two contract manufacturers, FlexMan and SoleMan, and shipping to two shippers, FedUp and UpFed. To purchase the equipment, a customer submits its requirements to MedEq. MedEq analyzes the requirements, and creates a proposal containing the equipment details, and a quoted price. The customer may accept the proposal or negotiate for a better price. There can be up to two iterations between MedEq and the customer before they either agree upon the price, or abort the transaction. If MedEq and a customer reach an agreement, the customer proceeds to placing an order and specifying the equipment, shipping address, contact information, and payment information. Upon receiving the order, MedEq validates the order. MedEq accepts the order if it is valid and rejects it otherwise. MedEq maintains warehouses in which it stocks the equipment. In case the ordered equipment is in stock, MedEq requests a shipper to ship the equipment to the customer. MedEq pays the shipping charges to the shipper.

If the equipment necessary to fulfill an order is not in stock, MedEq places a stock replenishment order with a contract manufacturer. The contract manufacturer employs a shipper to ship the equipment to MedEq’s warehouse. MedEq pays the contract manufacturer for the equipment. Once the equipment is in stock, MedEq fulfills the customer’s order.

$S_f$, the first modification, adds a new participant, a value-added reseller, MedRes. MedRes sells, installs, and supports (i.e., services) medical equipment. The customer now places its order with MedRes,
who orders the equipment from MedEq and provides the installation and support itself. The customer pays MedRes, and MedRes pays MedEq. MedRes supports the equipment as needed. The rest of the scenario remains unchanged.

$S_s$, the second modification, removes the contract manufacturers SoleMan and FlexMan from the original scenario. The rest of the scenario is unchanged.

### 4.2.1 Study Solution

![Comma model for $S_s$.](image)

Figure 4.2: Comma model for $S_s$.

Figure 4.2 shows the solution Comma model for the initial scenario, $S_s$. For brevity, we present only the final Comma model and omit the outputs of the intermediate methodology steps. The model is composed from the commercial transaction and the outsourcing patterns. For example, the commercial transaction pattern captures MedEq (Company) and the customer agreeing to exchange medical equipment for certain price. The model commitments $C_1$ and $C_2$ correspond to this pattern: in $C_1$, the customer commits to paying the company (payComp) if the company provides the equipment (goodsCust), and in $C_2$, the company commits to providing the equipment if the customer pays. The outsourcing pattern models MedEq employing a shipper (Shipper 1) to ship the medical equipment to the customer. The model commitments $C_2$, $C_3$, $C_4$, and $C_5$ correspond to this pattern: $C_2$ is the original commitment, $C_5$ is
the outsourced commitment, and $C_3$ and $C_4$ are the commitments in which the company and the shipper commit to paying and to creating $C_5$, respectively. Figure 4.3 shows four of the ten MSCs for the initial scenario, $S_i$, developed using Comma. These MSCs correspond to MedEq outsourcing the shipping to a shipper. We omit further description of these MSCs since Section 4.1 describes the outsourcing MSCs in detail.

Table 4.2: RosettaNet model PIPs for $S_i$.

<table>
<thead>
<tr>
<th>PIP</th>
<th>Name (shortened)</th>
<th>Subscenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>3A1</td>
<td>Request quote</td>
<td>Customer, MedEq negotiate</td>
</tr>
<tr>
<td>3A4</td>
<td>Purchase order</td>
<td>Customer orders from MedEq</td>
</tr>
<tr>
<td>3B12</td>
<td>Request shipping</td>
<td>MedEq ships to Customer</td>
</tr>
<tr>
<td>3C3</td>
<td>Notify of invoice</td>
<td>Shipper invoices MedEq, MedEq invoices customer, shipper invoices manufacturer, manufacturer invoices MedEq</td>
</tr>
<tr>
<td>3C4</td>
<td>Reject invoice</td>
<td>MedEq, customer, or manufacturer reject invoice</td>
</tr>
<tr>
<td>3C6</td>
<td>Remittance advice</td>
<td>MedEq pays the shipper, customer pays MedEq, manufacturer pays MedEq, MedEq pays manufacturer</td>
</tr>
<tr>
<td>7B5</td>
<td>Manufacturing order</td>
<td>MedEq orders from manufacturer</td>
</tr>
<tr>
<td>3B12</td>
<td>Request shipping</td>
<td>Manufacturer ships to MedEq</td>
</tr>
</tbody>
</table>
Table 4.3: Study exercises.

<table>
<thead>
<tr>
<th>Group A</th>
<th>Group A’</th>
<th>Group A”</th>
<th>Group B</th>
<th>Group B’</th>
<th>Group B”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Develop RosettaNet model and MSCs for $S_i$</td>
<td>Develop Comma model and MSCs for $S_i$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Modify $C$ to model $S_f$ Modify $R$ to model $S_f$</td>
<td>Modify $C$ to model $S_f$ Modify $R$ to model $S_f$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Modify $R$ to model $S_s$ Modify $C$ to model $S_s$</td>
<td>Modify $R$ to model $S_s$ Modify $C$ to model $S_s$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2 shows the RosettaNet model PIPs for the initial scenario, $S_i$. For example, the customer uses PIP 3A1 to request a quote from MedEq. Figure 4.4 shows three of the thirteen MSCs for the initial scenario, $S_i$, developed using RosettaNet. Figure 4.4(a) is the MSC for PIP 3B12 in which MedEq requests the shipper to ship the equipment to the customer. The shipper either accepts or rejects the request. The shipper invoices MedEq using PIP 3C3 in Figure 4.4(b). MedEq may reject the invoice using PIP 3C4. In Figure 4.4(c), MedEq notifies the shipper of remittance advice using PIP 3C6.

4.2.2 Study Mechanics and Threat Mitigation

We conducted a developer study with 34 subjects (graduate computer science students). Three exercises, corresponding to the three scenarios, $S_i$, $S_f$, and $S_s$, comprised the study. The study used a between-subject experimental design [Juristo and Moreno, 2001]. For each exercise, the study divided the subjects into two groups who applied different methodologies to model the same scenario. We carefully designed the study to mitigate the well-known threats [Juristo and Moreno, 2001] to its validity.

To mitigate the threat of skill differences between the participants, prior to the exercises, we surveyed the study subjects to gather information on their educational background, and experience in process
modeling and software engineering. We then divided the participants into two groups, A and B, of approximately equal skill levels. The first exercise compared groups A and B, and the subsequent exercises split and merged the same groups. For the first exercise, the subjects in groups A and B developed a model and MSCs for $S_i$ using RosettaNet and Comma, respectively.

For the second and third exercises, a primary threat was the learning effect, because after the first exercise, subjects would be familiar with the methodology they used. To mitigate this threat, we divided each group into two subgroups of equal size and combined a subgroup from each group to form new groups $A'B'$ and $A''B''$. A secondary threat was variance in the initial models developed by different subjects and their lack of familiarity with models developed by others. To mitigate this threat, we developed $C$ and $R$, respectively, Comma and RosettaNet model and MSCs for the initial scenario $S_i$.

In the second exercise, group $A'B'$ began from $C$ and applied Comma, and group $A''B''$ began from $R$ and applied RosettaNet, both to account for $S_f$.

In the third exercise, we swapped the two groups. Group $A'B'$ reviewed $R$ and applied RosettaNet, and group $A''B''$ reviewed $C$ and applied Comma, both to account for $S_s$.

Figure 4.5 summarizes how the study divided the subjects into groups, and Table 4.3 summarizes the exercises.

The subjects self-reported the time and difficulty for each methodology in a work log. To mitigate the threat of a subject forgetting to report relevant information, we required each subject to submit his or her work log three days a week, regardless of the effort they spent in that period.

### 4.2.3 Dependent Variables

This section describes the dependent variables of the study that we use to compare Comma and RosettaNet.
Table 4.4: Quality measures, as judged by experts.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Captures a methodology’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Coverage. Percentage of models that fully cover the problem scenario</td>
<td>Completeness in modeling a scenario</td>
</tr>
<tr>
<td>Model Precision. Percentage of models that include no aspects unrelated to the problem scenario</td>
<td>Effectiveness in avoiding bloated models</td>
</tr>
<tr>
<td>MSC Structure. Percentage of MSCs with correct and complete guards</td>
<td>Soundness: fewer errors in outcomes</td>
</tr>
<tr>
<td>MSC Flexibility. Average number of ALT (ernative) blocks per MSC</td>
<td>Support for participants’ flexibility</td>
</tr>
<tr>
<td>MSC Abstraction. Percentage of MSCs that use a role, not an agent, name</td>
<td>Support for reusability of models</td>
</tr>
</tbody>
</table>

Quality of the models, assessed by experts, using the measures of Table 4.4. (A higher value is better for each.)

Difficulty in completing a methodology step as (subjectively) reported by a subject. Difficulty ranges over extremely easy, easy, neutral, difficult, and extremely difficult. Subjects reported the difficulty in a work log; we calculate the percentage of responses for each difficulty level. In most reports, we combine best two as easy and the worst two as difficult.

Time taken to complete a methodology step as reported by a subject: a continuous variable in the unit of hours. Subjects reported the time they spent in a work log; we summed up the time for each subject.

4.3 Study Results

This section describes the key findings from the study.

4.3.1 Quality

Figures 4.6 and 4.7 show the quality measurements of the two methodologies from the initial exercise $S_i$.

Observation 1. As Figure 4.6 shows, both model coverage and model precision are superior for Comma (93% and 87%, respectively) than for RosettaNet (77% and 44%, respectively).

Observation 1 suggests that Comma is more effective than RosettaNet in creating complete and precise models. We credit this to the systematic nature of Comma and the fact that it focuses attention on
the relevant commitments and MSCs. On the contrary, the RosettaNet models tend to contain several superfluous PIPs.

**Observation 2.** As Figure 4.6 shows, the percentage of models in which MSCs do not miss any necessary guards is higher for Comma (81%) than for RosettaNet (33%).

Since RosettaNet focuses on individual interactions in the form of PIPs, a modeler often loses an overall perspective on the scenario. The modeler develops an MSC for each PIP, but fails to relate the MSCs to each other via appropriate guards. In contrast, Comma forces a modeler to think in terms of the commitment lifecycle. For example, a message that satisfies a commitment should be preceded by a message that creates the commitment.

**Observation 3.** As Figure 4.7 shows, Comma MSCs use a higher median number of ALTs per model (six) than RosettaNet MSCs (four).

RosettaNet tends to lead to rigid MSCs, i.e., those with only a few alternative paths. The MSCs included with Comma patterns promote flexibility, which is inherent in the commitment-based approach. As a telling example, almost all subjects developed RosettaNet MSCs in which the Customer pays MedEq
strictly after MedEq ships the ordered equipment. In contrast, many subjects developed Comma MSCs in which the Customer may pay MedEq either before or after MedEq ships the ordered equipment, a situation that has been discussed since the earliest works on commitment protocols [Yolum and Singh, 2002].

**Observation 4.** The percentage of models in which MSCs use a role name instead of a participant name is higher for Comma (100%) than for RosettaNet (88%).

Observation 4 supports the idea that Comma emphasizes role abstraction and more naturally yields reusable MSCs.

Since the second and the third exercises began from the models that we provided, the resulting models are of higher quality, and without perceptible difference between the two methodologies. Therefore, we present quality results only for the first exercise.

### 4.3.2 Difficulty

![Difficulty of modeling](image)

**Figure 4.8:** Difficulty of modeling, as percentage of responses by the subjects.

Figure 4.8 shows the percentage of work log responses corresponding to each difficulty level for the three exercises.

**Observation 5.** In $S_i$, the percentage of easy responses is smaller for RosettaNet (21.6%) than for Comma (27.5%), and the percentage of difficult responses is higher for RosettaNet (28.3%) than for Comma (23.7%).

Observation 5 suggests that Comma modeling is relatively easier as compared to RosettaNet modeling.

To identify the underlying cause of the extreme difficulty reports about Comma modeling, we analyzed the reported difficulty for each step. The analysis revealed that Comma Step 4, composing
patterns to create a model, significantly contributes to the difficulty. This finding indicates the need for simplifying Step 4.

**Observation 6.** In $S_i$, the percentage of difficult responses in developing MSCs using Comma (18.3%) is smaller than using RosettaNet (23.0%). However, the percentage of extremely difficult responses to developing MSCs using Comma (3.3%) is larger than using RosettaNet (0%).

Observation 6 is mixed. Although Comma appears to have been easier than RosettaNet overall, the number of subjects who found Comma extremely difficult was greater than the corresponding number for RosettaNet. This emphasizes the need for simplifying Comma Step 5, developing MSCs. A modeling tool, already under development, can assist a modeler by creating a base MSC model using the pattern MSCs.

**Observation 7.** Comma modeling has 0% extremely difficult responses, and 9.9% somewhat difficult responses in $S_f$, as compared to 2.8% extremely difficult responses, and 20.9% percent somewhat difficult responses in $S_i$.

![Figure 4.9: Time in hours expended in creating models, as reported by subjects.](image)

We explain Observation 7 based on two factors. First, some of the subjects gained experience modeling using Comma in the initial exercise. Second, the subjects started the first modification $S_f$ from a solution that we provided.

Relative to $S_i$ and $S_f$, $S_s$ has increased responses with lower difficulty levels. This is partially due to the learning that the subjects gained from the first two exercises, and partially since $S_s$ was a relatively easy exercise.

**Observation 8.** In $S_s$, the percentages of easy responses for modifying the Comma model (56.2%) and MSCs (50%) are higher than for modifying the RosettaNet model (22.1%) and MSCs (33.3%).

Observation 8 suggests that with some experience, Comma becomes simpler than RosettaNet.
4.3.3 Time

Figure 4.9 shows boxplots of the time taken by the subjects to develop Comma and RosettaNet models and MSCs in the three exercises. Throughout, we remove each outlier: a point that is greater than the third quartile or smaller than the first quartile by 1.5 times the interquartile range—i.e., the difference between the third and first quartiles.

**Observation 9.** In $S_i$, the median time to develop a model is smaller for Comma (6.7 hours) than for RosettaNet (10 hours).

Observation 9 suggests that Comma is more efficient than RosettaNet for creating a business model.

**Observation 10.** In $S_i$, the median time to develop MSCs is somewhat greater for Comma (6 hours) than for RosettaNet (5.5 hours).

Although Comma appears less efficient than RosettaNet, as Section 4.3.1 shows, the MSCs produced from Comma are of higher quality than those produced from RosettaNet.

**Observation 11.** In $S_i$, the spreads of the times for developing the model and MSCs are smaller for Comma than for RosettaNet.

Observation 11 indicates that Comma is more predictable than RosettaNet in terms of development effort.

**Observation 12.** Using Comma, the median modeling time for the first modification $S_f$ (6.6 hours) is about the same as that for the initial exercise $S_i$ (6.7 hours).

Observation 12 is surprising to us. We expected the Comma modeling time for $S_f$ to be smaller than for $S_i$. We attribute this result to a couple of key factors. First, the subjects needed time to comprehend the solutions we provided. Second, the subjects followed the same steps for modifying the model as the steps they followed for creating the model in the initial exercise. Comma should be improved to guide modelers in modifying existing business models.

**Observation 13.** In $S_f$, the median modeling time is higher for Comma (6.6 hours) than for RosettaNet (4 hours).

Observation 13 conflicts with Observation 9 from the initial exercise $S_i$. A primary reason for this result is the difference in the nature of the artifacts involved. A RosettaNet model is expressed as a textual list of PIPs, modifying which is easy. A Comma model is expressed as a graph of business relationships, modifying which is time consuming. Indeed, since we did not provide a Comma modeling tool, subjects expended considerable effort in developing the graphical models using drawing tools such as Visio.

**Observation 14.** In $S_f$, the median time to modify MSCs is lower for Comma (1 hour) than for RosettaNet (2.3 hours).
Observation 14 suggests that the Comma methodology is more efficient as compared to the RosettaNet methodology for developing MSCs. Note that this result is an improvement over Observation 10 from the initial exercise $S_i$ in favor of Comma, indicating the benefit of learning.

Table 4.5: Hypothesis testing for model and MSC development times.

<table>
<thead>
<tr>
<th>ID</th>
<th>Time for Exercise</th>
<th>Comma Mean ($\mu_c$)</th>
<th>RosettaNet Mean ($\mu_r$)</th>
<th>Alternative Hypothesis</th>
<th>Null Hypothesis $[\mu_c = \mu_r]$</th>
<th>p-value</th>
<th>Accepted at p-value of 5%?</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>$S_i$-Model</td>
<td>7.19</td>
<td>10.05</td>
<td>$\mu_c &lt; \mu_r$</td>
<td>0.046</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>H2</td>
<td>$S_i$-MSC</td>
<td>6.22</td>
<td>6.73</td>
<td>$\mu_c &lt; \mu_r$</td>
<td>0.610</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>H3</td>
<td>$S_f$-Model</td>
<td>7.59</td>
<td>4.84</td>
<td>$\mu_c &gt; \mu_r$</td>
<td>0.026</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>H4</td>
<td>$S_f$-MSC</td>
<td>1.42</td>
<td>2.26</td>
<td>$\mu_c &lt; \mu_r$</td>
<td>0.062</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>H5</td>
<td>$S_s$-Model</td>
<td>2.77</td>
<td>3.74</td>
<td>$\mu_c &lt; \mu_r$</td>
<td>0.290</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>H6</td>
<td>$S_s$-MSC</td>
<td>0.70</td>
<td>1.29</td>
<td>$\mu_c &lt; \mu_r$</td>
<td>0.053</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Observation 15. In $S_s$, the median times to modify the Comma model (2.75 hours) and MSCs (0.75 hours) are slightly smaller than the median times to modify the RosettaNet model (3 hours) and MSCs (1 hour), respectively.

Observation 15 suggests that Comma is slightly more efficient than RosettaNet for modifying models. This agrees with Observation 9 from the initial exercise $S_i$.

Observation 16. In $S_s$, the spreads of times taken in modifying the model and MSCs are smaller for Comma than for RosettaNet.

Observation 16 agrees with Observation 11, and reconfirms that Comma is more predictable than RosettaNet.

The above observations are from the descriptive statistics summarized by the box plots. We now present the results of formal hypothesis testing that checks if the difference between the timings of the two methodologies is statistically significant. Table 4.5 summarizes the hypotheses and the outcome of the independent samples t-test for each of them. $H_1$, $H_3$, and $H_5$ test the statistical significance of the difference between the modeling time of the two methodologies in $S_i$, $S_f$, and $S_s$, respectively. $H_2$, $H_4$, and $H_6$ test the statistical significance of the difference between the MSC development time of the two methodologies in $S_i$, $S_f$, and $S_s$, respectively. In $H_1$, the alternative hypothesis is $\mu_p < \mu_r$, that is, the mean time to develop the Comma model $\mu_p$ is less than the mean time to develop the RosettaNet model $\mu_r$. The corresponding null hypothesis is $\mu_p = \mu_r$, that is, the mean time to develop the Comma model.
is the same as the mean time to develop the RosettaNet model. The t-test rejects the null hypothesis with a p value of 0.046 at the 0.05 level of significance. This confirms that Comma is more efficient than RosettaNet in $S_5$, which agrees with Observation 9.

The t-test rejects the null hypothesis in $H_3$. This indicates that RosettaNet is more efficient than Comma in the first modification $S_f$. We discuss the reasons behind this result in Observation 13.

Since the t-test accepts $H_2$, $H_4$, $H_5$, and $H_6$, we conclude that the time differences for (1) modeling in $S_s$ and (2) developing MSCs in all exercises is not statistically significant.

### 4.4 Conclusion

We introduced Comma, a novel commitment-based methodology for business modeling. We carried out a substantial empirical evaluation of the effectiveness of Comma. We note in passing that such evaluations are not yet common in AOSE, though they are quite prevalent in the broader software engineering community.

We now summarize the lessons we learned. Our study confirmed the benefits in quality that we expected from Comma because of its foundation in commitments. Specifically, Comma does better on every quality measure: model coverage and precision, and MSC structure (guards), flexibility, and abstraction. The study demonstrated gains in ease of use from Comma in producing models but yielded mixed results with respect to MSCs. Comma yields a superior MSC product, but with a slightly greater difficulty. We expect to see benefits from improving the tooling and training materials supporting Comma. The time spent shows an improvement for Comma though with anomalies. Here too we conjecture that improved tooling and training will prove crucial.

### 4.5 Related Work

Researchers have proposed several agent-oriented software development methodologies [Deloach et al., 2001; Padgham and Winikoff, 2005; Cheong and Winikoff, 2009; Weigand et al., 2002]. Many of these methodologies focus on modeling a multiagent system that is under the control of a single organization. In contrast, Comma models cross-organizational relationships. In Comma, a high-level model based on commitments captures the social relationship among agents (the organizations that are business partners). Unlike Comma, many of the current Agent Oriented Software Engineering methodologies lack an appropriate abstraction for modeling social relationship between the agents.

Tropos [Bresciani et al., 2004] resembles Comma in terms of employing high-level concepts. A key difference between the two is how they model social relationships: Tropos employs goal and other dependencies whereas Comma employs commitments. Unlike dependencies, commitments are flexible as they can be manipulated. Commitments reflect the autonomy of the partners since each debtor adopts its commitments through its autonomous actions (communications).
Opera is a framework for modeling multiagent societies [Weigand et al., 2002], though from the perspective of a single designer or economic entity. In contrast, we model interactions among multiple entities. Opera’s concepts of landmark, scene, and contract are close to our concepts of task, protocol, and commitment, respectively. However, Opera uses traditional obligations, which lack the flexibility of commitments.

Mazouzi et al. [Mazouzi et al., 2002] model agent interaction protocols using Agent UML (AUML), and subsequently translate them into Colored Petri Nets (CPN) to verify low-level properties such as liveness. In contrast, in Comma, a modeler first develops a high-level business model, which provides the correctness properties at a business level [Telang and Singh, 2012]. Starting from a business model, the modeler develops agent interaction MSCs. Comma employs model-checking to verify if the MSCs satisfy the business model [Telang and Singh, 2012].

Verdicchio and Colombetti [2002] propose a commitment-based approach for modeling the flow of money, and goods in a supply chain. Our work shares its motivation with the above, but goes beyond it in two major ways. First, we propose business patterns that capture the structure of modern business ecosystems. Second, we formalize the patterns, and show how operations can be verified with respect to a business model.

Desai et al. [2009] employs commitment protocols for process modeling. Amoeba and Comma share the same underlying notion of commitments. In contrast to Comma, which is a methodology for business relationship modeling, Amoeba is a methodology for lower-level interaction modeling, and seeks to specify the protocols whose composition corresponds to the given business process.

El-Menshawy et al. [2009] extend our approach from [Telang and Singh, 2009] with argumentation to develop a methodology to model communities of web services. Unlike Menshawy et al.’s methodology, our approach is founded upon patterns. We formalize the patterns, and show how business executions are verified with respect to a business model.

Telang and Singh [2010] approach RosettaNet from the opposite end to the present paper. They abstract out business modeling patterns from RosettaNet PIPs, in essence by identifying the commitments of the business partners involved that are implicitly understood in each PIP. That is, Telang and Singh discuss how to create and apply patterns that could be included in the Comma library. They use the commitment life cycle as a basis for verifying process specifications.

Gordijn and Wieringa [2003] propose the e³-value approach, which captures a business organization as an actor, similar to our notion of an agent. Actors execute value activities, similar to our tasks. In e³, a value interface aggregates related in and out value ports of an actor to represent economic reciprocity. This concept is close to our concept of commitment, but it lacks formal semantics and doesn’t yield flexibility. For example, unlike value interfaces, commitments can be readily delegated. Due to this, during execution, the exchange and interaction may take place among actors different from those included in an e³ model.
Chapter 5

Modeling Healthcare Processes

Healthcare involves multiple autonomous participants—patients, caregivers, hospital administrators, laboratories, and insurance companies—carrying out complex interdependent activities. As in other domains, process specifications seek to reduce the communication gap between participants, increase operational efficiency, and improve the quality of service provided. Today, processes are commonly specified using the workflow concepts of task, control flow, and data flow. The workflow representation—exemplified by BPMN, the Business Process Modeling Notation [OMG, 2010]—emphasizes low-level operational details (the steps that each party must take under what conditions) as opposed to the meaning of the interaction (who relies on whom for what; who agrees to undertake what; and so on). Consequently, workflow models tend to be over-specified and rigid and do not readily admit of variations due to changing circumstances. Thus, such models are especially unsuitable for health care, where ad hoc interactions among individual and organizational participants are routine.

To address these shortcomings, we propose the Comma methodology to model healthcare processes that emphasizes the business relationships among the participants in terms of their commitments [Singh, 1999; Fornara and Colombetti, 2010] to one another. Commitments characterize the interactions among the participants in meaningful terms. The commitments provide a standard of correctness and the interactions can be realized in multiple ways depending upon the particular circumstances. For example, the physician may delegate part of the development of a treatment plan to a physical therapist or a patient counselor. The radiologist may rely upon a report from a fellow in order to support her claim.

In addition to practical commitments, we consider dialectical commitments and organizational context in modeling healthcare processes.

5.1 Dialectical Commitments

In a practical commitment, the debtor commits to providing a creditor a concrete or abstract thing or service, and the commitment is discharged when the thing or service has been delivered. In a dialectical
commitment, the debtor provides a guarantee of functionality, suitability, validity or truth of the consequent. The failure of the guarantee may activate one or more penalty commitments. For example, the physician (debtor) commits practically to the patient (creditor) to develop a treatment plan (consequent) if the patient enrolls in his care (antecedent). The radiologist (debtor) commits dialectically to the physician (creditor) that his report is correct (consequent). The correctness of report may be determined by a higher authority such as the tumor board.

We write a dialectical commitment using a notation similar to practical commitment: D(DEBTOR, CREDITOR, antecedent, consequent). For example, D(RADIOLOGIST, PHYSICIAN, ⊤, tumorBoardAgreesOnReport) means that the radiologist dialectically commits to the physician that tumor board would agree on the correctness of his report. We also allow the debtor to be a group of roles to handle the cases in which the group collectively guarantees the consequent. For example, in the breast cancer screening and diagnosis scenario, the radiologist and the pathologist jointly guarantee the accuracy of their integrated report to the physician: D({RADIOLOGIST, PATHOLOGIST}, PHYSICIAN, ⊤, tumorBoardAgreesToIntegratedReport). We now introduce a lifecycle for dialectical commitments.

### 5.1.1 Lifecycle of Dialectical Commitments

Figure 5.1 shows the lifecycle of a dialectical commitment as a parallel statechart diagram. The left-hand side of the statechart shows the objective state of the commitment, whereas the right-hand side shows the social state. The objective state is computed solely based on the antecedent and the consequent which are objective facts, whereas the social state is computed based on the actions that either the creditor or the debtor perform.

A dialectical commitment is in the state null before it is created. On the objective side, upon creation, the commitment transitions to the state active. The active state has two substates: conditional and detached. If the antecedent is not true, the commitment is in the state conditional. When the antecedent becomes true, the commitment transitions from the state conditional to the state detached. When the commitment is in the conditional state, if the antecedent fails, then the commitment transitions to the expired state. The commitment transitions to the state satisfied if its consequent is true when it is active. The commitment transitions to the state violated if its consequent fails when it is detached.

On the social side, upon creation, the commitment transitions to the state asserted. If the debtor cancels the commitment when it is asserted, then the commitment transitions to the state terminated. If the debtor suspends or the creditor releases the commitment when it is asserted, then the commitment transitions to the state pending. If the debtor reactivates the commitment when it is pending, then the commitment transitions to the state asserted.

Consider the interactions between the radiologist and pathologist from the breast cancer screening and diagnosis scenario from Section 1.1.2. Figure 5.2 shows a possible execution in which the radiologist and the pathologist interact to decide if the patient has cancer. The radiologist informs the pathologist
that the patient is free of cancer, and thus creates the dialectical commitment: $D_R = D$(RADIOLOGIST, PATHOLOGIST, $\top$, tumorBoardAgreesNoCancer). Upon creation, $D_R$’s objective state transitions to detached since its antecedent is true ($\top$), and its social state transitions to asserted. However, the pathologist disagrees with the radiologist and challenges his diagnosis, and creates the dialectical commitment: $D_P = D$(PATHOLOGIST, RADIOLOGIST, $\top$, tumorBoardAgreesHasCancer). Similar to $D_R$, $D_P$’s objective state transitions to detached, and social state transitions to asserted. It is possible that the radiologist and the pathologist resolve their difference of opinion among themselves. But in Figure 5.2 they seek input from the tumor board on the patient’s condition. In this execution, the tumor board concludes that the patient has cancer. This transitions $D_R$’s objective state to violated, and $D_P$’s objective state to satisfied. Finally, the radiologist agrees that the patient has cancer, that is, the radiologist cancels $D_R$, making its social state terminated.

Formalization

In order to enable verification of operational models, we now formalize the lifecycle of dialectical commitments in CTL. We group the specification into three types: state transitions, terminal states, and desirable executions. The CTL specifications for state transitions and terminal states follow from the lifecycle of dialectical commitments.
$D_R = D(\text{RAD}, \text{PATH}, \top, \text{tumorBoardAgreesNoCancer})$

$D_P = D(\text{PATH}, \text{RAD}, \top, \text{tumorBoardAgreesHasCancer})$

**Figure 5.2:** Dialectical commitments in radiologist and pathologist interactions.

**State Transitions**

- On any path, in a given state $s$, if a dialectical commitment’s objective state is $null$, then on all paths emanating from $s$ in the next state, the commitment’s objective state may remain $null$ or may transition to $conditional$, $expired$, $detached$, $satisfied$, or $violated$.

$$\text{AG} (D^N_{\text{obj}} \rightarrow AX D^N_{\text{obj}} V E V D V S V V)$$

- On any path, in a given state $s$, if a dialectical commitment’s objective state is $conditional$, then on all paths emanating from $s$ in the next state, the commitment’s objective state may remain $conditional$ or may transition to $detached$, $expired$, or $satisfied$.

$$\text{AG} (D^C_{\text{obj}} \rightarrow AX D^C_{\text{obj}} V E V D V S)$$

- On any path, in a given state $s$, if a dialectical commitment’s objective state is $detached$, then on all paths emanating from $s$ in the next state, the commitment’s objective state may remain $detached$ or may transition to $violated$ or $satisfied$.

$$\text{AG} (D^D_{\text{obj}} \rightarrow AX D^D_{\text{obj}} V V V S)$$
• On any path, in a given state $s$, if a dialectical commitment’s social state is null, then on all paths emanating from $s$ in the next state, the commitment’s social state may remain null or may transition to asserted.

$$AG \left( D_{soc}^N \rightarrow AX \; D_{soc}^{N\lor R} \right)$$

• On any path, in a given state $s$, if a dialectical commitment’s social state is asserted, then on all paths emanating from $s$ in the next state, the commitment’s social state may remain asserted or may transition to terminated or pending.

$$AG \left( D_{soc}^R \rightarrow AX \; D_{soc}^{R\lor T\lor P} \right)$$

• On any path, in a given state $s$, if a dialectical commitment’s social state is pending, then on all paths emanating from $s$ in the next state, the commitment’s social state may remain pending or may transition to asserted.

$$AG \left( D_{soc}^P \rightarrow AX \; D_{soc}^{P\lor R} \right)$$

**Terminal States**

• On any path, in a given state $s$, if a dialectical commitment’s objective state is expired, then on all paths emanating from $s$ in the next state, the commitment’s objective state remains expired.

$$AG \left( D_{obj}^E \rightarrow AX \; D_{obj}^E \right)$$

• On any path, in a given state $s$, if a dialectical commitment’s objective state is satisfied, then on all paths emanating from $s$ in the next state, the commitment’s objective state remains satisfied.

$$AG \left( D_{obj}^S \rightarrow AX \; D_{obj}^S \right)$$

• On any path, in a given state $s$, if a dialectical commitment’s objective state is violated, then on all paths emanating from $s$ in the next state, the commitment’s objective state remains violated.

$$AG \left( D_{obj}^V \rightarrow AX \; D_{obj}^V \right)$$

• On any path, in a given state $s$, if a dialectical commitment’s social state is terminated, then on all paths emanating from $s$ in the next state, the commitment’s social state remains terminated.

$$AG \left( D_{soc}^T \rightarrow AX \; D_{soc}^T \right)$$

**Desirable Executions**

We desire those executions in which the dialectical commitments terminate in the following states.

• At a business level, it is acceptable if a dialectical commitment is never created in an execution or forever remains conditional. That is, the commitment remains in the objective state null or conditional.
• On an execution, a dialectical commitment may be socially asserted and objectively satisfied, \( \langle \text{satisfied, asserted} \rangle \). At a business level, such an execution is acceptable since the debtor is asserting a statement that is deemed objectively true.

• On an execution, a debtor may create a dialectical commitment asserting something which turns out to be false, that is, the dialectical commitment transitions to the state \( \langle \text{violated, asserted} \rangle \). In such a case, the debtor should cancel the commitment thus transitioning it to the state \( \langle \text{violated, terminated} \rangle \).

We now write a CTL specification to capture the above desirable termination states of dialectical commitments.

• On all paths in future, on all paths globally, a dialectical commitment’s objective state stays null or conditional, or its objective and social state becomes \( \langle \text{satisfied, asserted} \rangle \), or \( \langle \text{violated, terminated} \rangle \).

\[
\text{AF AG} \left( D_{\text{obj}}^N \lor D_{\text{obj}}^C \lor (D_{\text{obj}}^S \land D_{\text{soc}}^R) \lor (D_{\text{obj}}^V \land D_{\text{soc}}^T) \right)
\]

**NuSMV Module for Dialectical Commitments**

Similar to the NuSMV module for a practical commitment from Section 3.5.1, we develop a module for dialectical commitment. We employ this module is verifying business models that contain dialectical commitments. Listing 5.1 shows a fragment of the module.

```
1 MODULE dialectical(create, antecedent, consequent, antecedent_fail, consequent_fail, suspend, react, cancel)
2
3 CONSTANTS NULL, CONDITIONAL, EXPIRED, PENDING, DETACHED, SATISFIED,
VOLATED, TERMINATED, ASSERTED
4
5 DEFINE
6     objstatus := case
7     !create : NULL;
8     create & !ant & !con & !ant_fail & !con_fail :
9         CONDITIONAL;
10    create & !ant & !con & !ant_fail & con_fail :
11         CONDITIONAL;
12    create & ant & !con & !ant_fail & !con_fail :
13         DETACHED;
14    create & ant & !con & ant_fail & !con_fail :
15         EXPIRED;
```

Listing 5.1: NuSMV module for the dialectical commitment.
Line 1 declares the module with parameters of create, antecedent, consequent, antecedent failure, consequent failure, suspend, reactivate, and cancel. Lines 4–5 employ the CONSTANTS keyword to declare the commitment states as symbolic constants. Note that these include both objective and social states. Lines 7–26 compute the objective and social state of a dialectical commitment based on the input parameters. Lines 28–32 encode the CTL specification: $\text{AG}(D_{\text{obj}}^N \rightarrow AX D_{\text{obj}}^{N^\lor C \lor E \lor D \lor S \lor V})$.

### 5.2 Organizational Context

An organizational context (context, for short) is a distinguished agent that enjoys certain powers with respect to the debtor and creditor that enable it to help handle exceptions that may arise between them. Specifically, the context can entertain complaints of commitment violations, resolve disputes, impose sanctions, and potentially expel malfeasant participants. One can think of eBay as a canonical context that regulates the commitments of sellers and buyers to one another within the eBay marketplace. In the same vein, hospital sections and entire hospitals are contexts, the first nested within the second. In settings that include dedicated information infrastructure, the context can potentially monitor some of the interactions. For example, eBay knows exactly what price a buyer bids. And, a hospital knows what time a report was checked in. However, eBay doesn’t know whether the item sold was delivered. And, a hospital doesn’t know if the diagnosis provided by a pathologist is correct or whether the nurse injected the correct amount of a gadolinium-based compound.
An organizational context may also serve as an authority for resolving dialectical commitments. For example, a hospital could play the role of an authority shown in Figure 5.2.

5.3 Commitment Patterns

5.3.1 Service Provider with Correctness

Figure 5.3 illustrates the service provider with correctness pattern. A provider (1) practically commits to a client to bring about a consequent condition if some antecedent condition holds, and (2) dialectically commits that either the client, or in case of a disagreement between the client and the provider, a higher authority would agree with the consequent.

![Service provider with correctness pattern](image)

For example, a primary care physician practically commits to a patient to provide a diagnosis if the patient carries health insurance and keeps the necessary appointments. In addition, the physician dialectically commits to the correctness of the diagnosis.

5.3.2 Escalate

Figure 5.4 illustrates the escalation pattern. A service provider commits (C1) to a client to bring about a consequent condition if some antecedent condition holds under some organizational context:

\[ C_1 = C(\text{SERVICE PROVIDER}, \text{CLIENT}, \text{antecedent}, \text{consequent}) \]

The context commits to bringing about the creation of a commitment (C2) if the service provider violates its commitment, and the client escalates to the context. In C2, another service provider commits to the client to bring about the consequent. Additionally, the context may penalize the violating service provider or not, depending on the circumstances, some of which need not concern the client. For example, if a radiologist (service provider) fails to deliver mammography results to a physician (client), then the hospital (context) may get another radiologist (service provider') to perform mammography.
5.3.3 Chained Service Providers with Joint Correctness

Figure 5.5 illustrates the chained service providers with joint correctness pattern. This pattern contains a chain of service providers (the figure shows two): a service provider commits to the next in the chain to bring about a consequent if some antecedent holds. The head service provider commits to a client to bring about a consequent-1 if some antecedent-1 holds. Additionally, the head service provider dialectically commits to the client that either the client, or in case of a disagreement between the client and the provider, a higher authority would agree with the consequent-1, if the upstream providers (service provider 2 and 3) do not violate their dialectical commitment (D3). The upstream providers jointly dialectically commit to the head service provider that either the head service provider, or in case of a disagreement, a higher authority would agree with consequent-3. Such pattern is common in healthcare wherein a care provider would often rely on others for crucial information.

Figure 5.5: Chained service providers with joint correctness pattern.
5.4 Applying the Patterns

We apply the Comma methodology and the above patterns on the breast cancer screening and diagnosis scenario from Section 1.1.2 to produce a commitment-based model, as shown in Figure 5.6. We compare this model with the original published model in Figure 1.2.

5.4.1 Breast Cancer Screening and Diagnosis

Table 5.1 describes the subscenarios that we extract from the diagnosis scenario. Figure 5.6 shows the resulting business model with the patterns highlighted.

Next, we describe the Comma patterns used in Figure 5.6 for each of the subscenarios.

**Patient’s imaging and biopsy appointments:** (Unilateral commitments (C2, C3), highlighted in yellow) The patient commits to the physician to keep imaging (C2) and biopsy appointments (C3) if requested.

**Add patient to cancer registry:** (Unilateral commitments (C7, C8), highlighted in yellow) The pathologist commits to the hospital (C7) to reporting the patient to the registrar if the patient has cancer, and the registrar commits to the hospital (C8) to add the patient to the cancer registry.

**Patient’s radiology and pathology diagnosis:** (Chained service provider with joint correctness (C1, D1, C4, C6, D3), highlighted in green) The pathologist commits (C6) to the radiologist to provide a pathology report if the radiologist requests it and provides a tissue sample. The radiologist commits to the physician (C4) to provide an integrated radiology and pathology report if the physician requests it and the patient keeps the necessary appointment. The pathologist and the radiologist jointly and dialectically commit (D3) to the physician on the correctness of the integrated report. The physician commits to a patient (C1) to provide a diagnosis report if the patient requests for it and keeps necessary appointments. The physician dialectically commits (D1) to the correctness of the diagnosis report if the integrated radiology and pathology report is correct.

**Patient’s imaging:** (Service provider with correctness (C5, D2), highlighted in blue) The radiologist commits to the physician (C5) to provide imaging (mammography) results if the physician requests for it. In addition, the radiologist dialectically commits to the physician (D2) on the correctness of the imaging results.

**Escalate radiologist’s failure to provide imaging results:** (Escalate (C5, C9, C5’, D2’), highlighted in red) The hospital commits to the physician to bring about the creation of practical (C5’) and dialectical (D2’) commitments from an alternate radiologist if the original radiologist violates commitment C5 and the physician escalates.
Figure 5.6: The resulting commitment-based business model.
TUMOR BOARD provides input on a diagnosis: Unilateral commitments ($C_{10}$, $C_{11}$, $C_{12}$, $C_{13}$). The tumor board commits to the physician, radiologist, patient, and pathologist to provide its input on a diagnosis upon a request.

RADIOLOGIST and PATHOLOGIST guarantee their diagnoses: Dialectical commitments ($D_4$, $D_5$). The radiologist dialectically commits ($D_4$) to the pathologist that upon providing the radiology report, either the pathologist would agree with those results, or in the case of a disagreement, the tumor board will agree with those results. The pathologist makes similar commitment ($D_5$) to the radiologist regarding the pathology report.

5.4.2 Comparison with the Traditional Approach

Importantly, the Comma model of Figure 3 does not capture the internal activities of individual participants (e.g., interpret slides activity of pathologist) that Figure 1 shows. The internal activities of a participant are private to that participant, and therefore they ought not to appear in a cross-organizational model. Introducing internal activities in a cross-organizational model results in unnecessary tight coupling between the participants.

Figure 1 exemplifies the traditional workflow approach, which emphasizes message flows between collaborators and highlights the internal activities of each collaborator, but ignores the meanings of their interactions. Thus it tightly couples the internal activities of the collaborators, yielding a rigid model. By contrast, Comma captures the business relationships between the collaborators. Comma provides natural patterns, expressed in terms of commitments, to structure interactions.

Since Comma captures the business relationships among the participants, it provides a basis for answering some significant questions that a traditional workflow model leaves unanswered. The following are some such examples.

- What happens if the treatment plan turns out to be incorrect? Who is or are responsible?
  
  Note that the physician dialectically commits ($D_1$) to the patient for the correctness of the diagnosis report (treatment plan) only if the integrated radiology and pathology report is correct. If the integrated radiology and pathology report is incorrect, then $D_1$ never detaches, and the physician will not violate $D_1$, that is, the physician is not responsible for the incorrect diagnosis. The case in which physician violates $D_1$ is when the integrated radiology and pathology report is correct, but the diagnosis report is incorrect. If the integrated radiology and pathology report is deemed incorrect, then the radiologist and pathologist both violate their joint dialectical commitment $D_3$.

- Can the radiologist delegate her responsibility of conducting a mammography or a biopsy to another radiologist?
  
  The radiologist can readily delegate her commitments for conducting a mammography ($C_5$) or a biopsy ($C_4$) to another radiologist. For example, to delegate $C_5$, the radiologist may get another
radiologist to create the commitment $C(\text{Radiologist, Physician, Hospital, imagingRequested} \land \text{imagingAppointmentKept, imagingResultsReported})$. She can then suspend $C_5$, and reactivate it if the alternative radiologist fails to satisfy the delegated commitment.

- What happens if the radiologist fails to deliver the mammography results?

The hospital commits ($C_9$) to bringing about the creation of $C_5'$ and $D_2'$ from an alternative radiologist provided the original radiologist fails to deliver the mammography results, that is, the radiologist violates $C_5$. (This situation illustrates the physician escalating the violation of $C_5$ to the hospital.) The alternative radiologist now commits ($C_5'$) to providing mammography results, and to its correctness ($D_2'$).

We now summarize important benefits of Comma over workflows. Business meaning: Comma captures the business relationships among the parties that reflect business-level requirements such as responsibilities and exception handling, which workflows fail to capture.

**Business meaning:** Comma captures the business relationships among the parties that reflect business-level requirements such as responsibilities and exception handling, which workflows fail to capture.

**Flexibility:** Comma offers flexibility in enactment since it minimally constrains participants. Any execution in which all detached commitments are eventually satisfied is acceptable. For example, in Comma, the debtor of a commitment may itself act to satisfy a commitment, or it may delegate the commitment to another participant.

**Comprehensibility:** Because Comma only declaratively captures the business relationships, it produces more compact and comprehensible models than workflows, which become unwieldy and incomprehensible if they incorporate all possible exceptions and deviations.

### 5.5 Applying Verification

This section applies the verification approach from Section 3.4 to the breast cancer screening and diagnosis process. Figure 5.7 shows some of the sequence diagrams we generate for the Comma model from Figure 5.6. In Figure 5.7(a), Physician creates commitment $C_1$ toward Patient by either offering to provide diagnosis or agreeing to perform one. In Figure 5.7(b), Physician observes a suspicious lump and requests Patient to obtain imaging, which Patient agrees to do. By agreeing, Patient creates commitment $C_2$ toward Physician. In Figure 5.7(c), Radiologist requests Patient to arrive for imaging. Radiologist agrees to doing so and creates $C_5$. Patient arrives at Radiologist’s office for imaging, thus satisfying $C_2$ by keeping her imaging appointment. Finally, Radiologist sends the imaging report to Physician, which satisfies $C_5$, and creates $D_2$. 

74
In Figure 5.8, after Radiologist provides the imaging results, Physician either agrees with the results or, in case of a disagreement, requests Tumor Board to review them. Tumor Board either agrees with the imaging results, or disagrees with the results. The dialectical commitment $D_2$ satisfies if either Physician or Tumor Board agree with the result. In the case both Physician and Tumor Board disagree with the results, $D_2$’s objective state becomes violated. Subsequently, Radiologist agrees to modify the results as
per Tumor Board’s recommendation, that is, Radiologist cancels $D_2$. This transitions $D_2$’s social state to *terminated*. We present the remaining sequence diagrams in Appendix B.

![Sequence Diagram](image)

**Figure 5.8:** Physician gets an authority to review the imaging results.

On a computer with 2.66 GHz Intel Core 2 Duo processor, and 8 GB memory, NuSMV verified the breast cancer screening and diagnosis model in 0.2 seconds. The sequence diagrams from this section satisfy all the CTL specifications. Figure 5.9 shows a partial screenshot of NuSMV output showing that the model satisfies several CTL specifications. For example, the highlighted specification is NuSMV equivalent of: $\text{AG} (D_{\text{obj}}^N \rightarrow \text{AX} D_{\text{obj}}^N \lor E \lor D \lor S \lor V_{\text{obj}})$.

We consider a modified sequence diagram to demonstrate how our approach detects an error. From Figure 5.8, we remove the message from Radiologist to Physician agreeing to Tumor Board’s recommendation. In that case, NuSMV reports that the model fails to satisfy the CTL specification: $\text{AF AG} (D_{\text{obj}}^N \lor D_{\text{obj}}^C \lor (D_{\text{obj}}^S \land D_{\text{soc}}^R) \lor (D_{\text{obj}}^V \land D_{\text{soc}}^T))$ for $D_2$. Figure 5.10 shows a partial screenshot of the failure. NuSMV shows a counterexample in which $D_2$ stays in the state *(violated, asserted)*. That is, Radiologist does not agree with Tumor Board’s recommendation, and does not cancel $D_2$. 

76
5.6 Empirical Evaluation of Comma compared to HL7

We conducted a developer study that compared Comma with the healthcare standard HL7. This study followed similar design as of the previous study from Chapter 4 that compared Comma with RosettaNet. We conducted the study with 47 subjects. These subjects were graduate students in computer science. Several had prior industry experience as developers: 11 had more than 5 years; 27 had one to four years; and 9 had none.

The developer study used the breast cancer screening and diagnosis scenario from Section 1.1.2, and one modification of that scenario. The modified scenario added a clerk between Physician and the pathologist. Physician orders a pathology report from the clerk, and the clerk in turn orders a pathology report from a pathologist. After completing the diagnosis, the pathologist sends the report to the clerk, and the clerk in turn sends it to the radiologist. The modified scenario also added a health coach. If a patient is diagnosed with cancer, Physician assigns a health coach to her. The patient meets the health coach twice a month to receive counseling.

We performed the study in two phases. In the first phase, we partitioned the subjects into two groups, Group A and Group B. The subjects in Group A built sequence diagrams for the scenario given in Section 1.1.2 using HL7 messages and those in Group B did the same using Comma. Each subject worked
In the second phase, we partitioned the subjects into two new groups, Group C and Group D, each with half its members drawn from Group A and half from Group B. We developed HL7 and Comma solutions for the first phase. We provided Group C subjects our HL7 solution, and Group D subjects our Comma solution. Each subject worked individually to modify the provided model to accommodate the provided changes to the scenario.

5.6.1 Managing Threats

Our study design addresses the following threats to validity [Juristo and Moreno, 2001].

- It balances the subjects’ skill-sets by creating groups and subgroups with equal mean expertise (calculated based on their educational background, business process modeling, and software development experience, as obtained via a qualifying survey).

- It mitigates subjects’ learning effect by balancing Groups C and D with respect to experience in Phase 1 with HL7 and Comma. Moreover, it requires each subject to work individually without communicating with others.
• It eliminates the **instrumentation difference** by having subjects use the same tools; developing sequence diagrams using IBM RSA v8.0 and Comma models using an Eclipse-based tool Kalia et al. [2012] (download at: http://research.csc.ncsu.edu/maas/code/Protos).

### 5.6.2 Dependent Variables

We measure the following dependent variables for each approach, and compare them via statistical significance tests.

**Time:** (in minutes) taken to model sequence diagrams for the scenario.

**Difficulty** a subject perceives in modeling: an integer 1–5, interpreted as *extremely easy, easy, neutral, difficult*, and *extremely difficult*.

**Flexibility:** the number of executions a model permits. Greater flexibility in general leads to increased choices for a participant. We employ two measures of flexibility.

- **Sequence diagram count:** indicates modularity and generally yields greater numbers of interleavings of messages from multiple sequence diagrams.
- **Count of ALT, OPT, and PAR fragments:** indicates more numerous possible executions.

**Objective quality:** indicates the quality of a model based on the number of missing guards and the number of incorrect sequence diagrams structures.

- **Number of missing guards** indicates the count of incorrect enactments.
- **Number of incorrect sequence diagram structures** indicates the count of errors produced while modeling the messages.

**Subjective quality:** indicates the quality of a model based on the scenario coverage, precision, and comprehensibility.

- **Scenario coverage:**  *high* (covers the entire scenario),  *medium, low*, and *very low*.
- **Scenario precision:**  *high* (no unnecessary aspects),  *medium, low*, and *very low*.
- **Comprehensibility:**  *high* (easy for a human to comprehend),  *medium, low*, and *very low*.

Each subject submitted a worklog three times a week, reporting the time they spent and the difficulty they felt. We computed flexibility programmatically. Experts (the authors) judged the subjective quality.
5.7 Results of Empirical Evaluation

5.7.1 Flexibility and Objective Quality

Figure 5.11(a) shows boxplots for flexibility in terms of number of sequence diagrams and $\text{ALT}$, $\text{OPT}$, and $\text{PAR}$ fragments. The Y-axis represents the two study phases: (1) developing sequence diagrams (HL7 and Comma), and (2) modifying the sequence diagrams (HL7$_M$ and Comma$_M$). The X-axis shows the sum of the sequence diagram and $\text{ALT}$, $\text{OPT}$, and $\text{PAR}$ counts. Observe that the median number of sequence diagrams and $\text{ALT}$, $\text{OPT}$, and $\text{PAR}$ fragments is higher for Comma (11) than for HL7 (8.5). This suggests that the sequence diagrams designed using Comma are more modular and flexible than those designed using HL7. We attribute Comma’s higher modularity and flexibility to its focus on commitments along with its reusable patterns. Further, observe that the median number of sequence diagrams and $\text{ALT}$, $\text{OPT}$, and $\text{PAR}$ fragments for Comma$_M$ (27) is significantly higher than the median for HL7$_M$ (17). We attribute Comma$_M$’s higher values partially to the modular solution we provided that the subjects modified.

Figure 5.11(b) shows the objective quality of the sequence diagrams as assessed by domain experts. In Figure 5.11(b), missing guards and incorrect sequence diagrams have the same median (2) in HL7 and Comma. This may be the result of subjects using the Comma patterns without correctly adapting them to the given scenario. For example, several subjects failed to edit the guard when applying $\text{OPT}$ pattern fragments to the correct scenario-specific value, leaving it as the default value of $\text{true}$. Additionally, observe that the median number is zero for both HL7$_M$ and Comma$_M$, which we attribute to the quality of the solutions we provided in the second phase.
5.7.2 Subjective Quality

Figure 5.12 shows the subjective quality of the sequence diagrams judged independently by two experts. (The experts reconciled any differences through discussions.) Figure 5.12(a) shows the scenario coverage for the sequence diagrams. Observe that the scenario coverage is high for both HL7 (92%) and Comma (92%), and for the modification task, the scenario coverage for Comma\textsubscript{M} (82%) is slightly lower than HL7\textsubscript{M} (88%). The higher scenario coverage in both the approaches may be due to the scenario being small. Figure 5.12(b) shows the precision for the sequence diagrams. Observe that the Comma and Comma\textsubscript{M} precision (40% and 61%, respectively) is higher than that of HL7 and HL7\textsubscript{M} (18% and 40%, respectively). We attribute Comma’s higher precision to its systematic nature and the fact that it focuses attention on the relevant commitments. Figure 5.12(c) shows the comprehensibility of the models. The comprehensibility for Comma (32%) is higher than HL7 (14%), which we attribute to Comma’s modular patterns. Further, observe that the comprehensibility for HL7\textsubscript{M} (92%) is slightly higher than that of Comma\textsubscript{M} (89%).

5.7.3 Time and Difficulty

Figure 5.13 shows the time taken and difficulty perceived in developing the sequence diagrams using HL7 and Comma. Figure 5.13(a) shows that Comma and Comma\textsubscript{M} timings are (140 minutes and 112.5 minutes) less than HL7 and HL7\textsubscript{M} timings (210 minutes and 112.5 minutes). We attribute this result to Comma’s reusable patterns and HL7’s complexity. Figure 5.13(b) shows that the difficulty perceived in Comma and Comma\textsubscript{M} (3 and 2.5) is higher than HL7 and HL7\textsubscript{M} (3 and 2.3). We attribute this result to Comma’s emphasis on identifying multiple alternative executions that are acceptable at a business level.
5.7.4 Hypothesis Testing

H1 Comma yields higher mean sequence diagram count than HL7 suggesting that models are more modular using Comma than HL7.

H2 Comma yields higher mean ALT, OPT, and PAR count than HL7 which means Comma yields more flexible models than HL7.

H3 Comma takes lower mean time than HL7, which means that Comma is more efficient than HL7.

H4 Comma helps the subjects to perceive lower mean difficulty than HL7, which means that it is easy to adopt Comma than HL7.

H5 Comma produces fewer missing guards than HL7 thereby, which means that Comma produces higher quality sequence diagrams with fewer incorrect executions.

H6 Comma produces fewer incorrect sequence diagram structures than HL7, which means that Comma produces higher quality sequence diagrams than HL7.

For each hypothesis, we performed the unpaired two-tailed t-test to determine whether its null hypothesis is rejected at the 5% confidence interval. As Table 5.2 shows, the null hypothesis is rejected for ALT, OPT, and PAR, time, and incorrect sequence diagrams, but not rejected for all other measures.

5.8 Conclusion

The empirical study shows that Comma, a new approach for modeling healthcare processes in a declarative framework based on commitments, gives promising results in terms of both quality and modeling effort.
In particular, this means that the novelty of Comma is not a major impediment. However, improved tooling would help, especially in helping catch certain simple errors, such as missing guards. In future work, we will tackle additional realistic healthcare processes.

HL7 has contributed significantly to standardizing healthcare communications but the bulk of its contributions have been in information modeling and message schemas. Other work in healthcare IT has also concentrated on information modeling from a semantic standpoint, e.g., UMLS UMLS. Our effort makes the case that healthcare process modeling too needs as strong a semantic basis.

5.9 Related Work

Grando et al. [2010] design a catalog of run-time exceptions. They use a goal-based approach for dealing with normal and exceptional workflows, expressed as keystones connected by constraints from Petri Nets. In contrast, we provide a business model based on commitments between parties. We employ patterns to handle deviations from a normal path. For example, the escalation pattern handles violations between the parties by imposing sanctions.

Grando et al. [2011] support task delegation by taking into consideration the competence, responsibility, and accountability of each party performing a delegated task. Responsibility and accountability map naturally to commitments in Comma. It will be interesting to incorporate an agent’s capabilities that capture its competence in our approach.

Chen et al. [2008] provide an approach for verifying and repairing complicated processes such as for blood transfusion, where several parties are involved. They define processes using the Little-JIL [Cass et al., 2000] language. Such processes are at a low-level, specified using constructs such as sequencing, choice, and parallel. In contrast, our high-level business model can capture the essential elements of complex processes such as those for blood transfusion. Our business model can complement Little-JIL to provide correctness properties that can then be verified via techniques such as model-checking.

Fox et al. [2004] emphasize how to make goals explicit in the clinical process so as to achieve flexibility and adaptability in workflow systems. The above approaches assume that there is a unitary process and that the participants do as they are told. In contrast, our approach recognizes that the participants are autonomous. Instead of specifying their internal goals we specify their commitments to one another as a natural way to streamline their interactions and to support flexible enactment.

Müller et al. [2009] describe the importance of interoperability in healthcare processes. They focus only on data interoperability, and ignore business-level interoperability. The process participants must be interoperable at all levels. Our approach enables verifying business-level interoperability as demonstrated by Chopra and Singh [2008] and could be combined with a data-level approach.
Table 5.1: Subscenarios and patterns from the breast cancer diagnosis scenario.

<table>
<thead>
<tr>
<th>Subscenario</th>
<th>Description</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient’s imaging and biopsy appointments</td>
<td>When requested, the patient visits the radiologist for imaging and biopsy. The radiologist performs imaging and provides results to the physician.</td>
<td>Unilateral commitment</td>
</tr>
<tr>
<td>Patient’s imaging</td>
<td>The radiologist provides imaging and provides results to the physician.</td>
<td>Service provider with correctness</td>
</tr>
<tr>
<td>Patient’s radiology and pathology diagnosis</td>
<td>The radiologist and pathologist perform the diagnosis, and provide an integrated report to the physician. The physician provides a treatment plan to the patient based on the integrated report.</td>
<td>Chained service provider with joint correctness</td>
</tr>
<tr>
<td>Add patient to cancer registry</td>
<td>The pathologist reports a cancer patient to the registrar. The registrar adds the patient to a registry.</td>
<td>Unilateral commitment</td>
</tr>
<tr>
<td>Escalate radiologist’s failure to provide imaging results</td>
<td>If the radiologist fails to provide imaging results to the physician, the physician escalates the failure to the hospital. The hospital assigns an alternate radiologist to perform the imaging. (This subscenario is not in the original scenario; we add it to demonstrate how our approach handles an exception.)</td>
<td>Escalate</td>
</tr>
<tr>
<td>Tumor board provides input on a diagnosis</td>
<td>The physician, radiologist, patient, or pathologist may request the tumor board to provide its input on a diagnosis.</td>
<td>Unilateral commitment</td>
</tr>
<tr>
<td>Radiologist and pathologist guarantee their diagnoses</td>
<td>The radiologist guarantees her diagnosis to the extent that either the pathologist would agree with it, or in the case of a disagreement, the tumor board would agree with it. The pathologist provides similar guarantee to the radiologist regarding his report.</td>
<td>Dialectical commitment</td>
</tr>
</tbody>
</table>
Table 5.2: Hypothesis testing via Student’s t-test.

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>HL7 ($\mu_h$)</th>
<th>Comma ($\mu_c$)</th>
<th>Null Hypothesis [(\mu_h = \mu_c)] p-value</th>
<th>Accepted at p-value of 5%?</th>
</tr>
</thead>
<tbody>
<tr>
<td>[H1-Sequence Diagram]</td>
<td>3.47</td>
<td>4.56</td>
<td>0.1748</td>
<td>✓</td>
</tr>
<tr>
<td>[H2-ALT+OPT+PAR]</td>
<td>4.05</td>
<td>9.96</td>
<td>0.0005</td>
<td>×</td>
</tr>
<tr>
<td>[H3-Time]</td>
<td>352.14</td>
<td>190.00</td>
<td>0.6434</td>
<td>×</td>
</tr>
<tr>
<td>[H4-Difficulty]</td>
<td>2.77</td>
<td>2.69</td>
<td>0.7044</td>
<td>✓</td>
</tr>
<tr>
<td>[H5-Missing Guard]</td>
<td>1.14</td>
<td>1.36</td>
<td>0.5999</td>
<td>✓</td>
</tr>
<tr>
<td>[H6-Incorrect Sequence Diagrams]</td>
<td>1.32</td>
<td>0.54</td>
<td>0.0221</td>
<td>×</td>
</tr>
</tbody>
</table>
Chapter 6

Commitments and Goals

This chapter presents our combined operational semantics for goals and commitments. It is based on my work in collaboration with Neil-Yorke Smith [Telang et al., 2012].

6.1 Operational Semantics

As we have observed, whereas a goal is specific to an agent (but see Section 6.7), a commitment involves a pair of agents. On the one hand, an agent may create commitments toward other agents in order to achieve its goals. On the other hand, an agent may consider goals in order to fulfil its commitments to other agents.

Chopra et al. (2010a) formalize a semantic relationship between commitments and goals. They write goal conditions in either or both of the antecedent or consequent of a commitment. For example, $C(x, y, g_1, g_2)$ is a commitment with debtor $x$, creditor $y$, antecedent $g_1$, and consequent $g_2$, where $g_i$ are (propositional) goal conditions. As an example, consider how a car insurer may commit to a repair garage to paying if the latter performs a repair: $C($INSURER, REPAIRER, car_repaired, payment_made$)$. Here, car_repaired is the success condition of the insurer’s goal. The insurer may consider a goal with the success condition of payment_made to satisfy the commitment.

In general, the antecedent and consequent of a commitment can, of course, refer to goals and commitments explicitly. However, it would be conceptually unclear to posit a commitment whose antecedent or consequent is a goal, since a commitment captures a public relationship between two agents whereas a goal captures a private state of one of the agents. Therefore, we consider commitments whose antecedent and consequent are objective conditions, which might also be the success conditions of one or more goals—this agrees with Chopra et al.’s representation.

We consider agents who follow a simple architecture, as depicted in Figure 6.1. We discuss the architecture informally here and formalize it below. Each agent maintains a set of beliefs, goals, commitments, goal action events, and commitment action events. Note that actions are initiated by the agent, and events
are from the environment. Goals and commitments are shown in the figure; action events are not shown. The agent executes iteratively in a control loop. Based on its perception of the environment, the agent updates its beliefs and updates the goal and commitment states. The agent then executes the practical reasoning rules that apply.

The practical rules apply according to the state of a commitment, a goal, or a commitment–goal pair. For each commitment and each goal, the agent selects at most one of the applicable practical rules to execute. Each selected practical rule yields exactly one action; the agent selects at most one action for each commitment and each goal. For example, the agent is not allowed to simultaneously fire two practical rules on a commitment, one to cancel and the other to suspend it. Each selected action corresponds to at most one transition in the life cycle of each commitment and each goal and results in updating the state of any affected commitment or goal. All such transitions are executed in parallel.

Finally, in addition to modifying its own state, the agent modifies the environment by sending messages. In a message, the agent communicates a commitment operation, or bringing about an objective condition. The control loop then repeats with the next perception.

6.1.1 Assumptions

Here we collate and itemize the assumptions we make.

- We consider commitments whose antecedents and consequents are propositions, except for allowing nested commitments in the consequent and the antecedent. This assumption simplifies the presentation and allows us to focus on the essentials of the semantics. As stated earlier, this means, in mapping from informal examples, we would (1) parameterize the antecedent and consequent, and (2) instantiate such parameters when we instantiate such a commitment.

- We disregard timeouts, and commitment delegation or assignment, and the notion of an organizational context, since they are not essential to our present contribution. Timeouts, and indeed temporal commitments and goals, add both realism and complexity (see, for instance, Marengo et al. [2011]). Commitment delegation and assignment applies when one considers more than two agents (see Section 8.2). The organizational context is an important consideration and considering commitments and goals in such a larger context is a significant area. All three are ripe for future investigation.

- We consider only achievement goals, reserving maintenance goals for future work. Achievement goals are the most common form of goals in the literature.

- As in most previous works on commitments (except Chopra and Singh [2009]), we assume for simplicity that the agents communicate synchronously.
Synchronous communication simplifies alignment: each commitment is represented in the same state by both its debtor and creditor. This assumption, while standard in the agents literature, is significant and has consequences on the realization of a multiagent system. Lifting the assumption, however, is not trivial and is a relevant topic for future works.

- Again, for simplicity of presentation and to restrain scope of this article, we limit the presentation of our semantics to the situation of a single commitment and a single (end) goal, and to two agents. Section 8.2 discusses the situation of multiple goals between two or more agents.

### 6.1.2 Formalization

In this subsection we formulate the concepts over which the rules of our operational semantics operate. The rules themselves follow in the subsequent subsection. Thus, these sections are at the heart of our contribution.
We consider the configuration or state of an agent $x$ as the tuple $S_x = \langle B, G, C, A^G, A^C \rangle$ where $B$ is its set of beliefs, $G$ its set of goals, $C$ its set of commitments, $A^G$ is a set of tuples (goal, goal action or event), and $A^C$ is a set of tuples (commitment, commitment action or event). Conceptually, an agent’s configuration relates to elements of both its cognitive state and the relevant components of the social state (commitments of which the agent is creditor or debtor). That is, an agent’s configuration incorporates its beliefs and goals as well as its commitments. In our approach, the notional social state is not stored independently of the agents—that is, it exists only in terms of its projections in the various agents. For simplicity, we assume the agents communicate synchronously. Thus the projections of the different agents of the social state remain mutually consistent. We adopt a standard propositional logic. Table 6.1 gives the formal syntax for the agent configuration (beliefs, commitments, and goals) in Backus-Naur Form.

Table 6.1: Syntax for beliefs, commitments, and goals in Backus-Naur Form.

| $S$ : AgentState | $\rightarrow \langle$ Beliefs, Goals, Commitments, GoalActionEvents, CommitActionEvents $\rangle$ |
| $B$ : Beliefs | $\rightarrow$ Belief $\langle$, Belief $\rangle^*$ |
| $G$ : Goals | $\rightarrow$ Goal $\langle$, Goal $\rangle^*$ |
| $C$ : Commitments | $\rightarrow$ Commitment $\langle$, Commitment $\rangle^*$ |
| $A^G$ : GoalActionEvents | $\rightarrow$ GoalActionEvent $\langle$, GoalActionEvent $\rangle^*$ |
| GoalActionEvent | $\rightarrow$ (Goal, GAction $|$ GEvent) |
| $A^C$ : CommitActionEvents | $\rightarrow$ CommitActionEvent $\langle$, CommitActionEvent $\rangle^*$ |
| CommitActionEvent | $\rightarrow$ (Commitment, CAction $|$ CEvent) |
| Belief | $\rightarrow$ B(Agent, atom) |
| Goal | $\rightarrow$ CState(Agent, Commitment | atom, atom) |
| Commitment | $\rightarrow$ CState(Agent, Agent, Commitment | atom, Commitment | atom) |
| Agent | $\rightarrow$ name |
| GState | $\rightarrow$ N | I | A | S | T | F | S |
| CState | $\rightarrow$ N | C | E | D | P | T | V | S |
| GAction | $\rightarrow$ consider | activate | suspend | reconsider | reactivate | drop | abort |
| GEvent | $\rightarrow$ fail | succeed |
| CAction | $\rightarrow$ create | suspend | reactivate | cancel | release |
| CEvent | $\rightarrow$ antecedent_failure | antecedent | consequent |

In detail, the components of agent configuration are as follows:

- $B$ is the set of agent $x$’s beliefs, of the form $B(x, p)$, about the current snapshot of the world, and may include beliefs about itself and other agents.

- $C$ is a set of commitments, of the form $C^t(x, y, s, u)$, where $x$ and $y$ are agents, $s$ and $u$ are either commitments or propositions, and $t$ is the state of the commitment. Figure 2.1 shows the possible values of $t$ (e.g. $D$ represents the state detached). For a given agent, this set contains those
commitments in which the agent either features as the debtor or as the creditor. For example, the
commitment $C^t(x, y, s, u)$ will be in agent $x$’s and agent $y$’s set of commitments, $\mathcal{C}$.

- $\mathcal{G}$ is a set of goals adopted by $x$, of the form $G^t(x, s, f)$, where $s$ (success condition) is either a
  commitment or a proposition, $f$ (failure condition) is a proposition, and $t$ is the state of the goal. Figure 2.2 shows the possible values of $t$ (e.g. $U$ represents the state suspended). $\mathcal{G}$ includes all of
  $x$’s goals, i.e., whether they are in state Inactive, Active, or Suspended. Since the goals in $\mathcal{G}$ are
  adopted, we take it that they are mutually consistent [Winikoff et al., 2002]. Note that $\mathcal{G}$ does not
  include goals that are in a terminal state.

- $\mathcal{A}_C$ is a set of tuples $\langle \text{Commitment, CAAction|CEvent} \rangle$. It maps a commitment to a commitment
  action or a commitment event: create, antecedent, consequent, suspend, reactivate, cancel, release,
  and antecedent_failure. An agent may employ a practical rule to set the action for a commitment.
  Alternately, the environment may set an event for the commitment. If an action or event is set for a
  commitment, then the commitment transitions to a new state as per the commitment life cycle.

- $\mathcal{A}_G$ is a set of tuples $\langle \text{Goal, GAction|GEvent} \rangle$. It maps a goal to a goal action or a goal event:
  consider, activate, suspend, reconsider, reactivate, drop, abort, fail, and succeed. Similar to a
  commitment action, an agent may employ a practical rule to set the action for a goal, or the
  environment may set an event for the goal. If an action or event is set for a goal, then the goal
  transitions to a new status as per the goal life cycle.

For clarity, notice that goals and operations on them are private to each agent: no agent may inspect
the goals of another. Each commitment is represented privately as well. However, a commitment being
an element of the social state, each commitment is represented by both its creditor and its debtor. The
rules we discuss apply to each agent’s internal representation separately.

The agents do not agree upon any actions. Each agent affects its goals and commitments unilaterally;
no agent can commit another agent [Singh, 2012]. Agents communicate regarding commitments. For
simplicity, we elide communication details: imagine each commitment operation as a synchronous
message. Synchrony prevents race conditions between agents affecting the same commitment. The
additional challenges of asynchrony are out of scope for our present topic, but see Chopra and Singh

We assume that all agents adopt the common operational semantics (life cycles) for goals and
commitments. That is, the different agents follow the same life cycle representation, which reflects the
core semantics of commitments and goals. Practical rules may differ across agents, because an agent’s
practical rules reflect its decision-making.

We capture the operational semantics of reasoning about goals and commitments via inference
rules. Each such rule represents that an agent can conclude the configuration below the line from the
configuration above the line.
6.1.3 Practical Rules

The life cycle of a commitment or a goal (as shown in Figures 2.1 and 2.2, respectively) specifies its progression in light of significant events, such as its creation and satisfaction. For example, if \( f \) holds, a goal whose failure condition is \( f \) would be considered as having Failed. In other words, the life cycle captures the hard integrity requirements on our formulation of goals and commitments. The agent may choose as it pleases but if it chooses to create a commitment, the commitment must be created.

In contrast to life cycle diagrams, practical (reasoning) rules capture not the necessary integrity requirements, but rather patterns of pragmatic reasoning that agents may or may not adopt under different circumstances. In that sense, rather than describing the mechanics of commitment and goal transitions, practical rules are the rules of an agent program. They are specified by the designers of the agents.

The practical rules may be neither complete nor deterministic: an agent may find itself at a loss as to how to proceed or may find itself with multiple options. Such nondeterminism corresponds naturally to a future-branching temporal model: each agent’s multiplicity of options leads to many possible progressions of its configuration and of the configurations of its peers. The convergence results we show below indicate that our formulated set of rules is complete (i.e., sufficient) in a useful technical sense.

Recall that the practical rules are merely options that an agent has available when it adopts these rules as patterns of reasoning. That is, the agent designer provides the agent with (some of) the practical rules at design time. At runtime, the agent can choose which, if any, of matching rules to execute in a given situation. An agent may refine these rules to always select from among a narrower set of the available options, for example, through other reasoning about its preferences and utilities. Our approach supports such metareasoning capability in principle, but we defer a careful investigation of it to future research.

A practical rule is a decision-making rule because it involves the agent deciding to perform an action. Such a rule, as shown next, is characterized by introducing an action into the agent’s \( \mathcal{A}^C \) or \( \mathcal{A}^G \) component. Consider a rule that creates a commitment \( C(x, y, s, u) \) if a goal \( G(x, s, f) \) is active. Agent \( x \) may choose this rule if it lacks capability for \( s \), or due to some other economic motivation. Equation 6.1 shows this rule.

\[
\begin{align*}
G^A(x, s, f) \in \mathcal{G} & \quad C(x, y, s, u) \notin \mathcal{C} & \mathcal{A}_2^C = \mathcal{A}_1^C \cup \{ (C^N(x, y, s, u), create) \} \\
\langle B, G, C, A^G, A^C \rangle & \rightarrow \langle B, G, C, A^G, A_2^C \rangle
\end{align*}
\] (6.1)

In Equation 6.1, the guard is on the top of the line. In this rule, the guard requires goal \( G^A(x, s, f) \) to be in \( x \)’s set of goals \( \mathcal{G} \), and commitment \( C(x, y, s, u) \) not to be in \( x \)’s set of commitments \( \mathcal{C} \). If the guard is true, then the transition shown below the line occurs. \( x \)’s configuration transitions from a state in which \( G^A(x, s, f) \) is in \( \mathcal{G} \), and \( C \) is not in \( \mathcal{C} \), to a state in which the create action applied to \( C^N(x, y, s, u) \) is added in the set of commitment actions and events \( \mathcal{A}^C \). For convenience and clarity, we abbreviate the rule from Equation 6.1 to Equation 6.2.
In Equation 6.2, $G$ is the goal $G(x, s, f)$, and $C$ is the commitment $C(x, y, s, u)$. The superscript indicates the state of a goal or commitment; $G^A$ means that $G$ is active, $C^N$ means that $C$ is in state null. Further, we employ conjunction or disjunction in the superscript. For example, $G^{A\lor U}$ means $G^{A} \lor G^{U}$, that is, $G^{A}(x, s, f) \in \mathcal{G}$ or $G^{U}(x, s, f) \in \mathcal{G}$. Note that actions are produced by practical rules and consumed by making transitions in the life cycles, as shown earlier in Figure 6.1.

We now introduce some additional notation that practical rules use. Consider an agent $x$ having a goal $G_{end} = G(x, s, f_1)$, where $x$ lacks (or prefers not to exercise) the capability to bring about $s$. However, $x$ can bring about $u$, and another agent $y$ can bring about $s$. We say that $C = C(x, y, s, u)$ is an offer in which $x$ commits to $y$ to bring about $u$ if $s$ holds. $G_{means} = G(y, s, f_2)$ is $y$’s goal with a success condition that is the same as that of $G_{end}$, and $G_{dis} = G(x, u, f_3)$ is $x$’s goal with a success condition that is the same as $C$’s consequent. We refer to $G_{end}$ as the end goal, $G_{means}$ as the means goal, and $G_{dis}$ as the discharge goal for commitment $C$.

The practical rules are designed to engender coherence between a pair of the components ($C$, $G_{end}$, $G_{means}$ and $G_{dis}$). When the driver component changes state, a rule acts upon the follower component to restore coherence. Specifically, $G_{end}$ is the driver component for $C$, and $C$ is the driver for $G_{means}$ and $G_{dis}$. For example, if $G_{end}$ is suspended, then a rule suspends $C$, and if $G_{end}$ is terminated, then a rule cancels $C$. In another example, if $C$ is created, then a rule considers $G_{means}$, and if $C$ is terminated, then a rule terminates $G_{means}$.

In developing the practical rules, we only consider one commitment and three related goals (end, means, discharge). Our practical rules are generic in that they can be repeatedly applied in case of multiple related commitments and goals. However, we leave it as a future work to formally apply and evaluate our approach on multiple commitments and goals as discussed in Section 8.2.

**Practical Rules: From End Goal to Commitment**

This section presents $x$’s practical reasoning rules that involve its end goal $G_{end}$, and the offer (commitment) $C$. As noted earlier, we design the practical reasoning rules to engender coherence between $G_{end}$ and $C$. The antecedent of each of these rules is a configuration that combines the state of the end goal, and the state of the offer.

- **ENTICE**: If the end goal ($G_{end}$) is active and the offer ($C$) is null, $x$ creates an offer ($C$) to another agent ($y$).

\[
\frac{\langle G^A_{end}, C^N \rangle}{create(C)}
\]

**Motivation**: (Only) by creating the commitment can $x$ satisfy its goal ($G_{end}$).
• **SUSPEND OFFER:** If the end goal \((G_{\text{end}})\) is suspended, then \(x\) suspends the offer \((C)\).

\[
\langle G_{\text{end}}^U, C^A \rangle \quad \text{suspend}(C) \tag{6.4}
\]

*Motivation:* By suspending the offer \((C)\), \(x\) indicates to \(y\) that \(y\) may employ its resources in other tasks instead of working on the offer \((C)\).

• **REVIVE:** If \(G_{\text{end}}\) is active or satisfied, and \(C\) is pending, then \(x\) reactivates \(C\).

\[
\langle G_{\text{end}}^{A\lor S}, C^P \rangle \quad \text{reactivate}(C) \tag{6.5}
\]

*Motivation:* Agent \(x\) needs an active commitment \((C)\) to satisfy its active end goal \((G_{\text{end}})\). If the end goal is satisfied, then agent \(x\) needs to reactivate the pending commitment to be able to satisfy the commitment.

• **WITHDRAW OFFER:** If \(G_{\text{end}}\) fails or is terminated, then \(x\) cancels \(C\).

\[
\langle G_{\text{end}}^{T\lor F}, C^A \rangle \quad \text{cancel}(C) \tag{6.6}
\]

*Motivation:* The offer \((C)\) is of no utility once the end goal \((G_{\text{end}})\) for which it is created no longer exists.

• **REVIVE TO WITHDRAW:** If \(G_{\text{end}}\) fails or is terminated and \(C\) is pending, then \(x\) reactivates \(C\).

\[
\langle C_{\text{end}}^{T\lor F}, C^P \rangle \quad \text{reactivate}(C) \tag{6.7}
\]

*Motivation:* If the end goal \((G_{\text{end}})\) fails or is terminated, and the offer \((C)\) is pending, then \(x\) reactivates the offer \((C)\), and later cancels it by the virtue of WITHDRAW OFFER. As the commitment life cycle in Figure 2.1 shows, an agent needs to reactivate a commitment before canceling it.

• **NEGOTIATE:** If \(C\) terminates or expires, and \(G_{\text{end}}\) is active or suspended, then \(x\) creates another commitment \(C'\) to satisfy its goal.

\[
\langle G^{A\lor U}, C^E^\lor T \rangle \quad \text{create}(C') \tag{6.8}
\]

*Motivation:* \(x\) persists with its end goal \((G_{\text{end}})\) by trying alternative ways to induce other agents to cooperate.
• **ABANDON END GOAL**: If $C$ terminates or expires, then $x$ gives up on $G_{\text{end}}$.

\[
\frac{\langle G_{\text{end}}^{A\lor U}, C^{E\lor T} \rangle}{\text{drop}(G_{\text{end}})}
\] (6.9)

**Motivation**: $x$ may decide no longer to persist with its end goal ($G_{\text{end}}$).

In the above practical rules, the antecedent is a condition that contains $G_{\text{end}}$’s state and $C$’s state, and the consequent is an action either on $G_{\text{end}}$ or on $C$. Next, we present the practical rules in which the antecedent contains only $G_{\text{end}}$’s state. An agent may execute such a rule because of a motivation that is outside the scope of our reasoning. For example, an agent may drop an end goal ($G_{\text{end}}$) if achieving it is no longer beneficial, or if the agent concludes that the goal is no longer achievable.

• **DISCARD END GOAL**: $x$ may drop or abort $G_{\text{end}}$ when it is inactive, active, or suspended.

\[
\frac{\langle G_{\text{end}}^{I\lor A\lor U} \rangle}{\text{drop}(G_{\text{end}})}
\] (6.10)

\[
\frac{\langle G_{\text{end}}^{I\lor A\lor U} \rangle}{\text{abort}(G_{\text{end}})}
\] (6.11)

• **CONSIDER END GOAL**: $x$ may consider $G_{\text{end}}$ when it is null.

\[
\frac{\langle G_{\text{end}}^{N} \rangle}{\text{consider}(G_{\text{end}})}
\] (6.12)

• **ACTIVATE END GOAL**: $x$ may activate $G_{\text{end}}$ when it is inactive.

\[
\frac{\langle G_{\text{end}}^{I} \rangle}{\text{activate}(G_{\text{end}})}
\] (6.13)

• **SUSPEND END GOAL**: $x$ may suspend $G_{\text{end}}$ when it is inactive or active.

\[
\frac{\langle G_{\text{end}}^{I\lor A} \rangle}{\text{suspend}(G_{\text{end}})}
\] (6.14)

• **RECONSIDER END GOAL**: $x$ may reconsider $G_{\text{end}}$ when it is suspended.

\[
\frac{\langle G_{\text{end}}^{U} \rangle}{\text{reconsider}(G_{\text{end}})}
\] (6.15)
• **REACTIVATE END GOAL:** $x$ may reactivate $G_{end}$ when it is suspended.

\[
\begin{align*}
\langle G_{end}^U \rangle \\
\text{reactivate}(G_{end})
\end{align*}
\]

(6.16)

• **END GOAL FAILS:** $G_{end}$ may fail when it is inactive, active, or suspended.

\[
\begin{align*}
\langle G_{end}^{I\lor A\lor U} \rangle \\
\text{fail}(G_{end})
\end{align*}
\]

(6.17)

Since agent $x$ does not satisfy the end goal on its own (and depends upon $y$ to satisfy the means goal), there is no rule in which the end goal succeeds. Further, note that the above set of practical rules only considers the possible combinations of the goal and commitment states. For example, the state $\langle G_{end}^A, C^V \rangle$ is not possible since $C$ can be violated only after $G_{end}$ has satisfied; hence no rule is required that applies in $\langle G_{end}^A, C^V \rangle$.

**Practical Rules: From Commitment to Means Goal**

This section presents practical reasoning rules for agent $y$, whom agent $x$ attempts to engage. The rules involve the means goal $G_{means}$ and the offer $C$, as introduced above. Note that $y$ creates $G_{means}$ to detach $C$. The rules ensure that $G_{means}$ follows $C$’s life cycle.

• **DETACH:** If $G_{means}$ is null and $C$ is conditional, then $y$ considers and activates goal $G_{means}$ to bring about $C$’s antecedent.

\[
\begin{align*}
\langle G_{means}^N, C^C \rangle \\
\text{consider}(G_{means})
\end{align*}
\]

(6.18)

**DETACH’:** If $G_{means}$ is inactive and $C$ is conditional, then $y$ activates goal $G_{means}$ to bring about $C$’s antecedent.

\[
\begin{align*}
\langle G_{means}^I, C^C \rangle \\
\text{activate}(G_{means})
\end{align*}
\]

(6.19)

**Motivation:** The creditor brings about the antecedent of a commitment hoping to influence its debtor to discharge the commitment.

• **BACK BURNER:** If $G_{means}$ is active and $C$ is pending, then $y$ suspends $G_{means}$.

\[
\begin{align*}
\langle G_{means}^A, C^P \rangle \\
\text{suspend}(G_{means})
\end{align*}
\]

(6.20)

**Motivation:** By suspending the goal, the agent may employ its resources to work on other goals.
• **FRONT BURNER:** If $G_{\text{means}}$ is suspended and $C$ is conditional, then $y$ reactivates $G_{\text{means}}$.

$$\langle G^U_{\text{means}}, C^C \rangle \quad \text{reactivate}(G_{\text{means}})$$ (6.21)

*Motivation:* An active means goal is necessary for the agent to detach the commitment.

• **ABANDON MEANS GOAL:** If $G_{\text{means}}$ is active and $C$ expires or terminates (either $x$ cancels, or $y$ releases $x$ from $C$), then $y$ drops $G_{\text{means}}$.

$$\langle G^A_{\text{means}}, C^{E\lor T} \rangle \quad \text{drop}(G_{\text{means}})$$ (6.22)

*Motivation:* The goal is not needed since the commitment for which it is created no longer exists.

• **PERSIST (MEANS):** If $G_{\text{means}}$ fails or terminates and $C$ is conditional, then $y$ considers goal $G_{\text{means}}'$ identical to $G_{\text{means}}$ but potentially with a different failure condition. Note that an autonomous and rational agent may possibly try again by activating an identical goal: for example, if you don’t win a game, you may play again.

$$\langle G^{T\lor F}_{\text{means}}, C^C \rangle \quad \text{consider}(G_{\text{means}}')$$ (6.23)

• **PERSIST’ (MEANS):** If $G_{\text{means}}'$ is inactive and $C$ is conditional, then $y$ activates goal $G_{\text{means}}'$.

$$\langle G^{T\lor F}_{\text{means}}, G_{\text{means}}', C^C \rangle \quad \text{activate}(G_{\text{means}}')$$ (6.24)

*Motivation:* The agent persists in pursuing to detach the commitment.

• **GIVE UP:** If $G_{\text{means}}$ fails or terminates and $C$ is conditional, $y$ releases $x$ from $C$.

$$\langle G^{T\lor F}_{\text{means}}, C^C \rangle \quad \text{release}(C)$$ (6.25)

*Motivation:* The agent gives up pursuing its commitment by releasing the debtor from its responsibilities in the commitment.

We now present the practical rules in which the antecedent contains only $G_{\text{means}}$’s state. Similar to the end goal, an agent may execute such a rule because of a motivation that is outside the scope of our reasoning.
• RECONSIDER MEANS GOAL: $y$ may reconsider $G_{means}$ when it is suspended.

\[
\frac{\langle G^U_{means} \rangle}{\text{reconsider}(G_{means})} \quad (6.26)
\]

• REACTIVATE MEANS GOAL: $y$ may reactivate $G_{means}$ when it is suspended.

\[
\frac{\langle G^U_{means} \rangle}{\text{reactivate}(G_{means})} \quad (6.27)
\]

• MEANS GOAL FAILS: $G_{means}$ may fail due to a reason outside of the $y$’s control.

\[
\frac{\langle G^{I\lor A\lor U}_{means} \rangle}{\text{fail}(G_{means})} \quad (6.28)
\]

• MEANS GOAL SUCCEEDS: $y$ may bring about the success condition $s$ of $G_{means}$ when it is inactive, active, or suspended.

\[
\frac{\langle G^{I\lor A\lor U}_{means} \rangle}{\text{succeed}(G_{means})} \quad (6.29)
\]

Observe that there are no rules with only $G_{means}$’s state in the antecedent for considering, activating, suspending, reactivating, or dropping the means goal. This is because the means goal is considered, activated, suspended, reactivated, or dropped only in response to the commitment via the rules 6.18, 6.19, 6.20, and 6.21, and 6.22 respectively. Further, note that unlike the end goal, there is a rule in which agent $y$ satisfies the means goal.

Practical Rules: From Commitment to Discharge Goal

Next, we present $x$’s practical reasoning rules that involve the discharge goal $G_{dis}$ and the offer $C$. Agent $x$ creates $G_{dis}$ to discharge the detached offer $C$. The rules ensure that $G_{dis}$ follows $C$’s life cycle.

• DELIVER: If $G_{dis}$ is null and $C$ is detached, then $x$ considers goal $G_{dis}$ to bring about $C$’s consequent.

\[
\frac{\langle G^N_{dis}, C^D \rangle}{\text{consider}(G_{dis})} \quad (6.30)
\]

DELIVER’: If $G_{dis}$ is inactive and $C$ is detached, then $x$ activates goal $G_{dis}$ to bring about $C$’s consequent.

\[
\frac{\langle G^I_{dis}, C^D \rangle}{\text{activate}(G_{dis})} \quad (6.31)
\]
Motivation: The agent is honest in that it activates a discharge goal that would lead to discharging its commitment.

- **BACK BURNER**: If $G_{\text{dis}}$ is active and $C$ is pending, then $x$ suspends $G_{\text{dis}}$.

$$\langle G^A_{\text{dis}}, C^P \rangle \xrightarrow{\text{suspend}} G_{\text{dis}} \quad (6.32)$$

Motivation: By suspending the goal, the agent may employ its resources to work on other goals.

- **FRONT BURNER**: If $G_{\text{dis}}$ is suspended and $C$ is detached, then $x$ reactivates $G_{\text{dis}}$.

$$\langle G^U_{\text{dis}}, C^D \rangle \xrightarrow{\text{reactivate}} G_{\text{dis}} \quad (6.33)$$

Motivation: An active discharge goal is necessary for the agent to satisfy the commitment.

- **ABANDON DISCHARGE GOAL**: If $G_{\text{dis}}$ is active and $C$ terminates ($y$ releases $x$ from $C$) or violates ($x$ cancels $C$), then $x$ drops $G_{\text{dis}}$.

$$\langle G^A_{\text{dis}}, C^{T\lor V} \rangle \xrightarrow{\text{drop}} G_{\text{dis}} \quad (6.34)$$

Motivation: The discharge goal is not needed since the commitment for which it is created no longer exists.

- **PERSIST (DISCHARGE)**: If $G_{\text{dis}}$ fails or terminates and $C$ is detached, then $x$ considers goal $G_{\text{dis}}'$ identical to $G_{\text{dis}}$.

$$\langle G^T_{\text{dis}} \lor F, C^D \rangle \xrightarrow{\text{consider}} G_{\text{dis}}' \quad (6.35)$$

- **PERSIST’ (DISCHARGE)**: If $G_{\text{dis}}'$ is inactive and $C$ is detached, then $x$ activates goal $G_{\text{dis}}'$.

$$\langle G^T_{\text{dis}} \lor F, G_{\text{dis}}', I, C^D \rangle \xrightarrow{\text{activate}} G_{\text{dis}}' \quad (6.36)$$

Motivation: The agent persists in pursuing to satisfy the commitment.

- **GIVE UP**: If $G_{\text{dis}}$ fails or terminates and $C$ is detached, then $x$ cancels $C$.

$$\langle G^T_{\text{dis}} \lor F, C^D \rangle \xrightarrow{\text{cancel}} (C) \quad (6.37)$$

Motivation: The agent gives up pursuing to satisfy its commitment by canceling the commitment,
thereby violating the commitment. The agent may be better off violating the commitment and fulfilling any sanctions compared to repeatedly failing at satisfying the commitment.

The practical rules in which the antecedent contains only $G_{dis}$’s state are identical to such rules for $G_{means}$; we refrain from writing them out again here. We next prove properties of the whole set of practical rules that show their potential benefit for an agent designer.

### 6.1.4 Nested Commitments

As Table 6.1 shows, our grammar allows a commitment to be nested in a goal, and in another commitment. Our practical rules equally apply to the cases involving nested goals and commitments. In particular, the antecedent or consequent of a commitment may be another commitment.

Suppose an agent $x$ has an end goal: $G_{end} = G(x, C(y, x, u, v))$. Notice that $x$ cannot create commitment $C_{nested} = C(y, x, u, v)$ since its debtor is another agent $y$, and therefore, $x$ cannot satisfy $G_{end}$ goal on its own. In such a case, $x$ may employ ENTICE rule to create nested commitment: $C = C(x, y, C(y, x, u, v), q)$. Then, agent $y$ may employ DETACH and DETACH’ rules to consider and activate the goal: $G_{means} = G(y, C(y, x, u, v))$, respectively. Agent $y$ may satisfy $G_{means}$ (and, therefore, satisfy $G_{end}$, and detach $C$) by creating $C_{nested}$. Then, agent $x$ may employ DELIVER and DELIVER’ rules to consider and activate goal: $G_{dis} = G(x, q)$, respectively. Agent $x$ may satisfy $G_{dis}$ (and, therefore, satisfy $C$) by bringing about its success condition $q$. Similarly, agents $x$ and $y$ can employ our practical rules for the nested commitment: $C_{nested} = C(y, x, u, v)$.

Note that although our rules apply to all possible nestings of commitments, some of the nestings are pragmatically ill-formed. Next, we describe all possible nestings of commitments, and whether they are ill-formed.

- $C(x, y, C(x, y, u, v), q)$: This commitment is ill-formed since the debtor controls the antecedent. Agent $x$ may elect not to create the nested commitment, and therefore, avoid detaching the outer commitment.

- $C(x, y, p, C(y, x, u, v))$: This commitment is ill-formed since $x$ cannot create the nested commitment whose debtor is $y$.

- $C(x, y, p, C(x, y, u, v))$: This commitment is well-formed. However, this commitment can be reduced. Instead of committing to $C(x, y, u, v)$, $x$ may directly commit to $v$. In this case, the reduced commitment is: $C(x, y, w, v)$, where $w$ is $p \land u$.

- $C(x, y, C(y, x, u, v), q)$: This commitment is well-formed and useful.
6.2 Illustrative Application

We illustrate the value of integrated reasoning over commitments and goals with a real-world scenario in the domain of aerospace aftermarket services, which we adopt from the European Union CONTRACT project [van Aart et al., 2007].

Figure 6.2: A high-level model of the aerospace aftermarket process (from the CONTRACT project [van Aart et al., 2007]).

Figure 6.2 shows a high-level process flow of aerospace aftermarket services. The participants are an airline operator, an aircraft engine manufacturer, and a parts manufacturer. The engine manufacturer provides engines to the airline operator, and additionally services the engines to keep them operational; in return, the operator pays the manufacturer. If a plane waits on the ground for an engine to be serviced, the manufacturer pays a penalty to the operator. As part of the agreement, the operator regularly provides engine health data to the manufacturer, and may proactively request the manufacturer to perform scheduled engine maintenance. The manufacturer analyzes the engine health data and informs the operator of any required unscheduled engine maintenance. As part of servicing the engine, the manufacturer can either

---

**Operator (OPER)**
- Operate Aircraft
- Pull Aircraft

**Engine Manufacturer (MFG)**
- Monitor Engine Health
- Request Maintenance
- Arrange Maintenance
- Remove Engine
- Replace
- Refurbish

**Parts Manufacturer (EMFG)**
- Supply Parts

---
refurbish or replace it. The manufacturer maintains a supply of engines by procuring parts from a parts manufacturer.

Table 6.2 describes the goals and commitments that model this scenario. For clarity, we exclude the airline manufacturer’s activity of purchasing parts. In the table, proposition engine_with_service holds when the manufacturer provides the engine and creates commitments \( C_3 \) and \( C_4 \), and proposition paid_-health_rep_promised holds when the airline operator pays the manufacturer and creates commitment \( C_5 \).

Table 6.3 describes a possible progression of the aerospace scenario. Each step shows the life cycle transition or practical reasoning rule that the airline manufacturer (MFG) or the operator (OPER) employ, and how their configurations progress. For readability, we place new or modified state elements in bold, and omit satisfied commitments and goals in steps subsequent to their being satisfied.

In Steps 1 and 2, the airline manufacturer and the operator consider and activate end goals \( G_1 \) and \( G_2 \), respectively. In Step 3, the manufacturer entices (exercising the ENTICE rule) the operator to create \( C_1 \), which would enable the manufacturer to satisfy \( G_1 \). Notice how ENTICE causes manufacturer’s configuration to reach the coherent state \( \langle \{G^A_1\}, \{C^A_1\} \rangle \). Similarly, in Step 4, operator creates \( C_2 \).
Table 6.2: Goals and commitments from the aerospace scenario.

<table>
<thead>
<tr>
<th>ID</th>
<th>Goal, Commitment, or Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_1$</td>
<td>$G(MFG, \text{paid_health_rep_promised, insufficient_money})$</td>
<td>Airline manufacturer’s (MFG’s) goal to receive the payment and the promise to provide the health report</td>
</tr>
<tr>
<td>$G_2$</td>
<td>$G(\text{OPER, engine_with_service, engine_not_provided})$</td>
<td>Operator’s (OPER’s) goal to receive the engine and the promise to provide the service</td>
</tr>
<tr>
<td>$G_3$</td>
<td>$G(\text{OPER, paid_health_rep_promised, insufficient_money})$</td>
<td>Operator’s goal to make the payment and the promise to provide the health report</td>
</tr>
<tr>
<td>$G_4$</td>
<td>$G(MFG, \text{engine_with_service, engine_not_provided})$</td>
<td>Manufacturer’s goal to provide the engine and the promise to provide the service</td>
</tr>
<tr>
<td>$G_5[i]$</td>
<td>$G(\text{OPER, service_requested}[i], service_not_requested}[i])$</td>
<td>Operator’s goal to request the service; there is an instance of this goal for each occurrence of service needed</td>
</tr>
<tr>
<td>$G_6[i]$</td>
<td>$G(MFG, \text{service_provided}[i], service_not_provided}[i])$</td>
<td>Manufacturer’s goal to provide the service; there is an instance of this goal for each service request</td>
</tr>
<tr>
<td>$G_7[i]$</td>
<td>$G(MFG, \top, \text{penalty_paid}[i], \text{penalty_paid}[i], penalty_not_paid}[i])$</td>
<td>Manufacturer’s goal to pay the penalty if the engine is down; there is an instance of this goal for each engine down occurrence</td>
</tr>
<tr>
<td>$C_1$</td>
<td>$C(MFG, \text{OPER, paid_health_rep_promised, engine_with_service})$</td>
<td>Manufacturer’s commitment to operator to provide the engine and service if operator pays and promises to provide the health report</td>
</tr>
<tr>
<td>$C_2$</td>
<td>$C(\text{OPER, MFG, engine_with_service, paid_health_rep_promised})$</td>
<td>Operator’s commitment to the manufacturer to pay and to provide the health report if the manufacturer provides the engine and service</td>
</tr>
<tr>
<td>$C_3[i]$</td>
<td>$C(MFG, \text{OPER, service_requested}[i], service_provided}[i])$</td>
<td>Manufacturer’s commitment to the operator to provide the service if the operator requests service</td>
</tr>
<tr>
<td>$C_4[i]$</td>
<td>$C(MFG, \text{OPER, engine_down}[i], penalty_paid}[i])$</td>
<td>Manufacturer’s commitment to the operator to pay penalty if the engine is down; there is an instance of this commitment for each occurrence of the engine downtime</td>
</tr>
<tr>
<td>$C_5[i]$</td>
<td>$C(\text{OPER, MFG, health_report_requested}[i], health_report_provided}[i])$</td>
<td>Operator’s commitment to the manufacturer to provide the health report if the manufacturer requests the report; there is an instance of this commitment for each health report request</td>
</tr>
<tr>
<td>#</td>
<td>Event or Rule</td>
<td>MFG’s Action</td>
</tr>
<tr>
<td>----</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>1</td>
<td>(life cycle)</td>
<td>consider($G_1$) ∧ activate($G_1$)</td>
</tr>
<tr>
<td>2</td>
<td>(life cycle)</td>
<td>consider($G_2$) ∧ activate($G_2$)</td>
</tr>
<tr>
<td>3</td>
<td>ENTICE</td>
<td>create($C_1$)</td>
</tr>
<tr>
<td>4</td>
<td>ENTICE</td>
<td>create($C_2$)</td>
</tr>
<tr>
<td>5</td>
<td>DETACH</td>
<td>consider($G_4$) ∧ activate($G_4$)</td>
</tr>
<tr>
<td>6</td>
<td>DETACH</td>
<td>consider($G_3$) ∧ activate($G_3$)</td>
</tr>
<tr>
<td>7</td>
<td>(life cycle)</td>
<td>suspend($G_4$)</td>
</tr>
<tr>
<td>8</td>
<td>SUSPEND OFFER</td>
<td>suspend($G_2$)</td>
</tr>
<tr>
<td>9</td>
<td>BACK BURNER</td>
<td>suspend($G_4$)</td>
</tr>
<tr>
<td>10</td>
<td>(life cycle)</td>
<td>reanimate($G_4$)</td>
</tr>
<tr>
<td>11</td>
<td>REVIVE</td>
<td>reanimate($C_2$)</td>
</tr>
<tr>
<td>12</td>
<td>REVIVE</td>
<td>reanimate($G_4$)</td>
</tr>
<tr>
<td>13</td>
<td>(life cycle)</td>
<td>engine</td>
</tr>
<tr>
<td>14</td>
<td>(life cycle)</td>
<td>create($C_3$)</td>
</tr>
<tr>
<td>15</td>
<td>(life cycle)</td>
<td>create($C_4$)</td>
</tr>
<tr>
<td>16</td>
<td>(life cycle)</td>
<td>paid</td>
</tr>
<tr>
<td>17</td>
<td>(life cycle)</td>
<td>create($C_5$)</td>
</tr>
<tr>
<td>18</td>
<td>service_needed[1]</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>DETACH</td>
<td>consider($G_5[1]$) ∧ activate($G_5[1]$)</td>
</tr>
<tr>
<td>20</td>
<td>(life cycle)</td>
<td>service_requested[1]</td>
</tr>
<tr>
<td>21</td>
<td>DELIVER</td>
<td>consider($G_6[1]$) ∧ activate($G_6[1]$)</td>
</tr>
<tr>
<td>22</td>
<td>(life cycle)</td>
<td>service_provided</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In Step 5, the manufacturer considers and activates $G_4$ to detach (DETACH rule) $C_2$. Observe how DETACH activates manufacturer’s (debtor’s) means goal $G_4$, which corresponds to the operator’s (creditor’s) end goal $G_2$. In Step 6, the operator considers and activates means goal $G_3$ to detach $C_1$.

In Step 7, possibly due to its other priorities, the operator decides to suspend $G_2$. The operator suspends $C_2$ (SUSPEND OFFER rule) in Step 8, which transitions its configuration to the state $\langle \{G_2^U\}, \{C_2^P\} \rangle$. In Step 9, the manufacturer suspends $G_4$ (BACK BURNER rule), which transitions its configuration to the coherent state $\langle \{G_4^U\}, \{C_2^P\} \rangle$. Observe how the practical reasoning rules cause the manufacturer (debtor) to suspend its means goal $G_4$ in response to the operator (creditor) suspending its end goal $G_2$. In Steps 10–11, the operator reactivates $G_2$, and reactivates (REVIVE rule) $C_2$. In Step 12, the manufacturer activates (REVIVE rule) $G_4$.

In Steps 13–15, the manufacturer provides engine (engine) to the operator and creates $C_3$ and $C_4$. Recall that engine_with_service means providing the engine and creating $C_3$ and $C_4$, and that the satisfaction condition of $G_2$ and $G_4$ is engine_with_service. Therefore, in Step 15, $G_2$ and $G_4$ are satisfied. Further, since engine_with_service is the consequent of $C_1$ and the antecedent of $C_2$, in Step 15, $C_1$ is satisfied and $C_2$ is detached. In Steps 16–17, the operator pays the manufacturer (paid), and creates $C_5$ (paid_health_rep_promised). This satisfies $G_1$, $G_3$, and $G_2$. Observe how, in Step 17, the practical reasoning rules cause the manufacturer’s and the operator’s configurations to reach the coherent states $\langle \{G_2^S\}, \{C_2^S\} \rangle$ and $\langle \{G_3^S\}, \{C_2^S\} \rangle$, respectively.

The goals and commitments as stated are schematic in that each proposition would be instantiated for a particular business transaction. Following common practice, we consider each proposition schema as instantiated via an ID parameter, $i$. A service_needed event occurs at Step 18; it instantiates the parameter $i$ with a value of 1. In response, the operator activates the means goal $G_5[1]$, an instance of $G_5$, to request the service in Step 19. By its requesting the service, in Step 20, the operator satisfies $G_5[1]$ and detaches $C_2[1]$, the corresponding instance of $C_3$. To deliver upon its commitment, the manufacturer activates the discharge goal $G_6[1]$ in Step 21, and provides the service in Step 22. This service being provided satisfies both $G_6[1]$ and $C_3[1]$. Finally, in Step 23, only the recurring commitments $C_3$, $C_4$, and $C_5$ remain in the agent configurations.

### 6.3 Establishing Convergence Properties

In general, we cannot guarantee that any agent will succeed with its goals, because the agent may lack the capabilities and resources to achieve them all, and other agents may not wish to help it; and because of the exogenous effects of the environment. Thus we cannot prove that an agent’s goals and commitments will all reach successful terminal states. Instead, we motivate the idea of a coherent state and ask whether we can guarantee that a coherent state will be reached, no matter how the agents decide to act—provided that they act according to the life cycles and practical rules we have given. Informally, in a coherent state, corresponding goals and commitments align. We make this notion precise below. We show that the
practical rules given in Section 6.1.3 are sufficient for an agent to reach a coherent state.

Convergence between the goals and commitments of an agent is crucial, as otherwise an agent may fail to achieve its goals, or may expend effort in satisfying unnecessary commitments. As an example, consider an agent \( x \) that has a goal \( G(x, s, f) \), but that lacks the capability to bring about \( s \). If \( x \) fails to create a commitment \( C(x, y, s, u) \) toward some other agent \( y \), then there is no clear path for \( x \) to achieve the goal. However, note that creating the commitment by itself does not guarantee that \( x \) achieves the goal, since \( y \) may fail to bring about \( s \). In case \( y \) fails to bring about \( s \), \( x \) needs to create a new commitment, or to abandon the goal.

We now define the notion of a coherent state. We would like for an agent execution (as described in Figure 6.1) to converge and terminate in one of the coherent states.

**Definition 1.** Let \( C = C(x, y, s, u) \) be a commitment. A goal \( G \) linked to \( C \) provided \( G \) is \( C \)'s debtor's goal for \( C \)'s antecedent or its debtor's goal for \( C \)'s consequent or its creditor's goal for \( C \)'s antecedent. That is, for some proposition \( f \), \( G = G(x, s, f) \) or \( G = G(y, s, f) \) or \( G = G(x, u, f) \).

**Definition 2.** Let \( C = C(x, y, s, u) \) be a commitment and \( G \) be a goal linked to \( C \). Then the terminal configurations \( \langle G^{T\lor F}, C^{N\lor E\lor T\lor V} \rangle \) and \( \langle G^S, C^S \rangle \), or the interim configurations \( \langle G^{I\lor A}, C^{C\lor D} \rangle \) and \( \langle G^{U\lor S\lor F}, C^{P\lor E\lor D} \rangle \) are coherent with respect to \( G \) and \( C \).

As explained previously, the life cycles for goals and commitments are treated as fixed in this paper. The practical rules combine with the life cycles to effect transitions from one configuration of an agent to the next. The intuition of the results stated below is to say that the sequence of agent configurations converges to one that is coherent. Intuitively, by a sequence of configurations converging to a coherent configuration, we mean that no infinite execution may exist that does not reach a coherent configuration.

However, while such a result is clean, our semantics includes certain rules that inherently cause cycles in execution. Therefore, we must address two possible causes of cycles between configurations. The first cause of a configuration cycling is if a goal cycles between Inactive, Active, and Suspended, seen in Figure 2.2. These transitions are captured in the three practical rules SUSPEND END GOAL, RECONSIDER END GOAL, and REACTIVE END GOAL. (6.14), (6.15), and (6.16). The second cause of a configuration cycling lies with the practical rules that can re-create goals and commitments, namely, PERSIST (DISCHARGE) (from commitment to means goal and from commitment to discharge goal) and NEGOTIATE. Any of these rules could cause endless cycles; therefore we introduce the notion of a progressive rule:

**Definition 3.** A nonprogressive rule is one of these: NEGOTIATE (6.8), SUSPEND END GOAL (6.14), RECONSIDER END GOAL (6.15), REACTIVE END GOAL (6.16), and PERSIST (MEANS) (6.23), and PERSIST (DISCHARGE) (6.35).

A progressive rule is any practical rule from Section 6.1.3 that is not nonprogressive.
Note that the cycle between commitment states Conditional and Pending in Figure 2.1 cannot occur for an agent following our practical rules, because rules SUSPEND OFFER, REVIVE, and REVIVE TO WITHDRAW act on the commitment in response to changes in the state of the end goal. Thus the commitment can cycle to and from Pending only if the end goal cycles to and from Suspended, and the latter cycling is already covered by the definition of nonprogressive rule.

We are now ready to formally state the main results. We specify the convergence properties as CTL formulas over a transition system. The offer, end goal, means goal, and discharge goal are in the state of the transition system. The transitions in this transition system occur when one of the following occurs:

- an agent executes a practical rule;
- a goal or the offer progresses to a new state due to the action set by a practical rule;
- an agent brings about the success condition of a goal;
- an agent perceives that some other agent has brought about the success condition of a goal; and
- an agent perceives an event such as a goal failure or a commitment expiration from the environment.

Theorems 1 and 2 capture the intuition of a configuration converging to one that is coherent. Intuitively, under certain constraints, all agent executions eventually lead to one of the coherent configurations if the agent obeys our proposed practical rules. Thus, convergence is akin to the liveness property familiar from distributed systems.

Before stating the theorems, to provide an intuitive illustration of how practical rules lead to convergence, we describe a possible agent execution that converges to a coherent state.

1. Suppose that agent $x$ considers and activates $G_{end}$: practical rules (6.12) and (6.13). Then, agent $x$’s configuration transitions to $\langle G^A_{end}, C_N \rangle$.
2. Agent $x$ employs ENTICE rule to create commitment $C$ toward agent $y$.
3. Agent $x$’s configuration thus transitions to $\langle G^A_{end}, C^C \rangle$, and agent $y$’s configuration transitions to $\langle C^C \rangle$.
4. Since $C$ is conditional, $y$ employs DETACH and DETACH’ to create $G_{means} = G(y, s, f')$, transitioning agent $y$’s configuration to $\langle G^A_{means}, C^C \rangle$: practical rules (6.18) and (6.19).
5. Agent $y$ eventually satisfies $G_{means}$ (and therefore $G_{end}$), which thus detaches $C$. Agent $y$’s configuration transitions to $\langle G^S_{means}, C^D \rangle$, and agent $x$’s configuration transitions to $\langle G^S_{end}, C^D \rangle$.
6. Upon the detachment of $C$, $x$ employs DELIVER and DELIVER’ to create discharge goal $G_{dis} = G(x, u, f'')$: practical rules (6.30) and (6.31).
7. When \( x \) eventually satisfies \( G_{\text{dis}} \), \( x \)'s configuration transitions to \( \langle G^{S}_{\text{end}}, C^{S} \rangle \) and \( y \)'s configuration transitions to \( \langle G^{S}_{\text{means}}, C^{S} \rangle \).

The foregoing outlines how a possible execution leads to \( \langle G^{S}_{\text{end}}, C^{S} \rangle \). Next, we present the formal theorems. We give an English explanation before the formal CTL formulas for each of the conditions and conclusions.

Intuitively, Theorem 1 says that agent \( x \)'s configuration with respect to \( G_{\text{end}} \) and \( C \) will converge to a coherent state provided that the nonprogressive rules are not applied infinitely, and \( C \) is not cancelled or released.

**Theorem 1.** (Goals to commitments) \( \text{Let } G_{\text{end}} = G(x, s, f) \text{ and } C = C(x, y, s, u). \text{ Assume:} \)

**Progression:** No sequence of rules occurs that contains an interleaving of infinitely many occurrences of nonprogressive rules.

\[
\text{FAIRNESS } \neg G^{N}_{\text{end}} \land \neg G^{U}_{\text{end}} \land \neg G^{A}_{\text{end}} \land \neg G^{I}_{\text{end}}
\]

**Persistence:** The agents do not give up (cancel or release) on the commitment.

\[
\text{INVAR } \neg \text{cancel}(C) \land \neg \text{release}(C)
\]

Then:

**Convergence to \( \langle G^{S}, C^{S} \rangle \):** On any of the agent executions if there is a state in which the end goal is satisfied and the commitment is conditional, detached, or pending, then on all paths emanating from that state, in future, the end goal remains satisfied, and the commitment is satisfied.

\[
\text{AG } (G^{S}_{\text{end}} \land C^{C \lor D \lor P} \rightarrow \text{AF AG } (G^{S}_{\text{end}} \land C^{S}))
\]

**Convergence to \( \langle G^{T \lor F}, C^{N \lor E \lor T \lor V} \rangle \):** On any of the agent executions if there is a state in which the end goal fails or is terminated and the commitment is null or conditional or discharged, then on all paths emanating from that state, in future, the end goal remains failed or terminated, and the commitment never gets created (remains null), expires, or is terminated.

\[
\text{AG } (G^{T \lor F}_{\text{end}} \land C^{N \lor E \lor T \lor V} \rightarrow \text{AF AG } (G^{T \lor F}_{\text{end}} \land C^{N \lor E \lor T \lor V}))
\]

**Convergence to \( \langle G^{I \lor A}, C^{C \lor D} \rangle \):** On any of the agent executions if there is a state in which the end goal is active and the commitment is null, then on all paths emanating from that state, in future, the commitment becomes conditional.

\[
\text{AG } (G^{A}_{\text{end}} \land C^{N} \rightarrow \text{AF } C^{C})
\]
Convergence to \( \langle G \lor S \lor F, C \lor P \lor E \lor D \rangle \): On any of the agent executions if there is a state in which the end goal is suspended and the commitment is conditional, then on all paths emanating from that state, in future, the commitment is either suspended, or it expires or detaches.

\[
\text{AG} \left( G_{\text{end}} \land C \rightarrow \text{AF} \, C \lor P \lor E \lor D \right)
\]

Note that the fairness condition for the progression assumption in Theorem 1 means that the end goal must not infinitely often be in the Null, Inactive, Active, or Suspended states. These are the states that occur on the infinite loops caused by the practical rules CONSIDER END GOAL, SUSPEND END GOAL, RECONSIDER END GOAL, and REACTIVATE END GOAL. The invariant for persistence assumption eliminates all executions on which the commitment is cancelled or released. This invariant is necessary to ensure convergence to the coherent state \( \langle G^S, C^S \rangle \). Without this invariant, the commitment may be cancelled or released after the end goal is satisfied.

The elimination of executions on which the commitment is cancelled or released due to the persistence invariant raises an interesting question. How does the agent’s configuration converge in an execution in which the end goal is terminated or fails, and the commitment is conditional or detached? Since the commitment cannot be cancelled or released employing the WITHDRAW OFFER rule, the convergence seems impossible. This is not the case since the persistence invariant eliminates all paths on which the end goal is terminated or fails and the commitment is conditional or detached. The reason is that on such paths WITHDRAW OFFER rule would apply requiring the commitment to be cancelled.

Intuitively, Theorem 2 says that agent x’s configuration with respect to \( G_{\text{dis}} \) and \( C \), and agent y’s configuration with respect to \( G_{\text{means}} \) and \( C \) will converge to a coherent state provided that the nonprogressive rules are not applied infinitely, and \( C \) is not cancelled or released.

**Theorem 2.** (Commitments to goals) Let \( G_{\text{means}} = G(y, s, f') \) and \( G_{\text{dis}} = G(x, u, f'') \) and \( C = C(x, y, s, u) \). Assume:

**Progression:** No sequence of rules occurs that contains an interleaving of infinitely many occurrences of nonprogressive rules.

\[
\text{FAIRNESS} \quad \neg G_{\text{means}}^N \land \neg G_{\text{means}}^U \land \neg G_{\text{means}}^I \land \neg G_{\text{dis}}^N \land \neg G_{\text{dis}}^U \land \neg G_{\text{dis}}^I
\]

**Persistence:** The agents do not give up (cancel or release) on the commitment.

\[
\text{INVAR} \quad \neg \text{cancel}(C) \land \neg \text{release}(C)
\]

Then:

**Convergence to \( \langle G^T \lor F, C^N \lor E \lor T \lor V \rangle \):** On any of the agent executions if there is a state in which the \( C \) expires or is terminated, and \( G_{\text{means}} \) is inactive, active, or suspended, then on all paths emanating from that state, in future, \( C \) remains expired or terminated, and \( G_{\text{means}} \) fails or is terminated.
\( \text{AG} (C^{\lor T} \land G^{\lor AVU} \rightarrow \text{AF} \, \text{AG} (C^{\lor T} \land G^{T \lor F}_{\text{means}})) \)

Convergence to \( \langle G^S, C^S \rangle \): On any of the agent executions if there is a state in which the \( C \) is satisfied, and \( G^\text{means} \) is inactive, active, or suspended, then on all paths emanating from that state, in future, \( C \) remains satisfied, and \( G^\text{means} \) is satisfied.
\[ \text{AG} (C^S \land G^{\lor AVU}_{\text{means}} \rightarrow \text{AF} \, \text{AG} (C^S \land G^S_{\text{means}})) \]

Convergence to \( \langle G^I, C^C \lor D \rangle \): On any of the agent executions if there is a state in which the commitment is conditional and the means goal is null, then on all paths emanating from that state, in future, the means goal becomes inactive.
\[ \text{AG} (C^C \land G^N_{\text{means}} \rightarrow \text{AF} \, G^I_{\text{means}}) \]

Convergence to \( \langle G^U \lor S \lor F, C^P \lor E \lor D \rangle \): On any of the agent executions if there is a state in which the commitment is pending and the discharge goal is either inactive or active, then on all paths emanating from that state, in future, the discharge goal is either suspended, or it satisfies or fails.
\[ \text{AG} (C^P \land G^{\lor AVU}_{\text{means}} \rightarrow G^{U \lor S \lor F}_{\text{means}}) \]

Convergence to \( \langle G^T \lor F, C^N \lor E \lor V \rangle \): On any of the agent executions if there is a state in which the \( C \) is violated, and \( G^\text{dis} \) is inactive, active, or suspended, then on all paths emanating from that state, in future, \( C \) remains violated, and \( G^\text{dis} \) fails or is terminated.
\[ \text{AG} (C^V \land G^{\lor AVU}_{\text{dis}} \rightarrow \text{AF} \, \text{AG} (C^V \land G^{T \lor F}_{\text{dis}})) \]

Convergence to \( \langle G^S, C^S \rangle \): On any of the agent executions if there is a state in which the \( C \) is satisfied, and \( G^\text{dis} \) is inactive, active, or suspended, then on all paths emanating from that state, in future, \( C \) remains satisfied, and \( G^\text{dis} \) is satisfied.
\[ \text{AG} (C^S \land G^{\lor AVU}_{\text{dis}} \rightarrow \text{AF} \, \text{AG} (C^S \land G^S_{\text{dis}})) \]

Convergence to \( \langle G^I, C^C \lor D \rangle \): On any of the agent executions if there is a state in which the commitment is detached and the discharge goal is null, then on all paths emanating from that state, in future, the discharge goal becomes inactive.
\[ \text{AG} (C^D \land G^N_{\text{dis}} \rightarrow \text{AF} \, G^I_{\text{dis}}) \]

Convergence to \( \langle G^U \lor S \lor F, C^P \lor E \lor D \rangle \): On any of the agent executions if there is a state in which the commitment is pending and the discharge goal is either inactive or active, then on all paths emanating from that state, in future, the discharge goal is either suspended, or it satisfies or fails.
AG (C^P \land G^{I\lor A} \rightarrow AF G^{U\lor S\lor F})

Note that the fairness condition for progression in Theorem 2 means that the discharge goal and the means goal must not infinitely often be in the Null, Inactive, Active, or Suspended states.

6.3.1 Proof by Model Checking: The NuSMV Model

To prove the theorems just stated, we employ model checking, a formal verification technique. A benefit of this approach is that it not only provides a rigorous proof but also leads to a generic methodology whereby we can verify the correctness of any set of practical rules that a designer might consider for a particular setting.

This section describes the NuSMV model of our coupled semantics for goals and commitments, and the output of running the model checker on the model.

We employ a modular approach to build the NuSMV model. Besides a main module, our NuSMV model composes twelve other modules that implement (i) the skeleton, (ii) the life cycles, and (iii) the practical rules, for each of the commitment, the end goal, the means goal, and the discharge goal. A skeleton module declares and initializes a set of variables that model the status and the actions of the commitment or a goal. These self-contained modules accept only the necessary model variables as parameters. For clarity and to save space, we describe only the three commitment modules and the main module in detail below.¹

Commitment Skeleton Module

Listing 6.1 shows a fragment of the commitment skeleton module. Line 1 declares the module with parameters of end goal (g_end), means goal (g_means), and discharge goal (g_dis). Note that these parameters are instances of skeleton modules for the different goals.

Line 4 declares status as an enumerated variable whose value can be a commitment state. Lines 6–13 declare Boolean variables, one variable per transition that features in Figure 2.1’s commitment life cycle. Line 14 declares variable cpr of type commit_pract_rules which is a module that implements commitment practical rules, and Line 17 declares variable csr of type commit_struct_rules which is another module that implements the commitment life cycle. Line 20 assigns initial values of NULL to status, and FALSE to all the Boolean variables.

Commitment Life Cycle Module

Listing 6.2 shows a fragment of the module that encodes the commitment life cycle. Line 1 declares the module with the parameters of commitment status, create, ant_fail, ant, con, react, suspend, cancel, and

¹The entire source code is available as an online appendix to this article.
Listing 6.1: Commitment skeleton.

```
MODULE commit(g_end, g_means, g_dis)
VAR
status : {NULL, CONDITIONAL, DETACHED, EXPIRED,
          PENDING, TERMINATED, VIOLATED, SATISFIED};
create : boolean;
ant_fail: boolean;
ant : boolean;
con : boolean;
react : boolean;
suspend : boolean;
cancel : boolean;
release : boolean;
cpr : commit_pract_rules(status, create, ant_fail,
                         ant, con, react, suspend, cancel, release,
                         g_end, g_means, g_dis);
csr : commit_struct_rules(status, create, ant_fail,
                         ant, con, react, suspend, cancel, release);
INIT
status = NULL & create = FALSE & ant_fail = FALSE &
ant = FALSE & con = FALSE & react = FALSE &
suspend = FALSE & cancel = FALSE & release = FALSE;
```

release.

Listing 6.2: Commitment life cycle.

```
MODULE commit_struct_rules (status, create, ant_fail,
                            ant, con, react, suspend, cancel, release)
ASSIGN
  next(status) :=
  case
    status = NULL & create & !ant : CONDITIONAL;
    status = NULL & create & ant : DETACHED;
    status = CONDITIONAL & ant & con & ant_fail & suspend
    & cancel & release : {DETACHED, SATISFIED, EXPIRED,
                         PENDING, TERMINATED};
    status = CONDITIONAL & ant & con & ant_fail & suspend
    & cancel & !release : {DETACHED, SATISFIED, EXPIRED,
                           PENDING, TERMINATED};
    ...
  esac;
```
Line 6 encodes the transition of the commitment labeled create. If the current status of the commitment is NULL, create is true, and ant is false then in the next state the commitment status becomes CONDITIONAL. Line 7 encodes the transition labeled create ∧ ant of the commitment from the NULL state to the DETACHED state. Line 10 captures the possible transition from the CONDITIONAL state. From the CONDITIONAL state, any one of the propositions ant, con, ant_fail, suspend, cancel, or release may hold, which would transition the commitment to DETACHED, SATISFIED, EXPIRED, PENDING, or TERMINATED state respectively.

Commitment Practical Rules Module

Listing 6.3 shows a fragment of the module that encodes the commitment practical rules. Line 1 declares the module with the parameters of commitment status, create, ant_fail, ant, con, react, suspend, cancel, release, g_end, g_means, and g_dis. Here ant, con, react, and g respectively abbreviate antecedent, consequent, reactivate, and goal.

Lines 7–17 assign a value to the cancel variable in next state using a case block. Lines 9–11 encode the WITHDRAW OFFER practical rule. If the end goal fails or is terminated, and the commitment is conditional or detached, then in the next state, cancel is TRUE. In the NuSMV model, a practical rule sets a variable to true, and based on that variable, we make a transition to update the state of the commitment. Lines 12–14 encode the GIVE UP practical rule. If the discharge goal fails or terminates and commitment is detached, then in the next state if x does not consider the discharge goal (that is, does not use the PERSIST (DISCHARGE) practical rule, next(g_dis.consider) = FALSE), then x cancels the commitment in the next state. Similarly, Lines 21–22 encode REVIVE TO WITHDRAW, and Lines 23–25 encode REVIVE practical rules. If cancel is TRUE in the current state, then Line 15 resets it to FALSE in the next state. Similarly, Line 26 resets react to FALSE in the next state, if it is TRUE in the current state. Line 16 is the default clause in the case block, which applies if none of the other clauses match. It assigns the value of cancel in the next state to be the value of cancel in the current state.

Main Module

Listing 6.4 shows a fragment of the main module. Line 1 declares the main module. Line 4 creates an instance of the end goal module with commitment (c) and means goal (g_means.g) as the arguments. Similarly, lines 5, 6, and 7 create an instance of the commitment module, the means goal module, and the discharge goal module, respectively. Lines 9–27 define the CTL specifications, fairness, and invariant constraints of Theorem 1. Lines 9–12 define AG (G^S_{end} \land C^{C \lor D \lor F} \rightarrow AF AG (G^S_{end} \land C^S)), and Lines 14–20 define AG (G^{T \lor F}_{end} \land C^{N \lor C \lor V} \rightarrow AF AG (G^{T \lor F}_{end} \land C^{N \lor E \lor T \lor V})). Lines 22–23 capture

\textsuperscript{2}We reuse a base goal module in the end goal, means goal, and discharge goal modules. We access the base goal module using the dot operator, e.g. g\_means.g refers to the base goal module for the means goal g\_means.
Listing 6.3: Commitment practical rules.

```plaintext
MODULE commit_pract_rules(status, create, ant_fail,
    ant, con, react, suspend, cancel, release,
    g_end, g_means, g_dis)

ASSIGN

next(cancel) :=
    case
        (status = CONDITIONAL | status = DETACHED) &
        (g_end.status = TERMINATED | g_end.status = FAILED) &
        !cancel : TRUE;
        status = DETACHED & (g_dis.status = TERMINATED |
            g_dis.status = FAILED) & next(g_dis.consider) = FALSE &
        !cancel : TRUE;
        cancel : FALSE;
        TRUE : cancel;
    esac;

next(react) :=
    case
        status = PENDING & (g_end.status = TERMINATED |
            g_end.status = FAILED) & !react : TRUE;
        status = PENDING & (g_end.status = ACTIVE |
            g_end.status = SATISFIED) & !react :
        TRUE;
        react : FALSE;
        TRUE : react;
    esac;
```

the progression assumption of Theorem 1: FAIRNESS $\neg G_{end}^N \land \neg G_{end}^U \land \neg G_{end}^A \land \neg G_{end}^I$. Lines 25–27 capture the persistence assumption of Theorem 1: INVAR $\neg cancel(C) \land \neg release(C)$.

### 6.3.2 Output of Model Checking showing Convergence

We execute NuSMV (version 2.4) on the full versions of the above modules and specifications. It reports that the model satisfies all the CTL specifications: that is, our set of practical rules guarantee the stated convergence properties. Figure 6.3 shows a partial screenshot of the NuSMV output. The highlighted line, for example, shows that the model satisfies the CTL specification of Theorem 1: $AG (G_{end}^S \land C^{CVDP} \rightarrow AF AG (G_{end}^S \land C^S))$. 

113
6.3.3 Output of Model Checking showing Lack of Convergence

Each of the practical rules is necessary to ensure convergence to a coherent state. To illustrate one example of the necessity of the practical rules, we remove the REVIVE practical rule from the model (by commenting out Lines 23–25 shown in Listing 6.3). Upon re-executing, NuSMV reports that the updated model fails to satisfy the CTL specification $\text{AG}(G_{\text{End}} \land C_C \lor D \lor P \rightarrow \text{AF AG}(G_{\text{End}} \land C_S))$ of Theorem 1. Figure 6.4 is a partial screenshot of the NuSMV output showing the failure. The highlighted line shows that the model fails to satisfy the CTL specification: $\text{AG}(G_{\text{end}}^S \land C_{\text{revive}}^S \rightarrow \text{AF AG}(G_{\text{end}}^S \land C_S))$. NuSMV shows a counterexample in which the end goal is satisfied, but the commitment stays in the pending state forever. Similarly, we can show the necessity of the other practical rules.
6.4 Maintenance Commitments

We now add maintenance commitments (and goals) to the operational semantics from Section 6.1.

A maintenance commitment $S = S(x, y, l, m, d)$ means that debtor $x$ commits to creditor $y$ that if $l$ is brought about $x$ will sustain the maintenance condition $m$. ($S$ stands for sustains.) $S$ discharges when the discharge condition $d$ becomes true. We omit $d$ in our presentation except where necessary.

Fig. 6.5 shows the lifecycle of a maintenance commitment. The substate Sustain is a distinctive of maintenance commitments. A commitment becomes Active upon creation. Once the antecedent has become true, the commitment remains in state Detached. Should the maintenance condition $m$ become false (or be predicted to become false, i.e., $B_x \models \pi_x(\neg m)$), then the commitment transitions via the respond transition to state Sustain. Here, the agent acts to restore the truth of (or prevent the falsehood of) $m$, whereupon the commitment returns to Detached. This behaviour is akin to the Active-Waiting states of a maintenance goal. Unlike an achievement commitment, which is Satisfied when its consequent becomes true, a maintenance commitment persists (unless it expires, terminates, or is violated) until its discharge condition $d$ becomes true.

6.4.1 Lifecycle of a Maintenance Commitment

In detail, the lifecycle of a maintenance commitment as in Figure 6.5 is as follows:
If the debtor creates a commitment, then the commitment becomes active; it becomes conditional or detached depending on the antecedent.

The antecedent failure of a conditional commitment causes it to expire.

If the antecedent of a conditional commitment is brought about, then the commitment detaches.

If the debtor suspends a conditional or detached commitment, then the commitment becomes pending.

If the debtor reactivates a pending commitment, the commitment becomes active; moreover, if the antecedent holds, then the commitment becomes detached else conditional.

If the maintenance condition $m$ become false (or be predicted to become false, then the commitment transitions via the respond transition to state Sustain.

If the debtor restores the truth of (or prevent the falsehood of) $m$, then the commitment returns to Detached.

If the debtor cancels a conditional commitment, then the commitment terminates.

If the creditor releases the debtor from a conditional or detached commitment, then the commitment terminates.
6.4.2 Formalization including Maintenance Commitments and Goals

We now add maintenance commitments and goals to the formalization from Section 6.1.2. Agent $x$’s configuration is the tuple $E_x = \langle B, G, M, C, S, A^G, A^M, A^C, A^S \rangle$. The components of agent configuration beyond those from Section 6.1.2 are as follows:

- $S$ is a set of maintenance commitments, i.e., $S(x, y, l, m, d)$, where $x$ and $y$ are agents, and $l$, $m$, and $d$ are propositions (which may include commitments).

- $M$ is a set of maintenance goals, of the form $M(x, m, s, f)$, where $m$, $s$, and $f$ are propositions. We omit $R$ and $P$ (Sec. 2.3) except where necessary.
• \( \mathcal{A}_M \) is a set of tuples \( \langle MGoal, MAct, MEv \rangle \). It maps a maintenance goal to a maintenance goal action \( MAct \) or event \( MEv \), where \( MAct \) can be consider, activate, suspend, respond, reactivate, drop, or abort, and \( MEv \) can be fail or succeed. An agent may employ a practical rule to set \( MAct \) for a maintenance goal, or the environment may set \( MEv \) for the goal. If \( MAct \) or \( MEv \) is set for a goal, then the goal transitions to a new state as per its lifecycle (Fig. 2.3).

• \( \mathcal{A}_S \) is a set of tuples \( \langle SComm, SAct, SEv \rangle \). It maps a maintenance commitment to a maintenance commitment action \( SAct \) or event \( SEv \). \( SAct \) can be create, suspend, reactivate, cancel, release, sustained, or respond, and \( SEv \) can be antecedent_failure, antecedent, consequent, or discharge. An agent may employ a practical rule to set \( SAct \), or the environment may set \( SEv \). If \( SAct \) or \( SEv \) is set for a maintenance commitment, then the commitment transitions to a new state as per its lifecycle (Fig. 6.5).

Table 6.4 gives in BNF the formal syntax for configuration components including achievement and maintenance goals and commitments. The conditions of goals and commitments are propositional expressions; commitments (but not goals) can be nested. A commitment in a goal condition means create the commitment. For example, \( G(OPER, C(OPER, MFR, engine\_provided, engine\_paid)) \) corresponds to agent \( OPER \) having a goal to create the commitment.

Figure 6.6: Goal and commitment patterns applied, starting from a maintenance goal.
Table 6.4: Syntax for beliefs, commitments, and goals.

\[\begin{align*}
\mathbf{E}:\ AgentState & \rightarrow \langle \text{Beliefs}, \text{Goals}, \text{MGoals}, \text{Comms}, \text{SComms}, \text{GAEs}, \text{MGAEs}, \text{CAEs}, \text{SCAEs} \rangle \\
\mathbf{B}:\ Beliefs & \rightarrow \text{Belief}(, \text{Belief})^* \\
\mathbf{G}:\ Goals & \rightarrow \text{Goal}(, \text{Goal})^* \\
\mathbf{A}:\ MGoals & \rightarrow \text{MGoal}(, \text{MGoal})^* \\
\mathbf{C}:\ Comms & \rightarrow \text{Comm}(, \text{Comm})^* \\
\mathbf{S}:\ SComms & \rightarrow \text{SComm}(, \text{SComm})^* \\
\mathbf{A}^G: \text{GAEs} & \rightarrow \text{GAE}(, \text{GAE})^* \\
\mathbf{G}^M: \text{MGAEs} & \rightarrow \text{MGAE}(, \text{MGAE})^* \\
\mathbf{G}^C: \text{CAEs} & \rightarrow \text{CAE}(, \text{CAE})^* \\
\mathbf{S}^C: \text{SCAEs} & \rightarrow \text{SCAE}(, \text{SCAE})^* \\
\text{Belief} & \rightarrow \text{B}(\text{Agent}, \text{atom}) \\
\text{Goal} & \rightarrow \text{G}(\text{Agent}, \text{PC}, P) \\
\text{MGoal} & \rightarrow \text{M}(\text{Agent}, \text{PC}, P, P) \\
\text{Comm} & \rightarrow \text{C}(\text{Agent}, \text{Agent}, \text{PC}, \text{PC}) \\
\text{SComm} & \rightarrow \text{S}(\text{Agent}, \text{Agent}, \text{PC}, \text{PC}, P) \\
P & \rightarrow \text{atom} | P \land P | P \lor P | \neg P \\
\text{PC} & \rightarrow P | PC \land PC | PC \lor PC | \neg PC | \text{Comm} | \text{SComm} \\
\text{MState} & \rightarrow \text{null} | \text{inactive} | \text{active} | \text{waiting} | \text{suspended} | \text{terminated} | \text{failed} | \text{satisfied} \\
\text{SState} & \rightarrow \text{null} | \text{conditional} | \text{expired} | \text{detached} | \text{sustain} | \text{pending} | \text{terminated} \\
& \quad | \text{violated} | \text{satisfied} \\
\text{MAct} & \rightarrow \text{consider} | \text{activate} | \text{suspend} | \text{respond} | \text{reactivate} | \text{drop} | \text{abort} \\
\text{MEv} & \rightarrow \text{fail} | \text{succeed} \\
\text{SAct} & \rightarrow \text{create} | \text{suspend} | \text{reactivate} | \text{cancel} | \text{release} | \text{sustained} | \text{respond} \\
\text{SEv} & \rightarrow \text{antecedent\_failure} | \text{antecedent} | \text{consequent} | \text{discharge}
\end{align*}\]

6.5 Operational Semantics including Maintenance Commitments and Goals

We employ the notation introduced in Section 6.1.3 to present the practical rules for reasoning patterns involving achievement and maintenance goals and commitments.

Our approach is able to formalize important patterns of reasoning with maintenance goals and commitments. We specify a set of practical rules for each pattern. Fig. 6.6 provides a graphical summary of the patterns relating a maintenance goal to a maintenance and an achievement commitment. Note that Fig. 6.6 includes an achievement pattern (going from C to a pair G and G Telang et al. [2012]), even though it is applied here on top of S.
6.5.1 Maintenance Goal to Achievement Commitment

The bottom branch of Fig. 6.6 shows this pattern. An agent $x$ wishes to outsource maintaining a goal $M$ to some other agent $y$. Therefore, $x$ creates an achievement commitment $C$ with a nested maintenance commitment $S$. In this case, $S$ should be scoped so as to maintain the condition $m$ until $M$ satisfies, that is, the discharge condition $d$ of $S$ should be the same as the success condition $s$ of $M$. We refer to $C$ as the offer and $M$ as the end goal. Table 6.5 lists the practical rules for this pattern that $x$ should adopt.

- **ENTICE**: $x$ entices $y$ by making an offer $C$. If $y$ creates $S$, then $x$ commits to bringing about $q$. 
  
  *Motivation*: $x$ can entice $y$ to maintain $M$ for it. This pattern presumes that $y$ benefits from $q$, that is, $y$ has a goal for $q$.

- **SUSPEND OFFER**: $x$ suspends $C$ if $M$ is suspended. 
  
  *Motivation*: By suspending $C$, $x$ informs $y$ that it can work on other tasks instead of working on $C$.

- **REVIVE**: If $M$ is waiting, and $C$ is pending, then $x$ reactivates $C$. 
  
  *Motivation*: $x$ needs $C$ to be active to get $y$ to maintain $M$.

- **WITHDRAW OFFER**: If $M$ fails or is terminated, and $C$ is conditional, then $x$ cancels $C$. 
  
  *Motivation*: $C$ is of no utility if $M$ for which it was created no longer exists.

- **NEGOTIATE**: If $M$ is waiting or suspended, and $C$ is expired or terminated, then $x$ creates a new offer $C'$. 
  
  *Motivation*: $x$ persists with its goal $M$ by trying alternative ways to induce other agents to cooperate.

- **ABANDON END GOAL**: If $M$ is waiting or suspended, and $C$ is expired or terminated, then $x$ drops $M$. 
  
  *Motivation*: Instead of persisting with its goal $M$, $x$ decides to drop it.

- **SUSPEND END GOAL**: After detaching $S$, $x$ suspends $M$. 
  
  *Motivation*: Since $S$ is detached, $x$ expects $y$ to maintain $M$, and therefore suspends $M$.

- **REACTIVATE END GOAL**: In case $y$ violates the maintenance commitment $S$, $x$ reactivates $M$. 
  
  *Motivation*: By violating the maintenance commitment, $y$ stops maintaining the condition $m$. Hence, $x$ reactivates $M$ to ensure maintenance of $m$.

- **RELEASE MAINTENANCE**: In case $M$ fails or is terminated, and the maintenance commitment $S$ is detached, then $x$ releases $y$ from $S$. 
  
  *Motivation*: Since $M$ is failed or terminated, $x$ does not require $m$ to be maintained anymore.
Table 6.5: Practical rules: From m-goal to a-commitment.

<table>
<thead>
<tr>
<th>Rules for goal holder</th>
<th>Guard</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTICE</td>
<td>(M^{W}, C^{N})</td>
<td>create(C)</td>
</tr>
<tr>
<td>SUSPEND OFFER</td>
<td>(M^{U}, C^{C})</td>
<td>suspend(C)</td>
</tr>
<tr>
<td>REVIVE</td>
<td>(M^{W}, C^{P})</td>
<td>reactivate(C)</td>
</tr>
<tr>
<td>WITHDRAW OFFER</td>
<td>(M^{T\lor F}, C^{C})</td>
<td>cancel(C)</td>
</tr>
<tr>
<td>NEGOTIATE</td>
<td>(M^{W\lor U}, C^{E\lor T})</td>
<td>create(C')</td>
</tr>
<tr>
<td>ABANDON END GOAL</td>
<td>(M^{W\lor U}, C^{E\lor T})</td>
<td>drop(M)</td>
</tr>
<tr>
<td>SUSPEND END GOAL</td>
<td>(M^{W}, S^{D})</td>
<td>suspend(M)</td>
</tr>
<tr>
<td>REACTIVATE END GOAL</td>
<td>(M^{U}, S^{T\lor V})</td>
<td>reactivate(M)</td>
</tr>
<tr>
<td>RELEASE MAINTENANCE</td>
<td>(M^{T\lor F}, S^{D})</td>
<td>release(S)</td>
</tr>
</tbody>
</table>

6.5.2 Maintenance Goal to Maintenance Commitment

The top branch of Fig. 6.6 shows this pattern. Agent x uses a nested S to make the offer to y. The practical rules for this pattern resemble those in Table 6.5 with C replaced by S in the first six rules. For example, the ENTICE rule has guard \(M^{W}, C^{N}\), and action create(C). The remaining three rules apply on the nested commitment: \(S(y, x, l_{1}, m, d_{1})\).

6.5.3 Maintenance Commitment to Maintenance and Achievement Goals

Fig. 6.6 shows three instances of this pattern that starts from a maintenance commitment. For example, starting from \(S = S(y, x, l, m, d)\), x adopts an achievement goal for l, \(G = G(x, l)\), and y adopts a maintenance goal for m, \(M = M(y, m)\). We refer to \(G\) and \(M\) as means goals, since they are the means by which x and y perform their part in S. In this case, \(M\) should be scoped such as to maintain m until \(S\) discharges, i.e., the success condition \(s\) of the maintenance goal \(M\) should be the same as the discharge condition \(d\) of the maintenance commitment \(S\). Table 6.6 shows the practical rules for this pattern. We describe the rules for the creditor x of the maintenance commitment \(S\).

- DETACH and DETACH’: If \(G\) is null and \(S\) is conditional, then x considers and activates \(G\) to detach \(S\). Motivation: By detaching \(S\), x hopes to influence y to maintain m.

- BACK BURNER: If \(G\) is active and \(S\) is pending, then x suspends \(G\). Motivation: x need not work on \(G\) when \(S\) is suspended.

- FRONT BURNER: If \(G\) is suspended and \(S\) is conditional, then x reactivates \(G\). Motivation: An active \(G\) is necessary to detach \(S\).

- ABANDON MEANS GOAL: If \(G\) is active and \(S\) is expired or terminated, then x drops \(G\). Motivation: \(G\) is of no utility if \(S\) for which it is created no longer exists.
• **PERSIST and PERSIST’**: If $G$ fails or is terminated and $S$ is conditional, then $x$ considers and activates $G’$ (another instance of $G$). *Motivation*: $x$ still wants $S$ detached.

• **GIVE UP**: If $G$ fails or is terminated and $S$ is conditional, $x$ releases $y$ from $S$. *Motivation*: $x$ gives up detaching $S$.

Next, we describe the rules for the debtor $y$ of $S$:

• **MAINTAIN and MAINTAIN’**: If $M$ is null and $S$ is detached, then $y$ considers and activates $M$ to prevent violating its commitment $S$. *Motivation*: $y$ is cooperative and tries to prevent violating its commitments.

• **BACK BURNER**: If $M$ is waiting and $S$ is suspended, then $y$ suspends $M$. *Motivation*: $y$ need not maintain $M$ if $S$ is suspended, and can employ its resources on other tasks.

• **FRONT BURNER**: If $M$ is suspended and $S$ is detached, then $y$ reactivates $M$. *Motivation*: $y$ must maintain $M$ to prevent violating $S$.

• **ABANDON MEANS GOAL**: If $M$ is waiting and $S$ is terminated or violated, then $y$ drops $M$. *Motivation*: $M$ is of no utility if $S$ for which it is created ceases to exist.

• **PERSIST and PERSIST’**: If $M$ fails or is terminated and $S$ is detached, then $y$ considers and activates $M’$ (another instance of $M$). *Motivation*: $y$ persists in preventing violation of its commitment.

• **GIVE UP**: If $M$ fails or is terminated and $S$ is detached, then $y$ cancels $S$. *Motivation*: $y$ gives up trying to prevent violation of its commitment.

### 6.5.4 Nested Achievement Commitment to Achievement Goals

This pattern, not seen in Fig. 6.6, has an offer that nests two maintenance commitments, namely, $C = C(x, y, S(y, x, ⊤, m, d), S(x, y, ⊤, l, d))$. Such a commitment can capture a style of joint activity such as: “If you continue to hold up your end of the table, I will continue to hold up my end of the table, and we will both stop when $d$ comes to hold.” This pattern combines two patterns: achievement commitment to achievement goals Telang et al. [2012], and maintenance commitment to maintenance and achievement goal. The practical rules of these two patterns (Table 6.6 and [Telang et al., 2012, Sec. 3.3]) apply to this pattern. For example, $y$ considers and activates (DETACH and DETACH’) an achievement goal for $S_y = S(y, x, ⊤, m, d)$ to detach $C$, and $x$ considers and activates (DELIVER and DELIVER’ Telang et al. [2012]) an achievement goal for $S_x = S(x, y, ⊤, l, d)$. To satisfy the achievement goals, $y$ and $x$ create $S_y$ and $S_x$, respectively. Then to prevent violating $S_y$ and $S_x$, $y$ and $x$ consider and activate (MAINTAIN and MAINTAIN’) $M_y = M(y, m)$ and $M_x = M(x, l)$, respectively.
Table 6.6: Practical rules: From m-commitment to m-and a-goals.

<table>
<thead>
<tr>
<th>Rules for creditor</th>
<th>Guard</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>DETACH</td>
<td>$\langle G^N, S^C \rangle$</td>
<td>consider($G$)</td>
</tr>
<tr>
<td>DETACH’</td>
<td>$\langle G^I, S^C \rangle$</td>
<td>activate($G$)</td>
</tr>
<tr>
<td>BACK BURNER</td>
<td>$\langle G^A, S^P \rangle$</td>
<td>suspend($G$)</td>
</tr>
<tr>
<td>FRONT BURNER</td>
<td>$\langle G^U, S^C \rangle$</td>
<td>reactivate($G$)</td>
</tr>
<tr>
<td>ABANDON MEANS GOAL</td>
<td>$\langle G^A, S^{EVT} \rangle$</td>
<td>drop($G$)</td>
</tr>
<tr>
<td>PERSIST</td>
<td>$\langle G^{TVF}, S^C \rangle$</td>
<td>consider($G'$)</td>
</tr>
<tr>
<td>PERSIST’</td>
<td>$\langle G^{TVF}, G^I, S^C \rangle$</td>
<td>activate($G$)</td>
</tr>
<tr>
<td>GIVE UP</td>
<td>$\langle G^{TVF}, S^C \rangle$</td>
<td>release($S$)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rules for debtor</th>
<th>Guard</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAINTAIN</td>
<td>$\langle M^N, S^D \rangle$</td>
<td>consider($M$)</td>
</tr>
<tr>
<td>MAINTAIN’</td>
<td>$\langle M^I, S^D \rangle$</td>
<td>activate($M$)</td>
</tr>
<tr>
<td>BACK BURNER</td>
<td>$\langle M^W, S^P \rangle$</td>
<td>suspend($M$)</td>
</tr>
<tr>
<td>FRONT BURNER</td>
<td>$\langle M^U, S^D \rangle$</td>
<td>reactivate($M$)</td>
</tr>
<tr>
<td>ABANDON MEANS GOAL</td>
<td>$\langle M^W, S^{TVV} \rangle$</td>
<td>drop($M$)</td>
</tr>
<tr>
<td>PERSIST</td>
<td>$\langle M^{TVF}, S^D \rangle$</td>
<td>consider($M'$)</td>
</tr>
<tr>
<td>PERSIST’</td>
<td>$\langle M^{TVF}, M^I, S^D \rangle$</td>
<td>activate($M'$)</td>
</tr>
<tr>
<td>GIVE UP</td>
<td>$\langle M^{TVF}, S^D \rangle$</td>
<td>cancel($S$)</td>
</tr>
</tbody>
</table>

6.5.5 Achievement Goal to Maintenance Commitment

The right branch of Fig. 6.7 shows this pattern. An agent $x$ outsources an achievement goal $G = G(x, s)$ to $y$ by creating a maintenance commitment $S = S(x, y, s, m, d)$. That is, $x$ commits to $y$ to maintaining $m$ if $y$ brings about $s$. The practical rules for this pattern are similar to the first six rules of maintenance goal to achievement commitment pattern from Table 6.5. Instead of $M$ and $C$, the rules apply to $G$ and $S$. For example, the ENTICE rule has guard $\langle G^A, S^N \rangle$, and action create($S$). Note that the last three rules from Table 6.5 are irrelevant to this pattern since unlike $M$, $G$ satisfies when $S$ detaches.

6.5.6 Achievement Goal to Nested Maintenance Commitment

The left branch of Fig. 6.7 shows this pattern. It is similar to the achievement goal to maintenance commitment pattern except $x$ creates a nested commitment (achievement within maintenance): $S = S(x, y, C(y, x, \top, s), m, d)$. That is, $x$ commits to $y$ to maintaining $m$ if $y$ commits to bringing about $s$. Observe the difference between $S$ from this pattern, and $S' = S(x, y, s, m, d)$ from achievement goal to maintenance commitment pattern. To detach $S$, $y$ only needs to commit to bringing about $s$, whereas to detach $S'$, $y$ needs to bring about $s$. As compared to $S'$, $S$ has greater safety for $y$. Marengo et al. [2011]. This pattern has the same practical rules as Sec. 6.5.5.
6.6 Conclusion

This chapter studies the complementary aspects of commitments and goals by establishing an operational semantics of the related life cycles of the two concepts. We have distinguished the purely semantic aspects of their life cycles from the pragmatic aspects of how a cooperative agent may reason, and demonstrated desirable properties such as the convergence of agent states. From the viewpoint of agent programming, we have provided a foundational set of rules that is complete in a technical sense; their sufficiency in practice will be found through use.

We have established a set of coherence properties that follow from our semantics. Our proof proceeds by model checking CTL formulas. The coherence properties demonstrate desirable behavior, including the convergence of the configurations of cooperating agents, thereby delineating some theoretically well-founded yet practical modes of cooperation in a multiagent system. We illustrated our semantics with a scenario in the domain of aerospace aftermarket services.

We treat maintenance commitments as a first-class construct, enabling our main contribution of a cohesive account of maintenance goals and commitments. In contrast, current approaches (mostly on goals) treat commitments in an ad hoc fashion; and those involving commitments focus on achievement, leaving the treatment of maintenance to implicit procedures rather than logical models. The net benefit is that our approach facilitates cooperation in a wider variety of realistic settings, especially long-lived interactions where achievement-based approaches simply do not apply.
6.7 Related Work

Formalizations of Commitments

The commitment life cycle has been formalized by researchers before us, including by Fornara and Colombetti [2002], Mallya et al. [2003], and Marengo et al. [2011]. The variant we adopt uses state diagrams that include nested states, which simplifies the representation. Further, the states and transitions we adopt accord better with our intuitions. However, our approach could be applied to any of the alternative formulations as well.

Our work does not treat interagent communication and messaging, but studies the semantics of goals and commitments as a basis for cooperation. This semantics could underlie an account of reasoning about communications; indeed, commitments and agent communication have been well explored [Singh, 1999; Fornara and Colombetti, 2002].

Chopra et al. [2010a] formalize the semantic relationship between agents and protocols encoded as goals and commitments, respectively, to verify at design time if a protocol specification (expressed using commitments) supports achieving the goals in an agent specification, and vice versa. In contrast, our semantics applies at runtime, and we propose practical reasoning rules that agents may follow to achieve coherence between related goals and commitments. Dalpiaz et al. [2010] propose a model of agent reasoning based on the pursuit of variants—abstract agent strategies for pursuing a goal. We conjecture that Dalpiaz et al.’s approach can be expressed as sets of practical reasoning rules, such as those we described above.

Marengo et al. [2011] provide a logical formalization of commitments that include temporal regulations within the antecedent and consequent. They enable an agent to reason about when a temporal commitment is safe for the agent to accept as a debtor. They do not consider goals directly. Their work complements our work in that, conceivably, an agent may reason about the safety or otherwise of any commitments of which it is the debtor or creditor to decide how to proceed. In other words, an agent could use the more sophisticated reasoning provided by Marengo et al.’s approach to decide which practical rules to execute, for example, to drop or persist with the goals associated with a commitment.

The Belief-Desire-Intention Framework

The Belief-Desire-Intention (BDI) framework of Rao and Georgeff [1992] has been supplemented (or contrasted) with various cognitive and social notions, including for instance shared goals [Grosz and Kraus, 1996], and obligations or norms [Dignum et al., 2002a; Broersen et al., 2001]. Our work adopts some of the spirit of the BDI approach, but is not explicitly tied to it. We have used the terminology of beliefs and goals (which can be seen as correlating to Desires in Rao and Georgeff [1992]’s original terminology: see the discussions by, e.g. Dignum et al. [2002a]; Myers and Yorke-Smith [2007]; Braubach and Pokahr [2009b]). We have not considered intentions, since planning for achievement of goals and
execution of plans is outside our scope. Among many others, Winikoff et al. [2002] treat this aspect. Our setting incorporates social commitments in addition to elements of BDI-like cognitive state [Broersen et al., 2001; Myers and Yorke-Smith, 2005].

Criado et al. [2010] operationalize norm reasoning with the BDI framework, by adding a mechanism for recognizing norms in the environment and considering them during agent deliberation. Others have sought to operationalize norm-based reasoning in non-BDI architectures [Kollingbaum and Norman, 2003] and in planning [Panagiotidi and Vázquez-Salceda, 2011]. Like these efforts, our work provides an operational basis, but differs in that we specifically consider commitments.

Avali and Huhns [2008] relate an agent’s commitments to its beliefs, desires, and intentions using BDI\textsubscript{CTL*}, an extension of CTL* with modal operators for commitments, beliefs, desires, and intentions. El-Menshawy et al. [2011] define an extension of CTL with modalities for commitments and their satisfaction or violation. El-Menshawy et al. use model checking to formally verify properties of contracts modeled using commitments. In contrast with these approaches, we adopt a simpler and more tractable language, we consider commitments and goal life cycles, and propose practical reasoning rules that establish how an agent may reason about its goals and commitments systematically and with guarantees of arriving at a coherent state.

**Goals**

Among others, van Riemsdijk et al. [2008] and Thangarajah et al. [2011] propose abstract architectures for goals, on which is based the simplified goal life cycle that we consider. These and other authors formalize the operationalization of goals. In contrast, our work formalizes the combined operational semantics of goals and commitments. A natural extension of our work would be to address the different goal types that van Riemsdijk et al. and Dastani et al. propose. Our work is complementary also to the exploration of goals that have temporal extent (e.g. Braubach and Pokahr [2009a]; Hindriks et al. [2009]; Dastani et al. [2011]).

We have considered each goal to be private to an agent. Works that study the coordination of agents via shared proattitudes—such as shared goals—include, for example, works by Grosz and Kraus [1996] and Lesser et al. [2004]. A practical difference with our approach is that the shared attitudes approaches violate the heterogeneity and autonomy of agents, if the agents are provided access to each other’s goals. Also, if the agents have less than perfect trust in each other, the shared attitude collapses [Singh, 2012]. Commitments provide a cleaner interface between agents, specifying precisely what an agent would do for another and thus what the second agent should rely upon from the first.

We have considered each (means) goal in isolation. Thangarajah and Padgham [2011] share a similar conceptualization of goals as us, and treat the interaction between the goals and intentions of a single agent; Thangarajah and Padgham do not consider multiple agents nor commitments.

Kakas et al. [2008] present a modular agent architecture centred around agent state that evolves
according to transitions. They consider beliefs, goals, and plans, but do not consider commitments.

Winikoff [2007a] develops a mapping from commitments to BDI-style plans. He modifies SAAPL, an agent programming language, to include commitments in an agent’s belief-base and operational semantics update the commitments. Our operational semantics addresses goals (which are more abstract than plans) and commitments. It will be interesting to combine Winikoff’s work with ours to develop a comprehensive semantics for commitments, goals, and plans.

Günay et al. [2012] study dynamic protocol generation wherein agents generate commitments to other agents at runtime. The authors propose an algorithm that considers the goals and capabilities of the agent making the commitment, as well as the agent to whom it proposes the commitment, in order to make it more likely that the creditor agent will accept the protocol. Günay et al. [2012] require commitments to be explicitly accepted or rejected. Otherwise, their commitment and goal life cycles are similar to those we proposed previously and here [Telang et al., 2012], in line with prior works. Günay et al. [2012]’s work can be seen as a pre-cursor to ours, in that they study how to establish commitments, whereas we study the coherent management of commitments and goals, regardless of how they arise.

Applications of Model Checking

The use of model checking to establish properties of agent programs has further precedents. Bordini et al. [2008] build on a line of work. Specifically, they specify properties in a temporal language extended with certain agent-related modalities. They can verify the correctness of programs written in a variety of agent programming languages. Jongmans et al. [2010] apply model checking on-the-fly to agent programs, again to a variety of agent programming frameworks.

Dastani and Jamroga [2010] exemplify the verification of multiagent programs by developing a model of the system in a logic. They study the theoretical properties of a set of BDI-based agent programs in concurrent execution. In contrast to our use of CTL, Dastani and Jamroga employ an extended version of Alternating-time Temporal Logic (ATL). Like El-Menshawy et al. [2011], we use model checking to establish properties not of agent programs, but a semantics.

Maintenance Commitments and Goals

The lifecycle of maintenance commitments has not been previously studied. Mallya et al. [2003] discuss commitments to ensure the truth of a condition that happens to involve the maintenance of some condition. That is, they treat a maintenance commitment as an achievement commitment for a temporally extended condition. This dissertation proposes a lifecycle of maintenance commitments, and develops a joint operational semantics linking maintenance and achievement goals and commitments.

We define a maintenance goal and its lifecycle based on Duff et al. [2006]. Our approach to maintenance recognizes the inherent fallibility of agents in maintaining appropriate conditions in their
environment. A condition to be maintained is not ensured to hold continuously, but, every time it fails to hold, the agent \textit{reactively} attempts to resurrect it (as in Braubach et al. [2004]; Duff et al. [2006]).
Chapter 7

Hierarchical Planning about Commitments and Goals

Goals and commitments are declarative notions amenable to automated reasoning. There is a natural hierarchy in the plans that an agent generates to achieve its goals and to satisfy its commitments. For example, a goal may be decomposed into multiple conjunctive or disjunctive subgoals. The agent may satisfy a subgoal on its own, or may create a commitment toward another agent to satisfy it. If the commitment expires or is violated, then the agent may create a commitment toward some other agent, and sanction the violating agent. Not only can operations on commitments and goals feature within plans, but also to satisfy a commitment or a goal may require another hierarchical plan. Although multiple proposals for multiagent modeling based on goals and commitments exist, few approaches operationalize the resulting models. As an example, Winikoff [2007b] employs classical BDI plans to operationalize commitment-based interactions. However, Winikoff’s approach ignores goals and fails to take advantage of the natural hierarchy of goals and commitments.

This chapter proposes a novel approach based on Hierarchical Task Network (HTN) planning for operationalizing goals and commitments using the semantics proposed in Chapter 6. HTN planning has been in use in significant practical scenarios such as job scheduling [Ghallab et al., 2004]. We show how HTNs intuitively capture the plans that consider joint goal and commitment semantics. This chapter is based on my work in collaboration with Felipe Meneguzzi [Telang et al., 2013; Meneguzzi et al., 2013].

Purchase Scenario

We illustrate our approach via a simple purchase scenario involving a merchant and a customer. The merchant has a goal of getting paid, and can provide goods, and the customer has a goal to get the goods, and can pay. The merchant and the customer can achieve their goals in various ways. As an example, the merchant may commit to the customer to providing the goods if the customer pays. Eventually the
merchant may provide the goods to the customer (satisfying customer’s goal for goods), and the customer may pay the merchant (satisfying merchant’s goal to get paid) either before or after the merchant provides the goods. In another example, to achieve its goal for goods, the customer may manufacture the goods itself, and to achieve its goal of getting paid, the merchant may deposit funds in a bank and receive interest payments from the bank. However, it may be more costly for the customer to manufacture the goods than to procure the goods from the merchant, and the merchant’s interest payments may not be as much as the merchant’s profit by selling the goods to the customer.

Our HTN formalization enables the merchant and customer to generate a joint feasible plan to satisfy their goals. If the merchant or the customer prefer not to employ a generated plan, our formalization is capable of generating alternative plans.

### 7.1 Formalization within HTN

We now formalize the operational semantics of achievement commitments and goals as an HTN planning domain. Doing so, together with domain-specific operations, enables us to use an HTN planner such as JSHOP2 [Ilghami and Nau, 2003] to automatically synthesize correct protocols to achieve an individual agent’s goals either in isolation (Section 7.1.1) or via commitments among multiple agents (Section 7.1.4). We finish with the formalization of the HTN methods required to generate goal-commitment protocols in Section 7.1.5. We adopt JSHOP2’s convention of naming primitive tasks with an initial exclamation mark (!).

#### 7.1.1 Goal and Commitment Dynamics

As agents interact, their commitments and goals progress in a systematic manner. In essence, the dynamics of commitments and goals are captured using their lifecycles, as presented in Figures 2.1 and 2.2. Note that only some of the transitions in Figures 2.1 and 2.2 are under explicit control of an agent. For example, an agent can explicitly create a commitment, thus transitioning it to the active state. However, once a commitment is active, transitions to the satisfied and expired states occur based on what transpires or fails to transpire in the environment. Notice that the discharge of a commitment depends solely on the consequent becoming true, which could happen because of an action by any of the agents or through some environmental process. In settings where we wish to ensure that the debtor performs the action that brings about the consequent, we could specify the consequent to incorporate the debtor, e.g., paid(John, $1) instead of paid($1).

We formalize the dynamics using the HTN planning framework. This formalization seeks to support operations that an agent can explicitly execute to manipulate its internal representations of the current state of a goal or commitment. To capture the above intuition regarding the agent’s control, we define the dynamics in two parts: (1) a set of logical axioms characterizing the states of a goal and of a commitment;
and (2) a set of planning operators that induce the state changes over which the agent has direct control.

7.1.2 Commitment Axioms and Operators

We employ axioms \( p(C) \) and \( q(C) \) to represent the antecedent and consequent, respectively, of \( C \). These axioms employ domain-specific predicates. For example, for a commitment \( C(D, A, \text{pay, goods}) \), the predicate \( \text{pay} \) features in \( p(C) \), and the predicate \( \text{goods} \) features in \( q(C) \).

\[
\begin{align*}
p(C) &\leftarrow \text{commitment}(C,D,A) \land \text{commitmentType}(C,T_1) \land \text{pay} \\
q(C) &\leftarrow \text{commitment}(C,D,A) \land \text{commitmentType}(C,T_1) \land \text{goods}
\end{align*}
\]

In the above formulas, \( \text{commitment}(C,D,A) \) is a predicate meaning that \( C \) is a commitment from debtor \( D \) to creditor \( A \), and \( \text{commitmentType}(C,T_1) \) is a predicate meaning that \( C \) is of type \( T_1 \). The antecedent \( p(C) \) is true if \( \text{commitment}(C,D,A) \) and \( \text{commitment}(C,D,A) \) are true, and if the \( \text{pay} \) predicate is true. Note that, we introduce commitment type to allow the existence of multiple instances of a commitment with the same conditions.

We now present the axioms that characterize the states of a commitment \( C \). These axioms reference five predicates corresponding to commitment lifecycle states: \( \text{null}(C) \), \( \text{pending}(C) \), \( \text{canceled}(C) \), \( \text{released}(C) \), and \( \text{expired}(C) \).

\[
\begin{align*}
\text{conditional}(C) &\leftarrow \text{active}(C) \land \neg p(C) \\
\text{detached}(C) &\leftarrow \text{active}(C) \land p(C) \\
\text{active}(C) &\leftarrow \neg \text{null}(C) \land \neg \text{terminal}(C) \land \neg \text{pending}(C) \land \neg \text{satisfied}(C) \\
\text{terminated}(C) &\leftarrow (\neg p(C) \land \text{canceled}(C)) \lor \text{released}(C) \\
\text{violated}(C) &\leftarrow p(C) \land \text{canceled}(C) \\
\text{satisfied}(C) &\leftarrow \neg \text{null}(C) \land \neg \text{terminal}(C) \land q(C) \\
\text{terminal}(C) &\leftarrow \text{commitment}(C,D,A) \land (\text{canceled}(C) \lor \text{released}(C) \lor \text{expired}(C))
\end{align*}
\]

For example, the axiom \( \text{conditional}(C) \leftarrow \text{active}(C) \land \neg p(C) \) means that commitment \( C \) is conditional if it is active, \( \text{active}(C) \), and if its antecedent \( p(C) \) is false. Note that the \( \text{terminal} \) axiom is merely a shortcut for the set of states from which a commitment transition out to any other state.

We now present the HTN operators corresponding to the commitment operations. These operators add or delete predicates denoting the commitment lifecycle states if the specified preconditions hold.

\[
\text{operator} : \text{create}(C,D,A),
\]

131
pre : \((commitment(C, D, A) \land null(C))\),
\del : (null(C)), \add : (),
cost : 0

(operator : !suspend(C, D, A),
pre : \((commitment(C, D, A) \land active(C))\),
\del : (), \add : (pending(C)),
cost : 1)

(operator : !reactivate(C, D, A),
pre : \((commitment(C, D, A) \land pending(C))\),
\del : (), \add : (),
cost : 1)

(operator : !expire(C, D, A),
pre : \((commitment(C, D, A) \land conditional(C) \land timeout(C))\),
\del : (), \add : (expired(C)),
cost : 5)

(operator : !cancel(C, D, A),
pre : \((commitment(C, D, A) \land active(C))\),
\del : (), \add : (canceled(C)),
cost : 10)

(operator : !release(C, D, A),
pre : \((commitment(C, D, A) \land active(C))\),
\del : (), \add : (released(C)),
cost : 1)

For example, the create\((C, D, A)\) operator has the precondition that \(C\) is a commitment from debtor \(D\) to creditor \(A\) and \(C\) is \textit{null}. This operator deletes the predicate \textit{null}(C), and adds no predicate.

### 7.1.3 Goal Axioms and Operators

We employ axioms \(pg(G)\), \(s(G)\) and \(f\) to represent the precondition, success condition, and failure condition of a goal \(G\). These axioms employ domain-specific predicates. For example, for a goal \(G\textit{(pay, goods, deadline)}\), we encode the below axioms.

\[
pg(G) \leftarrow \text{goal}(G, A) \land \text{goalType}(G, GT1) \land \text{pay}
\]
\[
s(G) \leftarrow \text{goal}(G, A) \land \text{goalType}(G, GT1) \land \text{goods}
\]
\[
f(G) \leftarrow \text{goal}(G, A) \land \text{goalType}(G, GT1) \land \text{deadline}
\]

In the above formulas, \(\text{goal}(G, A)\) is a predicate meaning that \(G\) is a goal of an agent \(A\), and
goalType\((G, GT1)\) is a predicate meaning that \(G\) is of type \(GT1\). The success condition \(s(C)\) is true if \(goal(G, A)\) and \(goalType(G, GT1)\) are true, and if the \textit{goods} predicate is true.

We now present the axioms that characterize the states of a goal \(G\). These axioms reference five predicates corresponding to goal lifecycle states: \textit{null}(\(G\)), \textit{activatedG}(\(G\)), \textit{suspendedG}(\(G\)), \textit{dropped}(\(G\)), and \textit{aborted}(\(G\)).

\[
\begin{align*}
\text{inactiveG}(G) & \Leftarrow \neg null(G) \land \neg f(G) \land \neg terminal(G) \land \neg suspended(G) \land \neg active(G) \\
\text{activeG}(G) & \Leftarrow \text{activatedG}(G) \land \neg f(G) \land \neg satisfied(G) \land \neg terminal(G) \land \neg suspended(G) \\
\text{satisfiedG}(G) & \Leftarrow \neg null(G) \land \neg terminal(G) \land pg(G) \land s(G) \land \neg f(G) \\
\text{failedG}(G) & \Leftarrow \neg null(G) \land f(G) \\
\text{terminatedG}(G) & \Leftarrow \neg null(G) \land (\text{dropped}(G) \lor \text{aborted}(G)) \\
\text{terminalG}(G) & \Leftarrow \text{goal}(G, A) \land (\text{dropped}(G) \lor \text{aborted}(G))
\end{align*}
\]

For example, the axiom \textit{satisfiedG}(\(G\)) \Leftarrow \neg null(\(G\)) \land \neg \text{terminal}(\(G\)) \land pg(\(G\)) \land s(\(G\)) \land \neg f(\(G\)) means that goal \(G\) is satisfied if it is not null, \(\neg null(\(G\)),\) not in a terminal state, \(\neg terminal(\(G\)),\) its precondition \(pg(G)\) is true, its success condition \(s(G)\) is true, and its failure condition \(f(G)\) is false.

We now present the HTN operators corresponding to the goal operations. These operators add or delete predicates denoting the goal lifecycle states if the specified preconditions hold.

\[
\begin{align*}
\langle \text{operator} : !\text{consider}(G, A), \\
\text{pre} : & \text{goal}(G, A) \land null(G) \land pg(G), \\
\text{del} : & \text{null}(G), \text{add} :(), \\
\text{cost} : & 1 \rangle \\
\langle \text{operator} : !\text{activate}(G, A), \\
\text{pre} : & \text{goal}(G, A) \land \text{inactiveG}(G), \\
\text{del} : & (), \text{add} :\text{activatedG}(G), \\
\text{cost} : & 1 \rangle \\
\langle \text{operator} : !\text{suspend}(G, A), \\
\text{pre} : & \text{goal}(G, A) \land \text{terminalG}(G) \land \neg null(G), \\
\text{del} : & \text{activatedG}(G), \text{add} :\text{suspendedG}(G), \\
\text{cost} : & 1 \rangle \\
\langle \text{operator} : !\text{reconsider}(G, A), \\
\text{pre} : & \text{goal}(G, A) \land \text{suspendedG}(G) \land \neg \text{terminalG}(G) \land \neg null(G), \\
\text{del} : & (), \text{add} :\text{suspendedG}(G), \\
\text{cost} : & 1 \rangle \\
\langle \text{operator} : !\text{reactivate}(G, A), \\
\text{pre} : & \text{goal}(G, A) \land \text{suspendedG}(G) \land \neg \text{terminalG}(G) \land \neg null(G), \\
\text{del} : & (), \text{add} :\text{suspendedG}(G), \\
\text{cost} : & 1 \rangle
\end{align*}
\]
\[
\text{pre} : (\text{goal}(G, A) \land \text{suspended}(G) \land \neg \text{terminal}(G) \land \neg \text{null}(G)),
\]
\[
\text{del} : (\text{activated}(G)), \text{add} : (\text{suspended}(G)),
\]
\[
\text{cost} : 1 \}
\]

**operator** : !drop(G, A),

\[
\text{pre} : (\text{goal}(G, A) \land \neg \text{terminal}(G) \land \neg \text{null}(G)),
\]
\[
\text{del} : (), \text{add} : (\text{dropped}(G)),
\]
\[
\text{cost} : 1 \}
\]

**operator** : !abort(G, A),

\[
\text{pre} : (\text{goal}(G, A) \land \neg \text{terminal}(G) \land \neg \text{null}(G)),
\]
\[
\text{del} : (), \text{add} : (\text{aborted}(G)),
\]
\[
\text{cost} : 1 \}
\]

For example, the consider(G, A) operator has the precondition that G is a goal of agent, G is null, and G’s precondition is true. This operator deletes the predicate null(G), and adds no predicate.

### 7.1.4 Methods Relating Commitments and Goals

We now define the methods corresponding to the practical reasoning rules for achievement goals and commitments from Section 6.1.3. Recall that our practical rules are written as $\text{guard transition}$, where guard is a query on the state, and transition is specified as an operation on a commitment or goal. We express each rule as an HTN method. The guard of the rule becomes the precondition of the method, and the task network contains the operator corresponding to the transition.

For example, the ENTICE rule

\[
\text{ENTICE}
\]

is expressed as the HTN method below:

\[
\text{method : entice}(G, C, D, A),
\]
\[
\text{pre} : (\text{goal}(G, D) \land \text{active}(G) \land \text{commitment}(C, D, A) \land \text{null}(C) \land (s(G) \leftrightarrow p(C))),
\]
\[
\text{tn} : (!\text{create}(C, D, A))
\]

We combine the rules that consider and activate goals into one method. For example, we combine DELIVER and DELIVER’ rules into the below HTN method. Beyond considering and activating, this method invokes the achieveGoal method described in the next section.

\[
\text{method : deliver}(G, C, D, A),
\]
\[
\text{pre} : (\text{goal}(G, D) \land \text{null}(G) \land \text{commitment}(C, D, A) \land \text{detached}(C)),
\]
\[
\text{tn} : (!\text{consider}(G, D), !\text{activate}(G, D), !\text{achieveGoal}(G, D)))
\]

Similarly, we develop HTN methods for the remaining practical rules.
7.1.5 Bringing it all Together

For an HTN planner to generate valid plans to achieve an agent’s goals individually or through commitment protocols, we need methods to decompose an agent’s goals. We present these methods below.

(method : achieveGoals,
pre :(goal(G, A) ∧ pg(G) ∧ ¬activeG(G)),
tn :(!consider(G, A), !activate(G, A), achieveGoals))

(method : achieveGoals,
pre :(goal(G, A) ∧ activeG(G)),
tn :(achieveGoal(G, A)))

(method : achieveGoals,
pre :(goal(G₁, A₁) ∧ activeG(G₁) ∧ goal(G₂, A₂) ∧ activeG(G₂) ∧
commitment(C₁, A₁, A₂) ∧ (s(G₁) ↔ p(C₁)) ∧
commitment(C₂, A₂, A₁) ∧ (s(G₂) ↔ p(C₂))),
tn :({entice(G₁, C₁, A₁, A₂), entice(G₂, C₂, A₂, A₁)}, {detach(C₁), detach(C₂)}))

(method : achieveGoal(G₁, A₁),
pre :(goal(G₁, A₁) ∧ activeG(G₁) ∧ commitment(C, A₁, A₂) ∧ (s(G₁) ↔ p(C)) ∧
goal(G₂, A₁) ∧ (s(G₂) ↔ q(C)) ∧ (G₁ ≠ G₂)),
tn :({entice(G₁, C, A₁, A₂), detach(C), deliver(G₂, C, A₁, A₂)}))

The first achieveGoals method above recursively tries to activate all possible goals for an agent. If a certain goal is already active, the second method tries to decompose such a goal into an individually initiated plan to achieve that goal. Such a plan may consist of an individual plan: a designer must create a domain-dependent plan through an additional method to decompose the achieveGoal compound task. Or, the plan may consist of a generic commitment protocol: use the last method to create a commitment protocol based on enticing another agent \(A₂\) to adopt a commitment whose antecedent satisfies the goal, and whose consequent is a goal of \(A₂\). If \(A₂\) achieves \(A₁\)’s goal, this commitment detaches, in which case \(A₁\) delivers the promised consequent. Finally, the third method caters to two agents \(A₁\) and \(A₂\) whose goals can be achieved by mutual commitments. If both agents commit as appropriate, when \(A₁\) detaches its commitment, the goal of \(A₂\) is achieved, and vice versa.

Beyond the above methods, there will be domain dependent methods to achieve goals which correspond to an agent achieving a goal on its own. We present such methods for the purchase scenario in the next section.
7.2 Applying the HTN Formalization

We illustrate the HTN formalization on the purchase scenario we introduced at the beginning of this chapter. Table 7.1 summarizes the goals and commitments from this example. $G_{mp}$ is the merchant’s goal for getting paid before certain deadline. $G_{cg}$ is the customer’s goal for procuring the goods with the precondition that the customer has a need for the goods, and a deadline for receiving the goods as the failure condition. $G_{mg}$ is the merchant’s goal to provide the goods to the customer with the precondition of payment, and a deadline as the failure condition. $G_{cp}$ is the customer’s goal to pay the merchant with the precondition of receiving the goods, and a deadline as the failure condition. The commitment $C_{mc}$ is from the merchant to the customer to provide goods if the customer pays, and the commitment $C_{cm}$ is from the customer to the merchant to pay if the merchant provides the goods.

Table 7.1: Goals and commitments from the purchase example.

<table>
<thead>
<tr>
<th>Id</th>
<th>Commitment or Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_{mp}$</td>
<td>G(MERCHANT, ⊤, pay, deadline)</td>
</tr>
<tr>
<td>$G_{cg}$</td>
<td>G(CUSTOMER, needsGoods, goods, deadline)</td>
</tr>
<tr>
<td>$G_{mg}$</td>
<td>G(MERCHANT, pay, goods, deadline)</td>
</tr>
<tr>
<td>$G_{cp}$</td>
<td>G(CUSTOMER, goods, pay, deadline)</td>
</tr>
<tr>
<td>$C_{mc}$</td>
<td>C(MERCHANT, CUSTOMER, pay, goods)</td>
</tr>
<tr>
<td>$C_{cm}$</td>
<td>C(CUSTOMER, MERCHANT, goods, pay)</td>
</tr>
</tbody>
</table>

Table 7.2 shows the domain specific axioms for the goals and commitments in the purchase scenario. For example, the first three lines show the axioms for the precondition $pg(G_{mp})$, success condition $s(G_{mp})$, and failure condition $f(G_{mp})$ for the goal $G_{mp}$. The last two lines show the antecedent $p(C_{cm})$ and the consequent $q(C_{cm})$ for the commitment $C_{cm}$.

The HTN formalization for the purchase scenario reuses the domain-independent methods, operators, and axioms from Section 7.1. Table 7.3 shows the domain specific methods and operators for the purchase scenario.

**manufactureGoods:** Instead of procuring the goods from the merchant, the customer manufactures the goods on its own.

**earnInterest:** Instead of selling goods to and getting paid from the customer, the merchant deposits money in a bank to earn interest.

**sendPayment:** The customer pays the merchant for the goods.

**sendGoods:** The merchant sends the goods to the customer.
Table 7.2: Domain-specific axioms and methods for the goals and commitments in the purchase scenario.

\[\begin{align*}
pg(G_{mp}) & \leftarrow \text{goal}(G_{mp}, \text{merchant}) \land \text{goalType}(G_{mp}, GT_{mp}) \land \top \\
s(G_{mp}) & \leftarrow \text{goal}(G_{mp}, \text{merchant}) \land \text{goalType}(G_{mp}, GT_{mp}) \land \text{pay} \\
f(G_{mp}) & \leftarrow \text{goal}(G_{mp}, \text{merchant}) \land \text{goalType}(G_{mp}, GT_{mp}) \land \text{deadline} \\
pg(G_{cg}) & \leftarrow \text{goal}(G_{cg}, \text{customer}) \land \text{goalType}(G_{cg}, GT_{cg}) \land \text{needsgoods} \\
s(G_{cg}) & \leftarrow \text{goal}(G_{cg}, \text{customer}) \land \text{goalType}(G_{cg}, GT_{cg}) \land \text{goods} \\
f(G_{cg}) & \leftarrow \text{goal}(G_{cg}, \text{customer}) \land \text{goalType}(G_{cg}, GT_{cg}) \land \text{deadline} \\
pg(G_{mg}) & \leftarrow \text{goal}(G_{mg}, \text{merchant}) \land \text{goalType}(G_{mg}, GT_{mg}) \land \text{pay} \\
s(G_{mg}) & \leftarrow \text{goal}(G_{mg}, \text{merchant}) \land \text{goalType}(G_{mg}, GT_{mg}) \land \text{goods} \\
f(G_{mg}) & \leftarrow \text{goal}(G_{mg}, \text{merchant}) \land \text{goalType}(G_{mg}, GT_{mg}) \land \text{deadline} \\
pg(G_{cp}) & \leftarrow \text{goal}(G_{cp}, \text{customer}) \land \text{goalType}(G_{cp}, GT_{cp}) \land \text{goods} \\
s(G_{cp}) & \leftarrow \text{goal}(G_{cp}, \text{customer}) \land \text{goalType}(G_{cp}, GT_{cp}) \land \text{pay} \\
f(G_{cp}) & \leftarrow \text{goal}(G_{cp}, \text{customer}) \land \text{goalType}(G_{cp}, GT_{cp}) \land \text{deadline} \\
p(C_{mc}) & \leftarrow \text{commitment}(C_{mc}, \text{merchant}, \text{customer}) \land \text{commitmentType}(C_{mc}, CT_{mc}) \land \text{pay} \\
q(C_{mc}) & \leftarrow \text{commitment}(C_{mc}, \text{merchant}, \text{customer}) \land \text{commitmentType}(C_{mc}, CT_{mc}) \land \text{goods} \\
p(C_{cm}) & \leftarrow \text{commitment}(C_{cm}, \text{customer}, \text{merchant}) \land \text{commitmentType}(C_{cm}, CT_{cm}) \land \text{goods} \\
q(C_{cm}) & \leftarrow \text{commitment}(C_{cm}, \text{customer}, \text{merchant}) \land \text{commitmentType}(C_{cm}, CT_{cm}) \land \text{pay}
\end{align*}\]

**achieveGoal:** There are four achieveGoal methods. In the first method, the customer achieves its goal for goods \(G_{cg}\) by manufacturing the goods on its own. In the second method, the merchant achieves its goal for payment \(G_{mp}\) by depositing money in a bank and earning interest on it. In the third method, the merchant achieves its goal of providing goods \(G_{mg}\) by sending the goods to the customer. In the fourth method, the customer achieves its goal of paying \(G_{cp}\) by sending payment to the merchant.

**detach:** There are two detach methods corresponding to the two commitments \(C_{cm}\) and \(C_{mc}\). To detach \(C_{cm}\), the merchant sends the goods to the customer, and to detach \(C_{mc}\), the customer sends the payment to the merchant.

**deliver:** There are two deliver methods corresponding to the two commitments \(C_{cm}\) and \(C_{mc}\). To deliver \(C_{cm}\), the customer sends the payment to the merchant, and to deliver \(C_{mc}\), the merchant sends the goods to the merchant.

For simplicity, we use an absolute number for cost instead of an expression. We set the cost of manufactureGoods, earnInterest, sendPayment, and sendGoods to 20, 20, 1, and 1 respectively. Notice that we set the cost of manufactureGoods higher than sendPayment and of earnInterest higher than sendGoods. Here we assume the costs are centrally set; normally it is ill-founded to compare preferences across agents.
Table 7.3: Domain-specific methods and operators for the purchase scenario.

(operator : !manufactureGoods(A),
   pre : (¬goods ∧ (A = customer)),
   del :(), add : (goods),
   cost : 20)
(operator : !learnInterest(A),
   pre : (¬goods ∧ (A = merchant)),
   del :(), add : (pay),
   cost : 20)
(operator : !sendPayment(A, D),
   pre : (¬pay ∧ (A = customer) ∧ (D = merchant)),
   del :(), add : (pay),
   cost : 1)
(operator : !sendGoods(A, D),
   pre : (¬goods ∧ (A = merchant) ∧ (D = customer)),
   del :(), add : (goods),
   cost : 1)
(method : achieveGoal(G),
   pre : (G = GCG ∧ goal(G, customer) ∧ activeG(G)),
   tn : (!manufactureGoods(customer)))
(method : achieveGoal(G),
   pre : (G = GMP ∧ goal(G, merchant) ∧ activeG(G)),
   tn : (!learnInterest(merchant)))
(method : achieveGoal(G),
   pre : (G = GMP ∧ goal(G, merchant) ∧ activeG(G)),
   tn : (!sendGoods(merchant, customer)))
(method : achieveGoal(G),
   pre : (G = GCP ∧ goal(G, customer) ∧ activeG(G)),
   tn : (!sendPayment(customer, merchant)))
(method : detach(C),
   pre : (C = CMC ∧ active(C)),
   tn : (!sendPayment(customer, merchant)))
(method : deliver(C),
   pre : (C = CMC ∧ active(C)),
   tn : (!sendGoods(merchant, customer)))
(method : detach(C),
   pre : (C = CMC ∧ active(C)),
   tn : (!sendGoods(merchant, customer)))
(method : deliver(C),
   pre : (C = CMC ∧ active(C)),
   tn : (!sendPayment(customer, merchant)))

Figure 7.1 shows a portion of the HTN tree rooted at the achieveGoals method, showing it invokes the customer’s method to achieve its goals $G_{cg}$, achieveGoal($C, G_{cg}$). The achieveGoal method invokes entice, detach, and deliver methods. The entice method invokes the create($C_{cm}$) operator to create a commitment. Since the commitment $C_{cm}$ is active, and the merchant has a goal $G_{mp}$ to get paid, the detach($C_{cm}$) method is invoked, which invokes the domain-dependent operator sendGoods. That satisfies
Figure 7.1: A decomposition tree for the purchase scenario.

$G_{mg}$ and $G_{cg}$, and detaches $C_{cm}$. Once the commitment $C_{cm}$ is detached, the $\text{deliver}(C_{cm})$ method is invoked. Finally, the domain-dependent operator $\text{sendPayment}$ is invoked, which satisfies $C_{cm}$. Following this path, we obtain the plan: $\text{create}(C_{cm})$, $\text{sendGoods}$, and $\text{sendPayment}$. The total cost of this plan is 3.

A second possible plan (not shown in Figure 7.1) is: $\text{manufactureGoods}$, and $\text{earnInterest}$. These two operators satisfy the goals $G_{mp}$ and $G_{cg}$. However, the total cost of this plan is 40, which is higher than the previous plan.

These plans and their costs represent concrete realizations for the goals and commitments defined in the agent system modeled, and serve two purposes. First, the existence of plans that achieve the goals and satisfy commitments represents a proof of realizability for the specified agent system. Second, when multiple plans exist, their individual costs enable agents to reason about optimal realizations of its goals either by an agent on its own, or through commitments to other agents.

### 7.3 First Order Formalization in HTN

In the purchase scenario, when there are multiple merchants, goods, and prices, theoretical approaches, e.g., [Chopra and Singh, 2006], do not handle them, though interestingly practical rule-based approaches, e.g., [Desai et al., 2005], can handle them. An initial need for modeling interactions such as this in multiagent systems is to instantiate a small model of an interaction multiple times and to keep distinct instances from interfering with each other, yet relating to each other as appropriate. For example, a purchase protocol may be instantiated by multiple parties for multiple goods sold at multiple prices. This need can be addressed in a straightforward manner via a first-order representation.

Specifically, existing frameworks cannot encode domains containing the patterns of behavior that follow.
**Piecemeal progress.** The customer may pay the merchant in installments. The challenge to accommodate here is of arithmetic: we would like to handle the possibility that, for example, a payment of $6 followed by a payment of $4 is treated as equivalent to a payment of $10. (Notice whether a payment can be split depends on the regulations of the domain in question.)

**Concession.** The merchant may balk at providing the goods (or goods above a certain value) in advance of any payment. Therefore, we might amend the protocol so that the customer makes a partial deposit first, upon which the merchant delivers the goods, upon which the customer makes the remaining payment. Unlike piecemeal progress, this scenario involves altering the structure of the commitments involved: the merchant is committing to providing the goods only upon receiving a deposit and the customer is committing to paying the remaining amount upon receiving the goods. Concession is loosely inspired by Yolum and Singh’s [Yolum and Singh, 2007] approach which deals with nesting commitments to reduce the apparent risk to each party in a protocol.

**Consolidation.** If a customer places two purchase orders in close succession, the merchant may ship both of the ordered goods in the same package. Likewise, the customer may pay for both orders via one check. This is a clear case of flexibility in enactment that multiagent protocols ought to support. To realize it requires a richer representation wherein some actions (e.g., delivery) may be associated with more than one protocol instance.

**Compensation.** The customer may return goods to the merchant and the merchant would issue a refund. The refund should match the goods returned. This should result from a straightforward application of the first-order representation. Additionally, the protocol should ordinarily ensure that for piecemeal payments, only the amount received may be refunded. Further, the protocol may build in some fraud-resistant measures, such as that a prior refund disables a subsequent refund or that the total amount refunded in successive protocol instances does not exceed some threshold.

We now develop the first-order logical rules, operators and methods in HTN formalism that operationalize the goal and commitment dynamics.

### 7.3.1 Commitment Dynamics

We represent a commitment as a tuple $⟨Ct, De, Cr, P, Q, \vec{C}v⟩$, where: $Ct$ is the commitment type; $De$ is the debtor of the commitment; $Cr$ is the creditor of the commitment; $P$ is the antecedent, an unquantified first-order formula; $Q$ is the consequent, an unquantified first-order formula; and $\vec{C}v$ is a list $[v_1, \ldots, v_n]$ of variables identifying specific instances of $Ct$. The first challenge in encoding commitments in a first-order setting, is in ensuring that the components of a commitment are connected through their shared variables. In order to accomplish that, we model the entire set of variables of a particular commitment.
within one predicate. Thus, the number of variables \( n \) for a commitment is equivalent to the sum of arities of all first-order predicates in \( P \), and \( Q \), so if \( P = p_{a_0}(t_{a_0}) \ldots p_{a_k}(t_{a_k}) \) and \( Q = p_{c_0}(t_{c_0}) \ldots p_{c_k}(t_{c_k}) \), then \( n = \sum_{i=0}^{ck} |t_i| \). Thus, for each commitment \( C = \langle Ct, De, Cr, P, Q, \vec{C}v \rangle \), where \( P \) is a formula \( \varphi \) and \( Q \) is a formula \( \kappa \) we define the following rules:

\[
p(C, Ct, \vec{C}v) \leftarrow \text{commitment}(C, Ct, De, Cr) \land \varphi \\
q(C, Ct, \vec{C}v) \leftarrow \text{commitment}(C, Ct, De, Cr) \land \kappa
\]

Given these two basic formulas from the commitment tuple, we define rules that compute a commitment’s state, which follow from Figure 2.1. The \textit{null} state for a commitment is “instance dependent”, as each commitment has a number of possible instantiations, depending on the variables of the antecedent. In order to accomplish this, each commitment instance has an associated var predicate containing the commitment identifier and the list of variables associated to the instance. An active commitment is \textit{conditional} if its antecedent \((p)\) is false, and is \textit{detached} otherwise. A commitment is active if it is not null, terminal, pending, or satisfied. Note that terminal is a shortcut for the states cancelled, released, or expired. A commitment is \textit{terminated} if it is released or it is cancelled when its antecedent is false. A commitment is \textit{violated} if it is cancelled when its antecedent is true. A commitment is \textit{satisfied} if it is not null and not terminal, and its consequent \((q)\) is true.

\[
\text{null}(C, Ct, \vec{C}v) \leftarrow \neg\text{var}(C, Ct, \vec{C}v) \\
\text{conditional}(C, Ct, \vec{C}v) \leftarrow \text{active}(C, Ct, \vec{C}v) \land \neg p(C, Ct, \vec{C}v) \\
\text{detached}(C, Ct, \vec{C}v) \leftarrow \text{active}(C, Ct, \vec{C}v) \land p(C, Ct, \vec{C}v) \\
\text{active}(C, Ct, \vec{C}v) \leftarrow \neg\text{null}(C, Ct, \vec{C}v) \land \neg\text{terminal}(C, Ct, \vec{C}v) \\
\quad \land \neg\text{pending}(C, Ct, \vec{C}v) \land \neg\text{satisfied}(C, Ct, \vec{C}v) \\
\text{terminated}(C, Ct, \vec{C}v) \leftarrow (\neg p(C, Ct, \vec{C}v) \land \text{canceled}(C, Ct, \vec{C}v)) \lor \text{released}(C, Ct, \vec{C}v) \\
\text{violated}(C, Ct, \vec{C}v) \leftarrow p(C, Ct, \vec{C}v) \land \text{canceled}(C, Ct, \vec{C}v) \\
\text{satisfied}(C, Ct, \vec{C}v) \leftarrow \neg\text{null}(C, Ct, \vec{C}v) \land \neg\text{terminal}(C, Ct, \vec{C}v) \land q(C, Ct, \vec{C}v) \\
\text{terminal}(C, Ct, \vec{C}v) \leftarrow \text{commitment}(C, Ct, De, Cr) \land (\text{canceled}(C, Ct, \vec{C}v) \\
\quad \lor \text{released}(C, Ct, \vec{C}v) \lor \text{expired}(C, Ct, \vec{C}v))
\]

As an example, detached\((C, Ct, \vec{C}v) \leftarrow \text{active}(C, Ct, \vec{C}v) \land p(C, Ct, \vec{C}v)\) means that a commitment is detached if it is active \((\text{active}(C, Ct, \vec{C}v))\) and its consequent \((p(C, Ct, \vec{C}v))\) is true.

Next, we encode the transitions from Figure 2.1 as the planning operators. For a commitment, the create operator adds the var predicate if the commitment is null. If a commitment is active, executing suspend adds the pending predicate. If a commitment is pending, executing reactivate deletes the
pending predicate. If a commitment is conditional and a timeout has occurred, then executing expire adds the expired predicate. If a commitment is active, executing cancel adds the cancelled predicate. If a commitment is active, executing release adds the released predicate.

\[
\text{⟨operator : !create}(C, Ct, De, Cr, \vec{C}v)\text{,}
\]
\[
\text{pre : (commitment}(C, Ct, De, Cr)) \land \text{null}(C, Ct, \vec{C}v),
\]
\[
\text{del : ()}, \text{add : (var}(C, Ct, \vec{C}v)),
\]
\[
\text{cost : 0}\rangle
\]
\[
\text{⟨operator : !suspend}(C, Ct, De, Cr, \vec{C}v)\text{,}
\]
\[
\text{pre : (commitment}(C, Ct, De, Cr)) \land \text{active}(C, Ct, \vec{C}v),
\]
\[
\text{del : ()}, \text{add : (pending}(C, Ct, \vec{C}v)),
\]
\[
\text{cost : 1}\rangle
\]
\[
\text{⟨operator : !reactivate}(C, Ct, De, Cr, \vec{C}v)\text{,}
\]
\[
\text{pre : (commitment}(C, Ct, De, Cr)) \land \text{pending}(C, Ct, \vec{C}v),
\]
\[
\text{del : (pending}(C, Ct, \vec{C}v)), \text{add : ()},
\]
\[
\text{cost : 1}\rangle
\]
\[
\text{⟨operator : !expire}(C, Ct, De, Cr, \vec{C}v)\text{,}
\]
\[
\text{pre : (commitment}(C, Ct, De, Cr)) \land \\
\quad \text{conditional}(C, Ct, \vec{C}v) \land \text{timeout}(C, Ct, \vec{C}v),
\]
\[
\text{del : ()}, \text{add : (expired}(C, Ct, \vec{C}v)),
\]
\[
\text{cost : 1}\rangle
\]
\[
\text{⟨operator : !cancel}(C, Ct, De, Cr, \vec{C}v)\text{,}
\]
\[
\text{pre : (commitment}(C, Ct, De, Cr)) \land \text{active}(C, Ct, \vec{C}v),
\]
\[
\text{del : ()}, \text{add : (canceled}(C, Ct, \vec{C}v)),
\]
\[
\text{cost : 1}\rangle
\]
\[
\text{⟨operator : !release}(C, Ct, De, Cr, \vec{C}v)\text{,}
\]
\[
\text{pre : (commitment}(C, Ct, De, Cr)) \land \text{active}(C, Ct, \vec{C}v),
\]
\[
\text{del : ()}, \text{add : (released}(C, Ct, \vec{C}v)),
\]
\[
\text{cost : 1}\rangle
\]

As an example, the cancel\((C, Ct, De, Cr, \vec{C}v)\) operator has the precondition that \(C\) is a commitment (commitment\((C, Ct, De, Cr)\)) with matching type \((Ct)\), debtor \((De)\), creditor \((Cr)\), and variables \((\vec{C}v)\). This operator adds predicate canceled\((C, Ct, \vec{C}v)\).
7.3.2 Goal Dynamics

We represent a goal as a tuple \( \langle Gt, X, Pg, S, F, \vec{Gv} \rangle \), where: \( Gt \) is the goal type; \( X \) is the agent that has the goal; \( Pg \) is the goal precondition; \( S \) is the success condition; \( F \) is the failure condition; and \( \vec{Gv} \) is a list of variables identifying specific instances of \( Gt \). Similarly to commitments, the number of variables for a commitment will be equivalent to the sum of arities of all first-order predicates in \( Pg, S \) and \( F \).

Likewise, for each goal \( G = \langle Gt, X, Pg, S, F, Gv \rangle \), where \( Pg \) is a formula \( \varpi \), \( S \) is a formula \( \varsigma \), and \( F \) is a formula \( \vartheta \) we define the following formulas for our HTN encoding:

\[
pg(G, Gt, \vec{Gv}) \leftarrow \text{goal}(G, Gt, X) \land \varpi \\
s(G, Gt, \vec{Gv}) \leftarrow \text{goal}(G, Gt, X) \land \varsigma \\
f(G, Gt, \vec{Gv}) \leftarrow \text{goal}(G, Gt, X) \land \vartheta
\]

We now define rules that compute a goal’s state following Figure 2.2.

\[
\text{null}(G, Gt, \vec{Gv}) \leftarrow \neg \text{var}(G, Gt, \vec{Gv}) \\
\text{inactiveG}(G, Gt, \vec{Gv}) \leftarrow \neg \text{null}(G, Gt, \vec{Gv}) \land \neg f(G, Gt, \vec{Gv}) \land \neg s(G, Gt, \vec{Gv}) \land \\
\neg \text{terminalG}(G, Gt, \vec{Gv}) \land \neg \text{suspendedG}(G, Gt, \vec{Gv}) \land \neg \text{activeG}(G, Gt, \vec{Gv}) \\
\text{activeG}(G, Gt, \vec{Gv}) \leftarrow \text{activatedG}(G, Gt, \vec{Gv}) \land \neg f(G, Gt, \vec{Gv}) \land \neg \text{satisfiedG}(G, Gt, \vec{Gv}) \land \\
\neg \text{terminalG}(G, Gt, \vec{Gv}) \land \neg \text{suspendedG}(G, Gt, \vec{Gv}) \\
\text{satisfiedG}(G, Gt, \vec{Gv}) \leftarrow \neg \text{null}(G, Gt, \vec{Gv}) \land \neg \text{terminalG}(G, Gt, \vec{Gv}) \land \neg \text{suspendedG}(G, Gt, \vec{Gv}) \land \text{pg}(G, Gt, \vec{Gv}) \land \\
\text{s}(G, Gt, \vec{Gv}) \land \neg f(G, Gt, \vec{Gv}) \\
\text{failedG}(G, Gt, \vec{Gv}) \leftarrow \neg \text{null}(G, Gt, \vec{Gv}) \land \neg \text{var}(G, Gt, \vec{Gv}) \land \text{f}(G, Gt, \vec{Gv}) \\
\text{terminatedG}(G, Gt, \vec{Gv}) \leftarrow \neg \text{null}(G, Gt, \vec{Gv}) \land \neg \text{var}(G, Gt, \vec{Gv}) \land \text{f}(G, Gt, \vec{Gv}) \land \text{dropped}(G, Gt, \vec{Gv}) \lor \text{aborted}(G, Gt, \vec{Gv}) \\
\text{terminalG}(G, Gt, \vec{Gv}) \leftarrow \text{goal}(G, Gt, X) \land \text{dropped}(G, Gt, \vec{Gv}) \lor \text{aborted}(G, Gt, \vec{Gv})
\]

Finally, we define planning operators encoding the goal state transitions from Figure 2.2.

\[
\text{operator} : \text{!consider}(G, Gt, X, \vec{Gv}), \\
\text{pre} : (\text{goal}(G, Gt, X) \land \text{null}(G, Gt, \vec{Gv}) \land \text{pg}(G, Gt, \vec{Gv})), \\
\text{del} : (), \text{add} : (\text{var}(G, Gt, \vec{Gv})), \\
\text{cost} : 1 \\
\text{operator} : \text{!activate}(G, Gt, X, \vec{Gv}), \\
\text{pre} : (\text{goal}(G, Gt, X) \land \text{inactiveG}(G, Gt, \vec{Gv})), \\
\text{del} : (), \text{add} : (\text{activatedG}(G, Gt, \vec{Gv})), \\
\text{cost} : 1 \\
\text{operator} : \text{!suspend}(G, Gt, X, \vec{Gv}),
\]

143
\( \text{pre} : (\text{goal}(G, Gt, X) \land \neg \text{terminal}(G, Gt, \vec{G}v) \land \neg \text{null}(G, Gt, \vec{G}v)), \)
\( \text{del} : (\text{activated}(G, Gt, \vec{G}v)), \text{add} : (\text{suspended}(G, Gt, \vec{G}v)), \)
\( \text{cost} : 1 \)

\( \text{operator} : \text{!reconsider}(G, Gt, X, \vec{G}v), \)
\( \text{pre} : (\text{goal}(G, Gt, X) \land \text{suspended}(G, Gt, \vec{G}v) \land \neg \text{terminal}(G, Gt, \vec{G}v) \land \neg \text{null}(G, Gt, \vec{G}v)), \)
\( \text{del} : (), \text{add} : (\text{suspended}(G, Gt, \vec{G}v)), \)
\( \text{cost} : 1 \)

\( \text{operator} : \text{!reactivate}(G, Gt, X, \vec{G}v), \)
\( \text{pre} : (\text{goal}(G, Gt, X) \land \text{suspended}(G, Gt, \vec{G}v) \land \neg \text{terminal}(G, Gt, \vec{G}v) \land \neg \text{null}(G, Gt, \vec{G}v)), \)
\( \text{del} : (\text{activated}(G, Gt, \vec{G}v)), \text{add} : (\text{suspended}(G, Gt, \vec{G}v)), \)
\( \text{cost} : 1 \)

\( \text{operator} : \text{!drop}(G, Gt, X, \vec{G}v), \)
\( \text{pre} : (\text{goal}(G, Gt, X) \land \neg \text{terminal}(G, Gt, \vec{G}v) \land \neg \text{null}(G, Gt, \vec{G}v)), \)
\( \text{del} : (), \text{add} : (\text{dropped}(G, Gt, \vec{G}v)), \)
\( \text{cost} : 1 \)

\( \text{operator} : \text{!abort}(G, Gt, X, \vec{G}v), \)
\( \text{pre} : (\text{goal}(G, Gt, X) \land \neg \text{terminal}(G, Gt, \vec{G}v) \land \neg \text{null}(G, Gt, \vec{G}v)), \)
\( \text{del} : (), \text{add} : (\text{aborted}(G, Gt, \vec{G}v)), \)
\( \text{cost} : 1 \)

### 7.4 Formalizing the Patterns

This section applies our approach to capture the patterns from Section 7.3. Table 7.4 formulates the goals and commitments of a customer and a merchant. \( G_1 \) is customer’s goal for the goods (\( \text{goods}(123) \)) with the precondition that the customer needs the goods (\( \text{needsGoods}(123) \)). The failure condition of \( G_1 \) is a deadline (\( \text{deadline}(123) \)) for receiving the goods. Here, 123 is a transaction identifier. The goal \( G_2 \) is similar to \( G_1 \) except with a different transaction identifier. \( C_1 \) is the customer’s commitment to the merchant for paying (\( \text{pay}($100, 123) \)) if the merchant provides the goods (\( \text{goods}(123) \)). Here, 123 is the transaction identifier, and $100 is the payment amount. The commitments \( C_3 \) and \( C_4 \) are similar to \( C_1 \) except with different transaction identifiers and payment amounts. \( C_2 \) is the merchant’s commitment to the customer to provide goods (\( \text{goods}(123) \)) if the customer pays (\( \text{pay}($20, 123) \)). Finally, \( C_5 \) is the merchant’s commitment to the customer to refund (\( \text{refundpaid}(123) \)) if the customer returns the goods (\( \text{return}(123) \)). The refund amount should equal the amount that the customer actually paid.
Table 7.4: Goals and commitments for the patterns.

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Goal or commitment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_1$</td>
<td>$GT_1$</td>
<td>$G$(CUST, needsGoods(123), goods(123), deadline(123))</td>
</tr>
<tr>
<td>$G_2$</td>
<td>$GT_1$</td>
<td>$G$(CUST, needsGoods(456), goods(456), deadline(456))</td>
</tr>
<tr>
<td>$C_1$</td>
<td>$CT_1$</td>
<td>$C$(CUST, MER, goods(123), pay($100, 123))</td>
</tr>
<tr>
<td>$C_2$</td>
<td>$CT_2$</td>
<td>$C$(MER, CUST, pay($20, 123), goods(123))</td>
</tr>
<tr>
<td>$C_3$</td>
<td>$CT_1$</td>
<td>$C$(CUST, MER, goods(123), pay($80, 123))</td>
</tr>
<tr>
<td>$C_4$</td>
<td>$CT_1$</td>
<td>$C$(CUST, MER, goods(456), pay($200, 456))</td>
</tr>
<tr>
<td>$C_5$</td>
<td>$CT_3$</td>
<td>$C$(MER, CUST, return(123), refundpaid(123))</td>
</tr>
</tbody>
</table>

**Piecemeal progress.** Figure 7.2 shows an HTN decomposition tree for piecemeal progress. The customer creates $C_1$ to achieve its goal $G_1$. (For clarity of presentation, we omit the goal operations *consider* and *activate* in the HTN decomposition trees.) The merchant detaches $C_1$ by sending the goods. This presumes that merchant has a goal to get paid. To satisfy $C_1$, the customer needs to pay $100, which the customer may pay either as a lump sum, or in two installments. Fig. 7.2 shows a plan in which the customer pays two installments of $50 each.

![Figure 7.2](image)

Table 7.5 shows the methods *satisfy* and *pay*, and operators *paid* and *updatepaid*. The *satisfy* method encodes the plans for paying the $100. The *pay* method implements the arithmetic to add up the payments for a transaction identifier. If the customer has paid an installment, then the *pay* method invokes the *updatepaid* operator, which deletes the previous paid predicate, and adds a paid predicate with the new amount. Otherwise, the *pay* method invokes the *paid* operator, which adds a paid predicate.

145
Table 7.5: Methods and operators for the patterns.

```
(method : (satisfy(C)),
  pre : (commitment(C, CT, cust, mer) \& var(C, CT, (cAmount, tID))
  \& detached(C, CT, (cAmount, tID)),
  tn : (pay(cust, mer, cAmount, tID)),
  pre : (commitment(C, CT, cust, mer) \& var(C, CT, (cAmount, tID))
  \& detached(C, CT, (cAmount, tID)),
  tn : (pay(cust, mer, cAmount/2, tID) \& pay(cust, mer, cAmount/2 + cAmount%2, tID))
  pre : (commitment(C, CT, mer, cust) \& var(C, CT, (cAmount, tID))
  \& detached(C, CT, (cAmount, tID)),
  tn : (goods(mer, cust, tID))
  pre : (commitment(C, CT, mer, cust) \& var(C, CT, (tID)) \& detached(C, CT, (tID))),
  tn : (refundpaid(mer, cust, tID)))
(method : (satisfy(C, C)),
  pre : (commitment(C, CT, cust, mer) \& var(C, CT, (c1Amount, tID))
  \& detached(C, CT, (c1Amount, t1ID)) \& commitment(C, CT, cust, mer)
  \& var(C, CT, (c2Amount, t2ID)) \& detached(C, CT, (c2Amount, t2ID))),
  tn : (paytogether(cust, mer, amount, tID, t2ID)))
(method : (paytogether(cust, mer, amount, t1ID, t2ID)),
  pre : (commitment(C, CT, cust, mer) \& var(C, CT, (c1Amount, t1ID))
  \& detached(C, CT, (c1Amount, t1ID)) \& commitment(C, CT, cust, mer)
  \& var(C, CT, (c2Amount, t2ID)) \& detached(C, CT, (c2Amount, t2ID))
  \& (amount = c1Amount + c2Amount)),
  tn : (pay(cust, mer, c1Amount, t1ID) \& pay(cust, mer, (amount – c1Amount), t2ID))
(method : (paytogether(cust, mer, amount, tID)),
  pre : (commitment(C, C, cust, mer) \& var(C, C, (cAmount, tID))
  \& paid(cust, mer, oldAmt, tID)),
  tn : (updatepaid(cust, mer, (oldAmt + amount), tID))
  pre : (commitment(C, C, cust, mer) \& var(C, C, (cAmount, tID)),
  tn : (paid(cust, mer, amount, tID)))
(method : (!paid(cust, mer, amount, tID)),
  pre : (!agent(cust) \& !agent(mer)),
  del : (:), add : (paid(cust, mer, amount, tID)))
(method : (updatepaid(cust, mer, amount, tID)),
  pre : (!agent(cust) \& !agent(mer) \& paid(cust, mer, oldAmount, tID)),
  del : (paid(cust, mer, oldAmount, tID)), add : (paid(cust, mer, amount, tID)))
```

Concession. This pattern involves two commitments. The merchant commits \((C_2)\) to providing the goods upon receiving a deposit of $20. The customer commits \((C_3)\) to the merchant to pay the remaining amount of $80 upon receiving the goods. Fig. 7.3 shows an HTN decomposition tree for concession. The customer and the merchant create \((C_2)\) and \((C_3)\), respectively. Then the customer detaches \((C_2)\) by paying $20. The merchant satisfies \((C_2)\) by providing the goods, which also detaches \((C_3)\). Next, the customer pays $80 to satisfy \((C_3)\). The detach method has a structure similar to the satisfy method.
Consolidation. In this pattern, the customer has a second goal $G_2$ for goods(456). $C_4$ is the commitment from the customer to the merchant to paying $200 if the merchant provides the goods. Fig. 7.4 shows an HTN decomposition tree for consolidation. To achieve its goals $G_1$ and $G_2$, the customer creates commitments $C_1$ and $C_4$. The merchant detaches $C_1$ and $C_4$ by shipping the goods (goods(123) and goods(456)) together using the shiptogther method. The customer satisfies $C_1$ and $C_4$ by making a consolidated payment of $300 to the merchant. Table 7.5 shows the paytogether method. This method splits the $300 into $100 and $200, and applies them to the transactions 123 and 456.
### Compensation

In this pattern, the merchant commits \((C_5)\) to the customer to refunding the amount paid by the customer if the customer returns the goods. Fig. 7.5 shows an HTN decomposition tree for compensation. The customer and the merchant create \(C_1\) and \(C_5\), respectively. The merchant detaches \(C_1\) by providing the goods, and the customer makes a partial payment of $50 to the merchant. Next, the customer returns the goods, which detaches commitment \(C_5\). The merchant satisfies \(C_5\) by refunding $50 to the customer. Notice that the \textit{refundpaid} method identifies and refunds the actual amount paid by the customer.

![Figure 7.5: A decomposition tree for the compensation pattern.](image)

### 7.5 Conclusion

HTN planning provides a promising approach to provide depth of representation and reasoning to conceptualizing the connection between commitments and goals. We motivate some key patterns of reasoning about and enacting commitments and show how our approach naturally accommodates them. We articulate a first-order approach to represent commitments and the generation of plans that lead them to suitable states (satisfied in a happy path). Such a formalization not only affords greater expressivity than currently existing approaches, but also allows one to employ an off-the-shelf HTN planner as a validation tool for a business process.

### 7.6 Related Work

Chopra and Singh [2006] employ \(C^+\), an action description language, to model commitment protocols so they can be contextually adapted. Günay et al. [2012] propose an algorithm to automatically create commitment protocols that would achieve agent goals by matching goals to local capabilities and services.
from third parties. By contrast, our formalization applies an HTN planner such as JSHOP2 not only to create such plans and commitment protocols, but also to optimize the results based on the cost of the operators. Further, it supports visualizing the generation of a protocol in a readable way.

Chopra et al. [2010c,b] develop a semantic relationship between goals and commitments. Their approach can verify if a set of commitments supports achieving a set of agent goals, and if a set of agent goals supports satisfying a set of commitments. In contrast, given a set of goals and commitments, our approach produces operational level and feasible plans that lead to satisfaction of the goals and commitments.

Lee et al. [2012] propose Business-OWL (BOWL), an ontology to capture the operational level knowledge from the industry standards such as RosettaNet and OAGI. They identify two high-level goals, buy and sell, that these standards operationalize. Further, they propose a variant of HTN planning to generate plans starting from a goal. The plans specify the operational level tasks that would achieve the given goal. In contrast to BOWL, our approach is more general and is based upon a business model specification that includes commitments in addition to goals. Similar to BOWL, we employ HTN planning for generating operational plans. However, unlike BOWL, we develop domain independent axioms and operators for commitments and goals. We additionally develop a first-order representation of commitments and goals, which enables reasoning of complex patterns such as piecemeal, consolidation, and compensation.

Wu et al. [2003] employ SHOP2 HTN planning for composing web services specified using web ontology language for services (OWL-S). Their approach translates the OWL-S specifications of web services to SHOP2 domains, and the OWL-S specification of composition tasks to SHOP2 planning problem. In contrast to Wu et al. [2003]’s approach, our approach is founded upon high-level business model specifications based on commitments and goals.
Chapter 8

Discussion

8.1 Conclusions

This section summarizes the key conclusions of this dissertation.

Metamodel, patterns, and verification: This dissertation presents a commitment-based business metamodel, a set of modeling patterns, and an approach for formalizing business models and verifying operations with respect to models. We consider dialectical commitments in addition to the practical commitments. We evaluate our proposal using real-life scenarios from the supply chain and healthcare domains. Our core set of modeling patterns shows how we may construct additional patterns. Our approach helps a business modeler concentrate on high-level commitments, and helps detect flaws in business process implementations.

Comma: This dissertation presents Comma, a methodology for business modeling founded upon our metamodel and patterns. We conducted rigorous developer studies comparing Comma with RosettaNet, a supply chain standard, and HL7, a healthcare standard. The studies confirmed that Comma performs better on quality measures of coverage, precision, structure, flexibility, and abstraction. However, the studies revealed that Comma is slightly difficult. We conjecture that improved tooling and training would alleviate the difficulty.

Relating goals and commitments: This dissertation studies the complementary aspects of commitments and goals by establishing an operational semantics of the related lifecycles of the two concepts. We have distinguished the purely semantic aspects of their lifecycles from the pragmatic aspects of how a cooperative agent may reason, and demonstrated desirable properties such as the convergence of agent states. From the viewpoint of agent programming, we have provided a foundational set of rules that is complete in a technical sense. We develop maintenance commitments as a first-class construct, enabling our main contribution of a cohesive account of maintenance goals and commitments.
Hierarchical planning about commitments and goals: HTN planning is a promising approach to provide depth of representation and reasoning to conceptualizing the connection between commitments and goals. We motivate some key patterns of reasoning about commitments and goals and show how our approach naturally accommodates them. We articulate a first-order approach to represent commitments and the generation of plans that lead them to suitable states (satisfied in a happy path). Such a formalization not only affords greater expressivity than currently existing approaches, but also allows one to employ an off-the-shelf HTN planner as a validation tool for a business process.

8.2 Future Directions

In this section, we outline some of our key future directions.

Comma: On the theoretical side, we are considering expanding Comma to account for a richer variety of norms, e.g., in the spirit of Aldewereld et al. [2010], than just commitments. On the practical side, enhanced tooling is an obvious theme. A natural extension would be to support model-driven engineering using Comma. Further, we will enhance Comma so it provides guidance for situations where a model must be modified to accommodate evolving requirements. On the empirical side, we will conduct additional developer studies. Specifically, although our study design mitigated many important threats to validity that can arise in a comparative study, it did not consider important challenges to business interoperation in practice, such as dealing with a legacy system. We conjecture that increasing the complexity of a scenario will tilt the balance further in favor of commitment-based approaches: we defer such evaluations to future research. Further, a threat to validity of any empirical evaluation is whether the subjects correspond closely to the target population (industry practitioners, in our case) in their expertise, experience, and motivation. In their broadest scope, such problems are not readily amenable to comparative research studies, but we plan to explore simplified versions of them.

Commitments and goals: Directions for building on our operational semantics include considering a hierarchy of prioritized goals or commitments, and extending our semantics to include delegation and assignment of commitments (raising the situation of multiple agents and multiple commitments), shared goals, or plans, and additional cognitive state, such as desires or intentions. We have limited the discussion of our joint commitments and goals semantics to the situation of a single commitment and a single (end) goal, and to two agents. We conjecture that our results are general enough to handle the aggregate situation where there are multiple commitments and goals between two or more agents, such as C(x, y, . . .), C(y, z, . . .) and C(z, x, . . .). Intuitively, since in Theorem 1 the parameters y and u are unbound, a convergent execution path possibly can be
established by repeated application of that theorem. This more general situation must be formally proved.

**Planning:** Our HTN planning approach brings cost into consideration for reasoning about goals and commitments. However, it assumes a fully cooperative setting that takes into account the total cost over all participants. A natural and important research challenge is to extend the approach to handle self-interested agents who might wish to optimize for themselves individually even if the overall cost of the plan is increased as a result. Such agents may negotiate with each other or may even disclose their costs strategically. Another natural and important challenge is to apply our approach to address the design-time reasoning about a protocol so as to generate an operational representation. Previous approaches can produce finite state machine representations. Such representations do not provide a basis for incorporating costs to limit the execution paths that would be suitable for a particular agent based on the costs they can bear. Finally, although our approach can be used to generate protocol realizations when simultaneous goals and commitments do not conflict, and detect when conflicts make realizations impossible, it cannot currently resolve such conflicts automatically (e.g., suggest different methods to overcome conflicts). Therefore, we leave the design of planning tools to overcome domain-knowledge conflicts as future work.
REFERENCES


Appendix A

Metamodel, Patterns, and Verification

Mapping Sequence Diagrams to NuSMV Modules

The UML 2.0 specification Obj [2004] defines the sequence diagram to model the behavior of a system. A sequence diagram represents each participant as a vertical line. A directed horizontal line between any two participants represents a message. The ordering of the horizontal lines gives the temporal order of the messages. To modularize large and complex interactions, the specification further defines operators \textit{alt}, \textit{opt}, and \textit{loop}.

We now describe how we build a NuSMV FSM from a sequence diagram. A sequence diagram yields a finite state machine with Boolean state variables, one per message. A state variable is false until the participants exchange the corresponding message, at which time it becomes true. We assume that the messages do not cross, that is, messages arrive at a recipient in the same order in which they are sent by a sender. In the finite state machine, each message exchange causes a state transition. We consider four fundamental patterns that compose a sequence diagram: a pattern containing $n$ message exchanges, a pattern containing \textit{alt} (alternate), a pattern containing \textit{opt} (optional), and a pattern containing \textit{loop}. An algorithm maps each of these patterns to the NuSMV language, and composes them to obtain the mapping for a sequence diagram. Due to lack of space, we cannot present the details of the mapping.

\begin{quote}
\textbf{Listing A.1: FSM model (in NuSMV) for Figure 3.15(c).}

1 VAR
2 reqQuoteDS: boolean;
3 quoteSD: boolean;
4 ...
5
6 ASSIGN
7 init(reqQuoteDS) := 0;
8 init(quoteSD) := 0;
\end{quote}
As an example, Listing A.1 shows the mapping of sequence diagram from Figure 3.15(c) to NuSMV. The VAR and the ASSIGN sections of this listing declare and initialize the FSM state variables respectively. Lines 11–15 declare a transition from a state in which the guard confirmOrderDC holds and reqQuoteDS does not hold, to a state in which both of them hold. In this transition, the rest of the variables remain unchanged between the two states.

**Cancel Order Scenario**

Section 3.3.1 models the scenario in which a customer places an order. At a later time, the customer may decide to cancel the order. This section presents revert commercial transaction pattern, which enables modeling of scenarios such as order cancellation.

**Revert Commercial Transaction**

**Intent:** In the commercial transaction pattern, trading partners commit to executing certain tasks for each other. This pattern captures the scenario in which the partners desire to cancel the commercial transaction.

**Motivation:** In a commercial transaction, a buyer and a seller agree to exchange goods for payment. Later the buyer or the seller may desire to cancel the exchange.

**Implementation:** In case of cancellation, every debtor commits to undoing the tasks that the creditor executed, that is the antecedent, and releases the creditor from the reciprocal commitment. Figure A.1 shows this pattern. The meaning of undo depends upon the task. For example, undo(goods) means returning the goods, and undo(pay) means refunding the payment.
**Consequences:** This pattern applies in conjunction with the commercial transaction pattern.

**Applying revert commercial transaction pattern**

To each of the commercial transactions in the QTC model from Section 3.3.1, we apply the revert commercial transaction pattern. Next, we describe a few applications of this pattern.

- **Commercial transaction between the customer and the reseller:** Figure A.2 shows this pattern.

  If the customer or the reseller request cancellation (cancel), the customer commits (C₁₈) to returning the goods if the goods are shipped, paying for the installation if the installation is done (undo(install ∧ shipGoodsE)), and releasing the reseller from C₂ and C₃. Conversely, the reseller commits (C₁₉) to the customer to refunding the payment (undo(payR)) if the customer paid, and releasing the customer from C₁.

- **Commercial transaction between the reseller and the distributor:** Figure A.3 shows this pattern.

  As Section 3.3.1 describes, the reseller outsources the shipping of goods to the distributor, and the outsourcing pattern (C₄, C₅, and C₆) captures it. The outsourcing pattern includes a commercial transaction pattern (C₄ and C₅) to which we apply the revert pattern. The reseller commits (C₂₀) to the distributor to returning the goods if the distributor shipped the goods (undo(shipGoodsE)), and to releasing the reseller from C₅. Note that the reseller commits to undoing shipGoodsE which is the consequent of the commitment C₆, instead of undoing create of C₆. The distributor commits to
Figure A.2: Revert Commercial transaction: Customer purchases the goods and the installation service from the reseller.

\[ C_{18} \quad C(\text{CUSTOMER, RESELLER, cancel}(C_1), \text{undo}(\text{install} \land \text{shipGoodsE}) \land \text{release}(C_2 \land C_3)) \]
\[ C_{19} \quad C(\text{RESELLER, CUSTOMER, cancel}(C_2 \land C_3), \text{undo}(\text{payR}) \land \text{release}(C_1)) \]

Figure A.3: Revert Commercial transaction: Reseller outsources the shipping of the goods to the distributor.

\[ C_{20} \quad C(\text{RESELLER, DISTRIBUTOR, cancel}(C_4), \text{undo}(\text{shipGoodsE}) \land \text{release}(C_5)) \]
\[ C_{21} \quad C(\text{DISTRIBUTOR, RESELLER, cancel}(C_5), \text{undo}(\text{payD}) \land \text{release}(C_4)) \]

the reseller to refunding the payment if the reseller paid (undo(payD)), and to releasing the reseller from C₄.
Similarly, the revert commercial transaction pattern applies to the remaining commercial transactions in the QTC model.

![Diagram](image)

(e) A distributor requests a quote from Cisco.

(f) Cisco requests a quote from a shipper.

(g) The shipper ships goods to the distributor.

(h) A customer requests a quote from Cisco to purchase product support service.

Figure A.4: Operational interactions in a QTC implementation.

**Business Model for Insurance Scenario**

To substantiate the generality of the patterns from Section 3.2 we evaluate them on a second real-world scenario involving AGFIL, an insurance company in Ireland Browne and Kellett [1999]. AGFIL provides vehicle insurance service to the policy holders. As part of the vehicle insurance service, AGFIL provides claim reception and vehicle repair to the policy holders. AGFIL outsources the claim reception to Europ Assist, a call center provider, and vehicle repair to various repairers in its partner network. To guard against fraud, AGFIL needs to assess claims, which it outsources to Lee Consulting Services.
Lee Consulting Services employs inspectors to inspect the vehicle damage. It negotiates and approves the repair charge as reported by the repairers. Further, it presents invoice of repair charge to AGFIL. Figure A.6 shows the AGFIL cross-organizational scenario expressed as a business model. The figure describes the commitments that model the scenario.

A policy holder and AGFIL (insurer) negotiate and agree upon the insurance payment. The commercial transaction pattern models this interaction. The policy holder commits (C₁) to the insurer to paying if the insurer creates commitments C₃ and C₄. The insurer commits (C₂) to create C₃ and C₄ if the policy holder pays the insurer. As part of the insurance service, the insurer commits (C₃) to receiving a claim if the policy holder reports an accident, and commits (C₄) to repairing the vehicle if the policy holder requests repair service.

AGFIL (insurer) outsources the claim reception to Europ Assist (call center). The outsourcing pattern models this interaction. The call center and the insurer agree upon the payment for the claim reception service, and create the reciprocal commitments C₅ and C₆. The insurer commits (C₅) to the call center to paying if the call center creates commitment C₇, and the call center commits (C₆) to the insurer to creating C₇ if the insurer pays the call center. In this case, C₇ is the outsourced commitment. The call center commits (C₇) to the policy holder to receiving the claim if the policy holder reports an accident.

AGFIL (insurer) and Lee Consulting Services (assessor) negotiate the assessment fees. The commercial transaction pattern models this interaction. The insurer and the assessor create commitments C₈ and C₉. The insurer commits (C₈) to the assessor to paying the assessment fees if the assessor assesses the claim and brings about agreement to repair. Conversely, the assessor commits (C₉) to the insurer to assess the claim if the insurer pays the assessor.
The assessor and the inspector negotiate the vehicle inspection fees. The *commercial transaction pattern* models this interaction. The assessor and the inspector create reciprocal commitments $C_{10}$, and $C_{11}$.

The insurer outsources the vehicle repair to a repairer. The *outsourcing pattern* models this interaction. The insurer, and the repairer create the commitments $C_{12}$, $C_{13}$, and $C_{14}$. The insurer commits ($C_{12}$)
commits to the repairer to paying if the repairer creates the commitment $C_{14}$. Conversely, the repairer commits ($C_{13}$) to the insurer to create commitment $C_{14}$ if the insurer pays the repairer. Here, $C_{14}$ is the outsourced commitment. The repairer commits ($C_{14}$) to the policy holder to repairing the vehicle if the policy holder requests for the repair service.
Appendix B

Modeling Healthcare Processes

Sequence Diagrams for Breast Cancer Screening and Diagnosis Scenario

We now present the complete set of sequence diagrams. Figure B.1 represents the biopsy scenario. In Figure B.1(a) PHYSICIAN on finding PATIENT’s tumor suspicious recommends PATIENT for biopsy to which PATIENT agrees and creates C₃. Then, PHYSICIAN requests RADIOLOGIST for conducting biopsy as shown in Figure B.1(b). RADIOLOGIST agrees to it and creates C₄. In Figure B.1(c), RADIOLOGIST requests PATIENT to arrive for biopsy and PATIENT arrives. Finally, RADIOLOGIST notifies PATIENT’s arrival to PHYSICIAN discharging C₃.

Figure B.2 represents interactions between PATHOLOGIST and RADIOLOGIST for processing PATIENT’s diagnosis report. In Figure B.2(a)(a), RADIOLOGIST requests PATHOLOGIST for conducting a lab examination on PATIENT’s tissue sample to which PATHOLOGIST agrees. As a result, C₆ is created by PATHOLOGIST toward RADIOLOGIST. When PATHOLOGIST provides the lab examination report to RADIOLOGIST then C₆ is discharged. Now, RADIOLOGIST takes the report from PATHOLOGIST and provides a combined Pathology-Radiology report to PHYSICIAN, declaring this to be the final report. When RADIOLOGIST does so, she discharges C₄ and creates a dialectical commitment D₁. PHYSICIAN forwards the report to PATIENT, thereby discharging C₁ and creates a dialectical commitment D₃.

Figure B.3 describes PATIENT’s registration. If the diagnosis report provided by PHYSICIAN to PATIENT suggests PATIENT has breast cancer, then C₇ is created. When PATHOLOGIST reports PATIENT’s details to REGISTRAR, then C₈ is created, as shown in Figure B.3(a). PATHOLOGIST also reports the registration to HOSPITAL and discharges C₇. Finally, on the request of PATHOLOGIST, REGISTRAR registers PATIENT to the cancer registry and notifies HOSPITAL, thereby discharging C₈.

Figure B.4 describes the scenario where RADIOLOGIST provides its’ results to PATHOLOGIST and based on the results, PATHOLOGIST decides either to agree or requests an input from TUMOR BOARD. TUMOR BOARD upon request provides an input that suggests either it agrees or disagrees with the results reported. If TUMOR BOARD disagrees then RADIOLOGIST cancels its dialectical commitment D₄ and
Figure B.1: Biopsy scenario.

agrees with TUMOR BOARD.

Figure B.5 describes the scenario where PATHOLOGIST provides its’ results to RADIOLOGIST and based on the results, RADIOLOGIST decides either to agree or requests an input from TUMOR BOARD. TUMOR BOARD upon request provides an input that suggests either it agrees or disagrees with the results reported. If TUMOR BOARD disagrees then PATHOLOGIST cancels its dialectical commitment D₅ and agrees with TUMOR BOARD.

Figure B.6 describes the scenario where RADIOLOGIST provides joint results to PHYSICIAN and based on the results, PHYSICIAN decides either to agree or requests an input from TUMOR BOARD.
(a) **RADIOLOGIST** provides tissue samples to **PATHOLOGIST**.

(b) **PHYSICIAN** conveys diagnosis report from **RADIOLOGIST** to **PATIENT**.

**Figure B.2**: Diagnosis reports.

(a) **PATHOLOGIST** reports **PATIENT’s** details to **REGISTRAR** and **HOSPITAL**.

(b) **REGISTRAR** adds **PATIENT** in the cancer registry.

**Figure B.3**: **PATIENT’s** registration.

**TUMOR BOARD** upon request provides an input that suggests either it agrees or disagrees with the results reported. If **TUMOR BOARD** disagrees then **RADIOLOGIST** cancels its dialectical commitment $D_3$ and agrees with **TUMOR BOARD**.

Figure B.7 describes the scenario where **PHYSICIAN** provides its’ results to **PATIENT** and based on the results, **PATIENT** decides either to agrees or requests an input from **TUMOR BOARD**. **TUMOR BOARD** upon request provides an input that suggests either it agrees or disagrees with the results reported. If **TUMOR BOARD** disagrees then **PHYSICIAN** cancels its dialectical commitment $D_1$ and agrees with **TUMOR BOARD**.

173
Figure B.4: PATHOLOGIST requests TUMOR BOARD to review the RADIOLOGIST results.
Figure B.5: RADIOLOGIST requests TUMOR BOARD to review the PATHOLOGIST results.
Figure B.6: PHYSICIAN requests TUMOR BOARD to review the joint RADIOLOGIST and PATHOLOGIST results.

Figure B.7: PATIENT requests TUMOR BOARD to review the PHYSICIAN results.
Appendix C

Protos: Business Modeling Tool

Protos is an Eclipse-based tool that implements our proposed approach. It enables: (a) the development of a high-level business model by composing the reusable patterns, (b) the development of UML 2.0 sequence diagrams, as a low-level operational representation, and (c) the automated verification of the UML 2.0 sequence diagrams with respect to the high-level business model.

Tool Architecture

![Diagram of Protos tool architecture]

Figure C.1: Protos tool architecture.
Figure C.1 shows the architecture of the Protos tool. This architecture corresponds to our conceptual architecture from Figure 3.13. The Protos tool consists of five key components: (1) business modeler, (2) UML sequence diagram modeler, (3) Protos engine, (4) Protos parser, and (5) NuSMV model-checker.

**The Business Modeler** is implemented as an Eclipse plugin using the Eclipse Modeling Framework (EMF) [Eclipse, 2012a] and the Graphical Modeling Framework (GMF) [Eclipse, 2012b]. The concepts and the constraints of the business metamodel are specified in an EMF ECore model. The graphical aspects of the tool such as the concept icons, the connectors, and the menus are specified in the GMF model. An Eclipse plugin of the tool is then generated using the GMF framework. The business modeler enables saving a business model as an ECore model instance file.

Figure C.2: Protos business modeler.

Figure C.2 shows a screenshot of the business modeler presenting a model of a real-life Quote-to-Cash business process.

**The UML Sequence Diagram Modeler** is part of IBM’s Rational Software Architect (RSA) version
IBM RSA is an Eclipse-based tool that supports developing UML 2.0 sequence diagrams. The UML sequence diagram modeler can output a sequence diagram in a standard format.

Figure C.3 shows a screenshot of the UML sequence diagram modeler from IBM RSA, presenting a sequence diagram with buyer and seller lifelines. We capture the meaning of a message as an annotation: for example, the accept message means the creation of the commitment $C_1 = C(\text{Buyer, Seller, goods, pay})$.

**NuSMV** [NuSMV, 2012]: Section 2.5 from Chapter 2 describes NuSMV in detail.
**The Protos Engine** is a Perl script that (a) invokes the Protos parser APIs to generate a NuSMV input file, (b) invokes the NuSMV model checker on the generated file, and (c) parses NuSMV’s output to generate user-friendly output showing the result.

**The Protos Parser** is the heart of the Protos tool. The parser implements our verification approach from Chapter 3 [Telang and Singh, 2012] to generate the CTL specifications from the business model ECore file, and to generate finite state machine in NuSMV input language from the UML model file.