TRANSITION IN MULTIAGENT ORGANIZATIONS

Eric Matson

October 22, 2008

B.S. Computer Science, Kansas State University, 1988
M.B.A. Operations Management, The Ohio State University, 1993
M.S.E. Software Engineering, Kansas State University, 2002
A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the
Department of Computer Science
College of Engineering
University of Cincinnati
Cincinnati, OH 45221-0030
matsonet@email.uc.edu
Organizations exist to solve any number of goals or problems. Typically, an organization is tasked with solving problems an individual cannot resolve without assistance from others. Human organizations serve as an inspiration to create models of organization which can be applied to agents and multiagent systems. Capturing how humans form and interact with a capability-based organization model is a complex task. Applying the same model to software agents requires essentially the same structures and relationships as with human organization models. There have been many research efforts capturing specific functional elements of organization applied to a single problem domain or a similar set of problem domains. In these cases, the research and solution may work and possess functionality to solve the unique problem, but won’t necessarily translate into other problem domains or capture the general function of an organization.

The primary goal of my dissertation is to develop and formalize an organization model and transition algorithms, generic in definition but capable of capturing organization dynamics in domains in which intelligent agents act. The model’s generic nature will allow it to be applied to any problem where organization is required. The organization model will have structural, state and transitional elements. While the scope of this research will include the establishment and design of structural and state elements, the main scope of work will be in the formal definition, development and evaluation of transition processes and algorithms.

The transition formalization will include a formal transition model and relations, central, distributed and segmented transition algorithms. The model and algorithms will be formally analyzed. The formal model will be implemented for empirical evaluation and analysis. The formal and implemented instances will then be compared for evaluation using several task domains used to validate the model.
Acknowledgments

I would like to recognize and thank each of the people who have graciously agreed to spend their time to participate in my research committee as the research proposal goes to a complete product.

Committee Members

Prof. Raj Bhatnagar (Advisor/CS)
Prof. Ernie Hall (CS/MINE)
Prof. Ali Minai (ECE)
Prof. Carla Purdy (ECE)
Prof. John Schlipf (CS)
Prof. Scott DeLoach (CIS), Kansas State University
## Contents

1 Introduction ........................................ 1
   1.1 Motivation ........................................ 3
   1.2 Contribution ..................................... 4
   1.3 Thesis Outline .................................. 4

2 Literature Review .................................. 6
   2.1 Organization Background and Theory .......... 6
   2.2 Organization Models and Transition .......... 9
      2.2.1 Organization Models ....................... 10
      2.2.2 Structure .................................. 11
      2.2.3 Transitional Theory ....................... 13
      2.2.4 Previous Publications ..................... 15
   2.3 Summary .......................................... 15

3 Organization Model ................................ 16
   3.1 Organization Model ............................. 17
   3.2 Structure ....................................... 17
      3.2.1 Structural Objects ......................... 18
      3.2.2 Structural Relations ....................... 21
   3.3 State ........................................... 24
      3.3.1 State Objects .............................. 25
4.2.1 Computing Initial Organization ........................................... 67
4.2.2 Computing Reorganization .................................................. 72
4.2.3 Evaluation of Generic Initial Organization ................................. 77
4.2.4 Evaluation of Generic Reorganization ....................................... 80
4.3 Agent Perspective ............................................................... 83
4.4 Central Control Transition Algorithm ......................................... 87
  4.4.1 Initial Organization .......................................................... 87
  4.4.2 Evaluation of Central Control Initial Organization ....................... 90
  4.4.3 Reorganization ............................................................... 92
  4.4.4 Evaluation of Central Control Reorganization ............................ 93
4.5 Distributed Control Transition Algorithm - Complete Knowledge ......... 96
  4.5.1 Distributed Control Initial Organization - Complete Knowledge ....... 96
  4.5.2 Evaluation of Distributed Control Initial Organization - Complete Knowledge 98
  4.5.3 Distributed Control Reorganization - Complete Knowledge ............ 100
  4.5.4 Evaluation of Distributed Control Reorganization - Complete Knowledge . 100
4.6 Distributed Control Transition Algorithm - Local Knowledge ............... 104
  4.6.1 Distributed Control Initial Organization - Local Knowledge ............ 104
  4.6.2 Evaluation of Distributed Control Initial Organization - Local Knowledge . 107
  4.6.3 Distributed Control Reorganization - Local Knowledge ................ 109
  4.6.4 Evaluation of Distributed Control Reorganization - Local Knowledge .... 115
4.7 Segmented Control Transition Algorithm ...................................... 118
4.7.1 Segmented Control Initial Organization ........................................ 118
4.7.2 Evaluation of Segmented Control Initial Organization .................... 125
4.7.3 Segmented Control Reorganization ............................................. 127
4.8 Summary ............................................. 132

5 Analysis and Validation .................................................. 135

5.1 Algorithm Analysis ........................................................ 135
  5.1.1 Sparse and Dense Organizations ......................................... 137
  5.1.2 Analysis Experimental Setup ............................................ 137
  5.1.3 Generic Initial Organization ............................................. 140
  5.1.4 Generic Reorganization ................................................ 142

5.2 Algorithm Comparison ..................................................... 144
  5.2.1 Initial Organization Comparison ......................................... 144
  5.2.2 Reorganization Comparison .............................................. 148

5.3 Validation ............................................................... 150
  5.3.1 Small Software Development Organization ............................ 152
  5.3.2 Large Software Development Organization ............................ 158
  5.3.3 Sensor Organizations .................................................. 160

5.4 Summary ............................................................... 171

6 Conclusions ............................................................ 172

6.1 Contribution of Research .................................................. 172
6.2 Impact .................................................. 173
6.3 Future Work ............................................. 173

A Knowledge Base Approach 175

A.1 Realization .............................................. 175
A.2 Design .................................................. 176
A.3 Implementation ....................................... 176
A.4 Initial Results ......................................... 181
A.5 Evaluation .............................................. 183
   A.5.1 Initial organization .............................. 184
   A.5.2 Reorganization ................................... 186
   A.5.3 Simple Properties .............................. 186
   A.5.4 Adding Objects ................................. 187
   A.5.5 Adding relationships ............................ 189
   A.5.6 Deleting Objects ................................. 190
   A.5.7 Adding Complex Predicates ................... 191
   A.5.8 Deleting Complex Predicates .................. 192
   A.5.9 Adding complex predicates equal to existing organization ................ 193
   A.5.10 Deleting Complex Predicates larger than existing organization .... 194
A.6 Conclusions ............................................. 194
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Organization Transitions</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>Organization Model Diagram</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>Structural and State Objects</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>Achieves Relation</td>
<td>21</td>
</tr>
<tr>
<td>5</td>
<td>Requires Relation</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>Goal Structures Examples</td>
<td>23</td>
</tr>
<tr>
<td>7</td>
<td>Possesses Relation</td>
<td>26</td>
</tr>
<tr>
<td>8</td>
<td>Capable Relation</td>
<td>27</td>
</tr>
<tr>
<td>9</td>
<td>Capable Example</td>
<td>28</td>
</tr>
<tr>
<td>10</td>
<td>Assigned Relation</td>
<td>29</td>
</tr>
<tr>
<td>11</td>
<td>Organization Transition Concept</td>
<td>29</td>
</tr>
<tr>
<td>12</td>
<td>Initial Computation</td>
<td>31</td>
</tr>
<tr>
<td>13</td>
<td>Initial Computation - Assignment</td>
<td>32</td>
</tr>
<tr>
<td>14</td>
<td>Reorganization</td>
<td>32</td>
</tr>
<tr>
<td>15</td>
<td>Base Temporal Transition</td>
<td>35</td>
</tr>
<tr>
<td>16</td>
<td>Intermediate Transition</td>
<td>35</td>
</tr>
<tr>
<td>17</td>
<td>Simultaneous Arrival</td>
<td>37</td>
</tr>
<tr>
<td>18</td>
<td>Transition Machine</td>
<td>40</td>
</tr>
<tr>
<td>19</td>
<td>Transition Forces</td>
<td>43</td>
</tr>
<tr>
<td>20</td>
<td>Internal and External Organization Properties</td>
<td>44</td>
</tr>
</tbody>
</table>
Segmented Reorganization with decrease to organization size .................. 131
Comparison of Initial Organization Control Times ................................. 132
Comparison of Reorganization Control Times - Increasing Organization Size . . . 133
Comparison of Reorganization Control Times - Decreasing Organization Size . . . 134
Sparse and Dense Organizations .......................................................... 137
Generic Initial Organization - Sparse ................................................... 141
Generic Initial Organization - Dense ..................................................... 141
Generic Reorganization - Constant Growth ............................................ 142
Generic Reorganization - Constant Decline .......................................... 143
Initial Organization Comparison - SPARSE ............................................. 144
Initial Organization Comparison - DENSE ............................................. 145
Initial Organization Comparison - DENSE points 21 to 25 ....................... 146
Initial Organization Comparison with Segmentation - SPARSE .................. 147
Initial Organization Comparison with Segmentation - DENSE ................. 147
Reorganization Comparison - Constant Growth .................................... 148
Reorganization Comparison - Constant Decline .................................... 149
Transition Times - Small Software Organization .................................. 157
Transition Times - Large Software Organization .................................. 159
Sensor Organization .............................................................................. 161
Sensor Organization .............................................................................. 162
Sensor Example .................................................................................. 163
<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>Organization Map</td>
<td>166</td>
</tr>
<tr>
<td>85</td>
<td>Simulation Example - Sector 1</td>
<td>167</td>
</tr>
<tr>
<td>86</td>
<td>Assignments - Sector 1</td>
<td>168</td>
</tr>
<tr>
<td>87</td>
<td>Simulation Example - Sector 6</td>
<td>169</td>
</tr>
<tr>
<td>88</td>
<td>Assignments - Sector 6</td>
<td>170</td>
</tr>
<tr>
<td>89</td>
<td>Transition Times - Sensor Organization</td>
<td>170</td>
</tr>
<tr>
<td>90</td>
<td>Knowledge Transfer</td>
<td>177</td>
</tr>
<tr>
<td>91</td>
<td>Results</td>
<td>182</td>
</tr>
<tr>
<td>92</td>
<td>Initial Organization Time</td>
<td>185</td>
</tr>
<tr>
<td>93</td>
<td>Initial Organization Comparison on Ranges of Predicates</td>
<td>186</td>
</tr>
<tr>
<td>94</td>
<td>Reorganization: Objects Only</td>
<td>187</td>
</tr>
<tr>
<td>95</td>
<td>Reorganization: Objects and Relationships</td>
<td>189</td>
</tr>
<tr>
<td>96</td>
<td>Reorganization: Adding Objects and Relationships</td>
<td>190</td>
</tr>
<tr>
<td>97</td>
<td>Reorganization: Adding and Deleting Objects and Relationships</td>
<td>190</td>
</tr>
<tr>
<td>98</td>
<td>Reorganization: Complex Predicates</td>
<td>191</td>
</tr>
<tr>
<td>99</td>
<td>Reorganization: Complex Predicates - 1000 maximum</td>
<td>192</td>
</tr>
<tr>
<td>100</td>
<td>Reorganization: Complex Predicates - Add and Delete Sets</td>
<td>192</td>
</tr>
<tr>
<td>101</td>
<td>Reorganization: Complex Predicates - Double Size</td>
<td>193</td>
</tr>
<tr>
<td>102</td>
<td>Reorganization: Complex Predicates - Reduce By Half</td>
<td>194</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Complex Predicates</td>
<td>46</td>
</tr>
<tr>
<td>2</td>
<td>Object Predicates</td>
<td>48</td>
</tr>
<tr>
<td>3</td>
<td>Relationship Predicates</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>Generic Initial Organization</td>
<td>78</td>
</tr>
<tr>
<td>5</td>
<td>Generic Initial Organization</td>
<td>79</td>
</tr>
<tr>
<td>6</td>
<td>Generic Reorganization - Add</td>
<td>81</td>
</tr>
<tr>
<td>7</td>
<td>Generic Reorganization - Delete</td>
<td>82</td>
</tr>
<tr>
<td>8</td>
<td>Central Control Initial Organization</td>
<td>90</td>
</tr>
<tr>
<td>9</td>
<td>Central Control Reorganization - Add</td>
<td>93</td>
</tr>
<tr>
<td>10</td>
<td>Central Control Reorganization - Delete</td>
<td>94</td>
</tr>
<tr>
<td>11</td>
<td>Distributed Initial Organization - Complete Knowledge</td>
<td>98</td>
</tr>
<tr>
<td>12</td>
<td>Distributed Control Complete Knowledge Reorganization - Add</td>
<td>101</td>
</tr>
<tr>
<td>13</td>
<td>Distributed Control Complete Knowledge Reorganization - Delete</td>
<td>102</td>
</tr>
<tr>
<td>14</td>
<td>Initial Organization</td>
<td>107</td>
</tr>
<tr>
<td>15</td>
<td>Distributed Control Local Knowledge Reorganization - Add</td>
<td>115</td>
</tr>
<tr>
<td>16</td>
<td>Distributed Control Local Knowledge Reorganization - Delete</td>
<td>116</td>
</tr>
<tr>
<td>17</td>
<td>Initial Organization</td>
<td>125</td>
</tr>
<tr>
<td>18</td>
<td>Segment Reorganization - Add</td>
<td>129</td>
</tr>
<tr>
<td>19</td>
<td>Segment Reorganization - Delete</td>
<td>131</td>
</tr>
<tr>
<td>20</td>
<td>Algorithm Model Summary</td>
<td>132</td>
</tr>
</tbody>
</table>
## List of Algorithms

1. General Initial Organization Algorithm ................................................. 67
2. Establish Goal Root, Internal Goals and Leaf Goals .............................. 69
3. Find the best assignment candidate for a leaf goal .................................. 70
4. Find the best agent candidate to play a role ......................................... 70
5. Role Capability Function (rcf) ............................................................... 71
6. General Reorganization Algorithm ....................................................... 72
7. Install Organization Predicates ............................................................. 73
8. Compute the new organization ............................................................... 73
9. Parse Organization Predicates ............................................................... 74
10. Change Predicates ................................................................................. 74
11. Add Predicates ....................................................................................... 75
12. Lose Predicates ..................................................................................... 76
13. Central Control Initial Organization Algorithm ...................................... 89
14. Compute Initial Central Organization Algorithm .................................... 89
15. Central Control Reorganization Algorithm ............................................ 92
16. Distributed Control Complete Knowledge Initial Organization Algorithm .... 97
17. Compute Initial Distributed Complete Organization Algorithm ................ 98
18. Distributed Control Complete Reorganization Algorithm ....................... 100
19. Distributed Control Initial Local Organization Algorithm .................... 106
20. Compute Initial Distributed Local Organization Algorithm ................... 106
21. Distributed Local Reorganization Algorithm ......................................... 110
22. Propagate Phi ...................................................................................... 112
23. Check Each Agent for objects ............................................................... 113
24. Proposal ............................................................................................... 113
25. Compute Distributed Phi ...................................................................... 113
26. Check Assignments .............................................................................. 114
27.Segmented Initial Organization Algorithm ............................................ 120
1 Introduction

This section introduces a general background on agent organization and transition. Through each section, there is a brief introduction into areas salient to expressing and understanding the research problem of agent transition. This section will provide a scope statement and show how the model will be employed in a realistic example.

Throughout human history people have sought ways to work together and interact. Often people form and interact to accomplish a goal or set of tasks without thinking or defining a formal process to do so. Without organizations, either formal or informal, there is no pattern or planning of interaction. This results in a spectrum of outcomes from poor interaction to a state of chaos or self-inflicted anarchy. These reasons alone provide necessary evidence as to why nature provides the innate capability of life forms to interact in a large spectrum of organizations. Humans have evolved around the concepts of interaction, cooperation and dependency. Although, we often don’t plan organizations specifically, the groups we are involved with tend to take on certain patterns and shapes.

Software agents, by contrast, do not possess the same emotional or physical needs and requirements as humans. Agents can work without emotion or interactive need. Independence of some human traits, makes the use of agents advantageous in the design of organizations. Agents can be developed to have a specific focus eliminating other traits or characteristics that often get in the way with humans.

The question of how an organization arrived at its current state, is often asked. In instantiated organizations, such as corporations or universities, the organization’s strength and success varies by the transitions made over the course of their history. Changes made over the organization’s life determine where the organization will be somewhere in the future. Often this is difficult to predict, otherwise all organizations can achieve success. In reality, this is not the case. To understand why an organization is so successful or on the contrary, fails so miserably, we must look at the interactions and transitions that occur to determine if the organization is making intelligent, rational collective
decisions and then propagating in that direction.

There is a tendency to simplify the analysis of organizations which transition over time. The general thought is that an organization is a collection of actors playing roles that tend to change every once in a while. Organization change is thought to be linear as organizations go from one state to the next in a more or less orderly sequence. The reality is that organizations are vast and complex from the perspective of the number of states they can potentially reach from each state. When considering non-trivial organizations, there are many variables and configurations for each organization instance. The complexity to compute all potential organizations is a daunting task.

Organizations and organization theory have developed over time to capture and describe the behavior people exhibit in using interaction to complete or accomplish goals. Over the course of the last century, academia has sincerely studied and attempted to create models of many types of organizations. The general thought on organizations is that they are not a simplistic formation but instead a complex, environmentally reactive systems with the autonomy and ability to adapt and change over time.

This research focuses on designing and implementing an agent organization model, generic in nature, capable of providing solutions to many domain problems. The model will contain not only structural elements of organization, but the function to transition. The model is targeted at solving problems in the areas of multi-agent teams, robotics, fault tolerant systems, software engineering, internet systems, human-computer interaction and enterprise integration systems.

**Scope Statement** The scope of this dissertation is to provide a fully developed model of an organization. The model will contain structural, state and transition elements. The primary focus is the transitional elements, but the structural and state elements are the building blocks used by transition. The organization model will be developed formally and then extended into a complete software implementation. The formal model will be analyzed from a theoretical perspective. The implemented model will be applied and evaluated empirically. The evaluations will then be compared.
There are a number of organization models that have been developed from either a formal or implementation format, as analyzed by Horling and Lesser [49]. The novelty of this approach lies in its generic nature. While there are examples of research that contain organizational elements, our work consists of a complete model and implementation that can be applied in a number of task scenarios. The model contains multiple algorithms providing the ability to transition in centralized, decentralized and segmented directions. The model also uses the capability to transition using structural and logical formats. This work also recognizes that transition of an organization has two distinct processes, initial organization and re-organization. Each has different requirements and computational characteristics.

1.1 Motivation

The central motivation is the development of the first complete, fully implemented capability-based organization model with full autonomous transitional abilities to organize and reorganize. The model can be applied to any problem domain requiring organization. The minimal prototype has already been applied to successful project outcomes. The complete implementation will have extensive and more universal applicability where capability based organizations can be employed.

There are many advantages in using organization models. Application to many domains, the ability to limit or reduce complexity, reduction of overall necessary agent interaction, assumed intent, cooperation, fault tolerance and recovery are just a few of the advantages. Our model is generic, and purposely designed with the intent of making it applicable to any capability-based organization problem. With the ability to specialize labor, the complexity can be reduced by the nature of simple planning and interaction in relation to roles that will satisfy the system goals. Reducing complexity and specializing roles will reduce overall interaction between agents in comparison with strict peer point-to-point agent implementations. This complexity reduction will allow us to construct larger agent teams, with less computation effort than is currently possible. Because the organization is created around a specific set of goals, the system becomes intentional by nature and cooperative by design. Our organizational model allows the transition to a new
organization if there is a change of any resource or artifact employed with the team, such as goals, roles or agents. Reorganizing also can be triggered by a sub-optimal state occurring or developing in the current organization. The property of self reorganization allows for the implemented systems to be fault tolerant and recoverable.

Because of the generic design, our model is enabled for use in a wide range of applications. The prototype has been implemented in a number of problem domains such as Adaptive Information Systems (AIS), robotics, and agent-oriented software engineering domains. As we have shown here, there are many goal-based robotic applications where our organization model can be applied using an Organization-based MultiAgent System (OMAS) approach.

1.2 Contribution

There are several contributions of this research. The primary contribution is a generic organization model that defines the structural, state and transition elements. The generic nature of our model allows application to any task environment in which capability is used to describe the roles and agents involved. The model will have a full, formal and implemented instance of transition processes, allowing an organization, based on this model, to adapt and survive within reasonable expectations. While this work requires a complete formal model, the organization will be completely implemented into a publishable software package. This will allow others to use the organization and verify any results. To validate the model, organization instances will be developed to demonstrate sensor networks, both simulation and physical, and a software engineering team.

1.3 Thesis Outline

Following this introduction, this work will be fully described in 6 additional chapters. Chapter 2 will expand on the background literature and work in the areas surrounding agent organization and models. Chapter 3 will formally describe the organization model elements, including transition and provide complete examples of organization instances. The central, distributed and segmented tran-
sition algorithms are shown in chapter 4, along with how these algorithms can overcome problems associated with transition processes. Chapter 5 provides evaluation, theoretical analysis, empirical analysis of the model and transition algorithms. Finally, three validation cases are used to show the validity of this model and transition abilities. Chapter 6 concludes the dissertation with contributions, impact and future work. Appendix A shows an alternative knowledge-based approach to the structural approach in chapter 4. The knowledge-based approach was the initial prototype implementation of the transitional model.
2 Literature Review

The study of agent theory, agency and multi-agent systems is a very broad and diverse field. Due to the size of the field, this literature review is not be an exhaustive examination of all agent research. It is limited to the main topic of agent organization and transition. The scope includes the modeling, computational and transitional aspects of organization and the issues encompassing this area. To appropriately frame the literature, material from non-technical areas is used to show motivation for the construction and modeling of organizations. The chapter begins with theoretical background material, of which much agent systems research is based upon, and then moves into agent and multiagent specific literature on modeling and transition of agent organizations.

2.1 Organization Background and Theory

In this section, we examine basic literature in the field of organization theory. Much of this work pre-dates agent theory and the study of artificial and intelligent systems, but is important to understand as a foundation for organization and transition theory. While much of the beginning literature is from non-computing fields, it is of paramount importance to consider the foundations of the area, in order to understand the more complex application of these works.

Throughout the historical study of organizations, they are commonly defined where humans play the roles in organizations. Early documentation of organization by Mooney [68] captures the set of interactions and principles required to construct a formal model. As the study and understanding of organization progressed, organizations began to directly reflect the social values of the period in which they were viewed [77].

In this research, the focus switches to agent organizations, of which humans can be a part of, but also encompass other entities, such as robots, software agents or sensors. Before formalizing an organization there must be a definition of organization. Dale [18] describes an organization as a set of tasks applied to the efforts of two or more people, a system of communication, a means of problem solving, and a means of facilitating decision making. Dale goes on to describe organization
as determining what must be done if a given aim is to be achieved, dividing the necessary activities into segments small enough to be performed by one person and providing a means of coordination, so that there is no wasted effort and members of the organization do not get into each other’s way.

Mooney describes a primitive organization as when two or more people combine their efforts toward the same end, even if the task is short lived. Daft [17] defines an organization, in a more recent context, ”social entities that are goal directed, deliberately structured activity systems with an identifiable boundary”.

To describe the foundations of organizations, the distinction must be drawn between formal and informal organizations. Formal organizations, as described by Blau [5] [7], have a blueprint or a formal design and adhere to this design. Informal organizations sometimes arise from formal organizations, or within them, but don’t necessarily follow the same patterns, rules, or structures. Informal organizations are sometimes referred to as social organizations. This research focuses solely on formal organizations, as informal organizations are intractable to formalize due to their very nature and definition. The formal organizations take on a collective intelligence, such as demonstrated by Far et al. [30].

The multiple definitions of organization routinely share common elements. Common attributes are the number of actors, or agents, which play in an organization, some coordination is required and there should be a common purpose. While these definitions are sufficient, in a general sense, they do not adequately describe the functional composition or relationships of an instantiable organization.

As a task grows beyond the scope and ability of one person to accomplish, additional agents must be employed [4]. When additional agents enter, some structure must exist to define the relationships of the agents acting together. The first task toward defining or implementing a new organization is to define the structure. Because of this requirement, research into organizational theory has yielded numerous possible structures with which to build an organization.

What is most often referred to as structure in an organization is the creation of the bureaucratic model by sociologist Max Weber [8] which is defined by 6 major dimensions:
1. A fixed division of labor among participants.

2. A hierarchy of offices.


4. Separation of personal from official property and rights.

5. Selection of personnel based on technical qualifications (capability or ability).

6. Employment viewed as a career by organizational participants.

Each of these dimensions can be applied in general organization theory in varying capacities. As the definition of an organization consists of more than one agent or person, it only stands to reason that roles must be defined. This allows specialization of labor, which by Smith’s declaration [78] is:

"The greatest improvement in the productive powers of labor, and the greater part of the skill, dexterity and judgment with which it is anywhere directed or applied, seem to have been the effects of the division of labor."

The hierarchy of offices defines the roles in an agent organization and their interrelationships. General rules that govern performance set constraints on the relationships and general structure of the organization. Technical qualifications are at the center of an organization’s definition, as they define the overall capability required or possessed. An organization’s capability defines what it can do.

The definition goes on to describe structural assumptions:

1. Organizations exist primarily to accomplish established goals.

2. For any organization, there is a structure appropriate to the goals, the environment, the technology, and the participants.
3. Structures work most effectively when environmental turbulence and personal preferences of participants are constrained by norms or rationality.

4. Specialization permits higher levels of individual expertise.

5. Coordination and control are accomplished best through the exercise of authority and impersonal rules.

6. Structures can be systematically designed and implemented.

7. Organization problems usually reflect an inappropriate structure that can be resolved through redesign and reorganization.

2.2 Organization Models and Transition

Organization can be defined in numerous terms, but the definition of mechanistic and organic organization structures by Burns and Stalker [6] defines the polarized spectrum of organizations. In reality, most organizations are between these two ends of the spectrum. Complex systems research is defined by emergent systems with no central planning as described by [2] Bar-Yam and [76] Simon.

This research is closer to mechanistic, but also uses elements of organic, or emergent, functions. While structural models are typically defined as mechanistic, the transitional elements of environmental stimuli and self reorganization are typically associated with emergent models.

In this section, agent specific background literature is considered including intentional theory, teamwork, models and transitional theory.

While an organization requires a structure, it must also have dynamic process or transition elements which allow it to change states in the event conditions change. If the goals of the organization or the agents involved change, the organization must potentially be altered. Either condition triggers a potential change in the organization. Organizations must also have the ability to determine if they are executing at an optimal state of efficiency. Optimality is defined as employment of
all organization resources to provide maximal utility. To measure efficiency, there has to be some way to operationalize it. Becker and Neuhaser [3] suggest that,

"in its simplest form, organizational efficiency refers to the way in which the resources of an organization are arranged. In the maximally efficient organization the resources (whether land, labor, capital, goodwill, or any combination thereof) are so arranged that no other method would produce as profitable a return”.

2.2.1 Organization Models

By definition, an agent is any entity using sensors to perceive the environment and effectors to act upon the environment [72]. Using this definition, it is easy to formulate the idea of an agent as any number of entities; humans, animals, software programs and physical devices. Because of the variety and breadth of the various agent domains, it is important to provide a generic structure to support the numerous domains. Agents have become an indispensable part of artificial and intelligent systems theory and have a road map of progression [51].

An agent is potentially capable of independent action. The agent must possess some capability to interpret goals and solve them. A multiagent system consists of a number of interacting agents. In order to successfully interact, the agents of the team need to cooperate, coordinate and negotiate [84][75].

Early work, by Cohen and Levesque [14] [13] defined the Joint Intention theory of teamwork. The core idea is a team member’s key obligation to serve the team, possessing intention to accomplish team goals and placing team goals over individual goals. A joint persistent goal (g) is defined in relation to a team whose members mutually believe that g is false. The responsibility of the team, is to act jointly until all goals are accomplished or its decided that goals cannot be accomplished, or the reason for goals to be accomplished is no longer valid. Cohen and Levesque [15] define joint intentions, which still holds. It is suggested by Tambe [80] [79] that teamwork over an extended period of time is more than simply a union of coordinated simultaneous activity. Teamwork must
involve a common goal. This idea is further expanded by Grosz and Kraus [38] using collaboration of agents formalizing a model of joint planning. To plan as a team, it is inherent to have joint intention. The idea of joint collaboration is extended by Dunin-Keplicz and Verbrugge [29] into collective attitudes.

The idea of competition within a single team of agents is compared to the premise of cooperation, as described in the analysis by Mazurkiewicz [67]. In this research, we assume the basic premise that agents always act in a cooperative manner, with a global set of goals and beliefs. Cooperation as defined by Hall and Watkins [39] is the joint working of two or more persons and is as old as human society. They go on to say that,

"Nothing has contributed more to the economic and and social well-being of the human race than the practice of cooperation”.

In terms of a multiagent system, cooperation is defined as the "coordination among non-antagonistic agents”, by Weiss [83] and similarly by Ferber [32]. To cooperate, an agent must coordinate with others on a centrally agreed set of goals or tasks. Kirn and Gasser [54] and later Omicini and Ricci [70] propose the integration of coordination with organization.

The assumption is that organizations form to achieve a defined set of goals. With this being stated, it can be assumed that an organization is an intentional system. An intentional system requires all agents to work toward accomplishment of the same set of goals in a cooperative manner, such that their intent is the same. Early work on teamwork provides a foundation for more complex societies and organizations.

2.2.2 Structure

An agent organization contains many of the same elements as a human or general organization. There must be a structural model with elements such as goals, roles and agents. The formalities must be captured to define the structure and nature of an organization. All mechanical elements of an organization must be formalized including organizational rules, structures and patterns before
the interactions and transitions can be exhibited, as shown by Zambonelli et al. [85].

Agent organizations are computational, where numerical relationships are established to choose the optimal linkages between goal, role and agent elements. An example of a linkage is the capability of some agent to play a role. There may be several agents that can effectively play the role, but one agent is more capable than all others. Establishing the maximal relationship between all elements results in a computationally optimal organization. Computational organizations have long been studied by Carley [10][12] in theoretical form. While there was not a viable implementation, the base work provides an understanding of the inherent complexity in computing an organization. Organization computation has also been approached using elements of the organization, such as capabilities, which are used to define relationships [47].

The elements of an agent organization structure are goals, roles, agents, laws, constraints, capabilities, and relationships. These elements and the physical relationships that occur between them are the mechanical foundations of organization theoretic models. A great deal of research has been attempted in this area, with widely ranging views on how the elements and relationships are modeled.

Tambe [80] describes the STEAM system introducing the idea of implemented joint intentions and teamwork, using production rules.

Glaser et al. [36][37] proposed a structural model for the purpose of organizing autonomous robots, using multiagent systems. A basic model with concepts of beliefs and shared goals is proposed by Filipe [33]. The model includes epistemic, deontic and axiological components, but does not provide a transitional function. Dikenelli and Erdur [28] propose an ontology-based adaptive model supporting action in a complex environment. Although each of these models captures some critical components of organization, each lacks the definition of a transition element.

Dastani, Dignum and Dignum propose an organization model in terms of an agent society where the model is divided into two layers. The first layer, facilitation, consists of roles designed to govern social behavior of agents. The operational layer translates the overall objectives into intended society (organization) actions [19] [27]. MOISE is a very interesting model of role-based
multiagent organization system by Hannoun et al. [40]. Although the system was targeted for a specific problem domain, it modeled roles and interactions comparable to a complex organization. It was extended by Hubner, in 2002, to the MOISE+ system [45] adding deontic features and integrating the structure and function of an organization.

Matson and DeLoach propose a generic organizational model [55], [26] with structural, state and transition elements.

Roles are typically viewed as the laborers used to satisfy goals. As defined by Dastani, Dignum and Dignum, organizational roles are the abstract representation of a policy, service or function [20]. They go on to state that,

"Role descriptions in the organizational model identify activities and services necessary to achieve society goals and enable us to abstract from the individuals that eventually will enact the role”.

Once a structural model is developed, then the dynamic elements that describe how an organization forms and reforms, over its useful life, must also be described.

2.2.3 Transitional Theory

The essence of organization change is captured by Jonker, Schut and Treur [53] where they describe it by processes of successive organizational change that lead to an evolving system that is capable of adapting to its environment. The general requirement of an organization system is environmental adaptation. While that is stated in an abstract sense, it can be applied in any domain.

An organization that can compute one time will not be able to adapt to unforeseen events, given a complex task environment. Therefore, it is critical that an organization possesses the ability to adapt as described by Carley [11]. Organizations are often intentional systems [9] with a set of goals to accomplish. The goals must be accomplished even in the face of adversity or environmental complexity [71].
While there is a great deal of discussion and exploration in the area of agent organization, defining and implementing agent organization instances using a formal transition theory has not been so well developed. Glaser and Morigot [37] provide a useful, but incomplete, idea of reorganization, although many basic agent structure principles are documented.

Although there have been many formalizations of organization structures, the first true definition of a formal agent transition model is from Matson and DeLoach [61] and is extended in this document. The initial elements of organizational algorithms were explored by Zhong [86] in terms of the time needed to reorganize a small organization. Further work of modeling organizational change, specific to multiagent system, is described by Hoogendoorn generally [41] and also applied to more specific cases [42] or task domains [43].

The idea of self-organization occurs throughout the natural systems. Multi-agent systems are very promising technologies for their potential ability to dynamically self-reorganize as their conditions and environments change [52]. Nature has been cited as the inspiration for self-organizing systems in numerous implemented systems or simulations. One example of this is by Shen [73]. He uses hormone models to simulate adaptation in complex robotic systems. While there is not a formal organization structure, it does provides interesting ideas on biologically inspired change.

Ishida et al [50] employs organizations to structure activities using knowledge introducing the concept of agent primitives. The novelty is the ability to compose and decompose an agent structure, within the organization.

Rapid change in complex environments requires a system to adapt to survive or continue the stated mission. Turner and Turner [81], [82] show that principles of autonomous reorganization can be accomplished in the niche environment of oceanic sampling networks. While this example provides some valuable architectural insights, it is limited to a specific domain.

Societies, or organizations, are normally controlled by a set of codified policies. Feltovich [31] demonstrates how higher level animal and human societies must adapt using and conforming to a set of policies.
### 2.2.4 Previous Publications

We have already had success in conducting and publishing research related to this work. An early work in the progression of the concepts of organization and cooperation was in [22] and [24] by examining how Agent Oriented Software Engineering can be applied to cooperative robotics. During the same period, the evaluation of how to build a baseline for robotic agents was examined [56]. The first iteration of the introductory organization model [55] was published at the NASA Workshop Robosphere in 2002. The model was applied to the Adaptive Information Systems (AIS) problem domain in [23] and [58] and the area of cooperative robotics in [60]. To start expanding the depth of the model, the first effort at formalizing some constraints and concepts of capability was published in [57]. The initial transition theory is proposed at the IEEE KIMAS ’05 Conference [61]. The formalization of the properties that drive transition was published in 2006 [59].

There are issues in the creation of autonomous agent organizations. One problem, with reorganization cycles, are the temporal effects [62]. Another issue is the state explosion of organizations [63].

### 2.3 Summary

This study is inspired by numerous fields, in addition to computer science, such as psychology, philosophy and anthropology. To consider and formalize a complete model, all aspects of organization research must be considered, both technical and non-technical.
3 Organization Model

The main focus of this research is the transition of organizations. To fully understand the transition of organizations, we must first explore the structural elements and composition of the organization model. Instantiation of an organization structure results in a specific organization state. The mechanism to move from state to state is the process of transition.

Before we can completely understand the intricacies of an organization, we must clarify that an organization does not exist within a vacuum. By definition, an organization exists for some purpose. This purpose may be the accomplishment of a goal or, more commonly, a set of goals. The solution of the goals exist within the scope of some defined task environment, whereas the agent view of task environment is defined by Decker and Lesser [21].

This model is a variant of the core OMACS model [26]. This variant, the Transitional OMACS model, focuses on the ability to interact with its environment through the use of transitional properties, which are stimuli from the environment, that will cause change to an organization. In other words, this model is focused on the ability to transition and adapt within a task environment. Fig. 1 shows the organization as it transitions through a number of states.

![Figure 1: Organization Transitions](image)

In this section, all three elements of the organization model will be described. This model and the transition theory contained within forms the foundation of this research effort. An example will be used at the end this section to demonstrate and clarify each feature of the organization model and transition function.
3.1 Organization Model

To implement an organization consisting of autonomous, heterogeneous agents, we developed an organizational model, which defines and captures the required elements of an adaptable organization. While most people have an intuitive idea of what an organization is, there is no single, agreed upon standard definition. However, in most organizational research, organizations have typically been understood as including agents playing roles within a structure in order to satisfy a given set of goals. Our organizational model \( O \) encompasses structural and state models and a transition function.

\[
O = (O_{structure}, O_{state}, O_{transition})
\]

The organization \( O \) is composed of the structural model \( O_{structure} \), the state model \( O_{state} \) and the set of algorithms that enable transition from one state of the organization to the next \( O_{transition} \). Each of these elements will be formally defined in the following sections.

3.2 Structure

The basic intuition of organization structure is defined by all elements and relationships between elements in the organization as shown in Fig. 2. More formally described, the structural model includes a set of goals \( G \) that the team is attempting to achieve, a set of roles \( R \) that must be played to attain those goals, a set of capabilities \( C \) required to play those roles, and a set of rules or laws \( L \) that constrain the organization. The model also contains static relations between roles and goals (achieves), roles and capabilities (requires), goals and subgoals (subgoals) and a unary relation for conjunctivity between subgoals of a goal (conjunctive). Formally, we model the organization structure as a tuple:

\[
O_{structure} = < G, R, L, C, ACH, REQ, SUB, CON >
\]
where:

$G$ is a set of Goals

$R$ is a set of Roles

$L$ is a set of Laws

$C$ is a set of Capabilities

$ACH \leftarrow \{achieves(r, g) \rightarrow [0..1] \mid r \in R, g \in G\}$

$REQ \leftarrow \{requires(r, c) \rightarrow \text{Boolean} \mid r \in R, c \in C\}$

$SUB \leftarrow \{\text{subgoal}(g_1, g_2) \rightarrow \text{Boolean} \mid g_1, g_2 \in G\}$

$CON \leftarrow \{\text{conjunctive}(g) \rightarrow \text{Boolean} \mid g \in G\}$

### 3.2.1 Structural Objects

There are four sets of structural and state objects in the organization model. The four object sets are Goals $G$, Roles $R$, Laws $L$ and Capabilities $C$. Goals, Roles and Capabilities are defined in this
section, while Agents will be defined in the next section. Fig. 3 shows the graphical description for elements of each of the structural and state objects used to define the organization elements and transition.

Figure 3: Structural and State Objects

**Definition: Goal**  A goal, \( g \in G \), is an object which partially or wholly describes what the organization intends to accomplish. The goal defines the specific purpose of the organization. All goals \( g \in G \) may be either abstract or specific but are entities that often must be decomposed to have deliverable outputs and used to identify the critical aspects of system requirements. Goals have their own structure, where a goal can be a subgoal of another higher level goal. A higher level, more abstract goal, is composed of a number of subgoals. The subgoals have either a conjunctive or disjunctive relationship. The organization goal set includes the abstract and discrete goal definitions, goal-subgoal decomposition, and the relationship between the goals and their subgoals, which are either conjunctive or disjunctive. This abstraction can be performed by removing detailed information when specifying goals. Goals can be abstract or discrete. An example of an abstract goal is "solve world hunger". An example of a discrete goal is "deliver 2 bags of rice to the soup kitchen".

**Definition: Role**  A role, \( r \in R \), describes an entity which performs some function within the system, analogous to roles played by actors in a play or by members of a typical company structure. In general, roles may be played by zero, one, or many agents simultaneously while agents may also play many roles at the same time. Each role requires a set of capabilities, which are inherent to particular agents.
Within OMACS, each organization contains a set of roles \( R \) that it can use to achieve its goals. A role defines a position within an organization whose behavior is expected to achieve a particular goal or set of goals. Roles are analogous to roles played by actors in a play or by members of a typical corporate structure. A typical corporation has roles such as president, vice-president, or mail clerk. Each role has specific responsibilities, rights and relationships defined in order to help the corporation perform various functions towards achieving its overall goal. Specific people (agents) are assigned to fill those roles and carry out the roles responsibilities using the rights and relationships defined for that role.

OMACS assumes that each role implies some minimal expected behavior. For instance, it would be assumed that someone playing the mail clerk role in a company would pick up mail from the mail room and eventually deliver that mail to its addressee. This minimal behavior defines the functionality associated with the role. Although an understanding of this behavior is critical to the design and operation of the actual system, it is not critical to the definition of the organization of the system and is not specified further in OMACS.

**Definition: Law**  
Organizational laws, \( l \in L \), are used to constrain the assignment of agents to roles and goals within the organization. Generic rules such as “an agent may only play one role at a time” or “agents may only work on a single goal at a time” are common. However, rules are often application specific, such as requiring particular agents to play specific roles. We introduce the notion of laws into the organization, which operationalize norms, sanctions/rewards, and their relationship. Laws should also conform to organizational values. Laws are constraints on actions and thus the law \((a, s)\) prohibits the action \( a \) from being taken when state \( s \) holds [74]

**Definition: Capability**  
Agents are defined by the individual capabilities, \( c \in C \), they possess. The agent’s capabilities define the roles they can play in meeting a team goal. The capabilities represent the level of ability or intelligence built into the agent. For example, a robot’s sensory capability is based on the sensors built into the robot and the algorithms that allow the robot to perceive the environment. Capability may be individual or part of a composition.
So far, we have used the term capability generically. A capability’s existence is based on the collective sense in which it is viewed. To specify this, we further define capabilities in relation to agents and roles that exist within a self-reorganizing multiagent team. As described above, an agent possesses specific capabilities while roles require particular capabilities, each with specific scores.

3.2.2 Structural Relations

The structural model relations define sets of relationships between the structural objects. The structural relations are: Achieves $ACH$, Requires $REQ$, Subgoal $SUB$ and Conjunctive $CON$.

**Definition: Achieves** Achieves is modeled as a function to capture the relative ability of a particular role to satisfy a given goal. Goal satisfaction is dependent upon the ability of a role to complete the requirements of a goal. A role that can be used to satisfy a particular goal is said to achieve that goal. In Fig. 4, we show the achieves relation, $achieves(r_1, g_1) = 0.5 \mid r_1 \in R, g_1 \in G$.

![Figure 4: Achieves Relation](image)

**Requires** Requires is a boolean relation that specifies a role must have some numerically definable capability. If this capability is not present, then the relation is false.

Likewise, the capability set of a role, $C_r$, is the set of capabilities required to play that specific role. The capability set formally describes what capabilities are required for agents potentially to enact and play the role [20]. All non-trivial roles must have at least one capability in order to accomplish some task or goal.
\[ C_r(r) = \{ c \mid \text{requires}(r, c) \} \]  

Figure 5: Requires Relation

In Fig. 5, the requires relation is shown, \( \text{requires}(r_1, c_1) = 0.7 \mid r_1 \in R, c_1 \in C \).

**Definition: Subgoal** Defines a boolean relationship between two goals where one goal is a direct subgoal of the other. If the relationship does not hold, the result is false. Goal and subgoal structures can be conjunctive or disjunctive. The subgoal relation is shown in Fig. 6 (a), where \( G = \{ g_1, g_2, g_3, g_4, g_5, g_6 \} \) and \( g_1 \) has subgoals \( g_2, g_3 \) and \( g_4 \), formally expressed as \( \text{subgoal}(g_1, g_2), \text{subgoal}(g_1, g_3), \text{subgoal}(g_1, g_4) \), respectively. Fig. 6 (c) displays \( g_1 \) with subgoals \( g_2, g_3 \) and \( g_4 \) and consequently \( g_4 \) with subgoals \( g_5 \) and \( g_6 \).

When subgoal relationships exist within a goal tree, there are internal goals \( G_{\text{int}} \), leaf goals \( G_{\text{leaf}} \) and a root goal \( g_{\text{root}} \) each having a specific definition. The definition of \( G \) in relation to subgoal relations is:

\[ G = \{ g_{\text{root}}, G_{\text{int}}, G_{\text{leaf}} \} \]

where \( g_{\text{root}} \) is an individual goal and \( G_{\text{int}} \) and \( G_{\text{leaf}} \) are goal sets. In terms of organizations, we can also describe the goal hierarchy as a set of abstract goals \( G_{\text{abstract}} = \{ g_{\text{root}}, g_{\text{int}} \} \), which cannot be directly solved and leaf goals \( G_{\text{leaf}} \), which are directly solved.
**Definition: Root Goal**  A root goal, $g_{\text{root}}$, is the most abstract, top-level goal in the goal-subgoal hierarchy. There is a single root goal within each organization. In Fig. 6, $g_{\text{root}}$ is $g_1$.

**Definition: Internal Goal**  Internal goals $G_{\text{int}}$ are descendants of the root goal and ancestors of leaf. They exist in the root path from a leaf goal to the root goal. They can only be accomplished by the accomplishment of all of their descendant goal nodes. In Fig. 6, $G_{\text{int}} = \{g_4\}$.

**Definition: Leaf Goal**  The set of leaf goals, $G_{\text{leaf}}$, represents all goals which contain no subgoal relationships. Leaf goals are the only goals which are capable to maintain an achieves relationship with a role. In Fig. 6, $G_{\text{leaf}} = \{g_2, g_3, g_5, g_6\}$.

Subgoal relationships cannot be cyclic. For example, goal 1 and goal 2, where there is a subgoal relationship.
relationship, expressed by \( \text{subgoal}(g_1, g_2) \), is strictly singular. If \( g_2 \) is a subgoal of \( g_1 \), then \( g_1 \) cannot be a subgoal of \( g_2 \).

**Definition: Conjunctive** The conjunctive relation \( \text{conjunctive}(r) \rightarrow \text{Boolean} \) specifies whether a goal has a conjunctive or disjunctive relationship with all of its subgoals. A conjunctive relation is defined by true whereas a disjunctive relation is defined by false. The conjunctive relation is shown in Fig. 6 (b), where \( g_1 \) has a conjunctive relationship over all of its subgoals, \( g_2, g_3 \) and \( g_4 \). In Fig 6 (a), the relationship is disjunctive, as indicated by the absence of the arc covering the relationship between \( g_1 \) and each of its subgoals. The subgoal relationships in Fig. 6 (c) are all disjunctive, whereas in (d) all subgoal relationships are conjunctive.

### 3.3 State

The second element of the Organization Model is state. The *Organizational State* \( (O_{\text{state}}) \) is an instance of the organizational structure at a point in time with additional relationships and elements added. As the organization structure is a template, the state is an instance of the model. In an instance of an organization state, each of the elements will be bound to a set of values that represent the organization attributes. An organization will possess at least one goal, one role to accomplish the goal, and one agent to play the role where the agent will possess capabilities required by the role. Not every organization state element is required to be populated by an instance variable for creation of a valid organization. The constraints and laws of an organization will govern the requirements of a specific state.

The organizational state model defines an instance of a team’s organization and includes a set of agents \( (A) \) and the actual relationships between the agents and the various structural model objects.

\[
O_{\text{state}} = \langle A, \text{POS}, \text{CAP}, \text{ASN} \rangle
\]  

where:

- \( A \) is a set of *Agents*
3.3.1 State Objects

**Definition: Agent**  Agents coordinate through the organization via conversations and act proactively and cooperatively to accomplish global and individual goals. The model includes a set of heterogeneous agents \( A \) in each organization. As described by Russell and Norvig, an agent is an entity that perceives and can perform actions upon its environment \([72]\), which includes humans as well as artificial (hardware or software) entities. For our purposes, we define agents as computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so, realize a set of goals. Thus, we assume that agents exhibit the attributes of autonomy, reactivity, pro-activity, and social ability. Autonomy is the ability of agents to control their actions and internal state. Reactivity is an agent’s ability to perceive its environment and respond to changes in it, whereas pro-activeness ensures that agents do not simply react to their environment, but that they are able to take the initiative to achieving their goals. Finally, social ability allows agents to interact with other agents, and possibly humans, either directly via communication or indirectly through the environment. Within the organization, agents must have the ability to communicate with each other, accept assignments to play roles that match their capabilities, and work to achieve their assigned goals.

3.3.2 State Relations

**Definition: Possesses**  An agent that possesses the required capabilities for a particular role is said to be capable of playing that role. Since not all agents are created equal, possesses is modeled as a real valued function, where 0 would represent absolutely no capability to play a role while a 1 indicates an excellent capability. In addition, since agent capabilities may degrade over time, this
value may actually change during team operation.

Capacity compositions exist anytime a role requires more than a single capability or an agent possesses more than a single capability.

The capability set of an agent, $C_a$, ranges from no capability to an extensive set of the capabilities that the agent intrinsically possesses. Typically, although not always, even a simple agent has multiple capabilities.

$$C_a(a) = \{ c | \text{possesses}(a,c) > 0 \}$$  \hspace{1cm} (6)

![Figure 7: Possesses Relation](image)

In Fig. 7, the possesses relation is shown, $\text{possesses}(a_1, c_1) = 0.8 | a_1 \in A, c_1 \in C$

**Definition: Capable** The capable function defines the ability of an agent to play a particular role and is computed based on the capabilities required to play that role.

An agent is capable of playing a role if $C_r(r) \subseteq C_a(a)$. How well agent $a$ can play role $r$ is determined by the role capability function ($rcf$) that is part of each role definition. The $rcf$ is part of the role and defines a role-specific computation based on the capabilities possessed by an agent. If an agent does not possess one of the required capabilities, then the agent has no capacity to play that role and $r.rcf(a) = 0$. Thus, the capability score of an agent playing a particular role is defined by:

$$\text{capable}(a,r) = r.rcf(a)$$  \hspace{1cm} (7)

A role $rcf$ determines the ability of an agent to play that role; it is user defined and computed
in terms of the agents capabilities. Having the required capabilities is not necessarily sufficient to determine whether an agent can actually play the role or decide which agent can best play the role. Some capabilities may be more important to the role than others. To capture this on a role-by-role basis, the designer must define a role specific rcf, which computes a value in the range of 0 to 1. The role capability function allows the role designer to specify how specific capabilities affect the ability of an agent to play that role. OMACS uses the notation r.rcf(a) to denote the application of the role capability function for role r on agent a.

\[
rcf(r, a) = \begin{cases} 
  \prod_{c \in \text{requires}(r, c)} \text{possesses}(a, c) = 0 & 0 \\
  \text{else} & \sum_{c \in \text{requires}(r, c)} \text{possesses}(a, c) / |\text{requires}(r, c)|
\end{cases}
\]  

(8)

Figure 8: Capable Relation

In Fig. 8, the capable relation is shown, \{\text{capable}(a_1, r_1) = 0.75 \mid a_1 \in A, r_1 \in R\}

Fig. 9, shows an example of capable relationships. In part (a), we see a relationship where an agent, \(a_1\), is capable of playing a role \(r_1\). Role \(r_1\) requires two capabilities, \(c_1\) and \(c_2\). Since \(a_1\) possesses both \(c_1\) and \(c_2\), a capable relationship is computed. In this example, the \(rcf\) for \(r_1\) is computed:

\[
rcf(r_1, a_1) = r_1.rcf(a_1) = \frac{\sum_{c \in \text{requires}(r_1, c)} \text{possesses}(a_1, c)}{|\text{requires}(r_1, c)|} = \frac{0.2 + 0.7}{2} = \frac{0.9}{2} = 0.45
\]

In part (b) of Fig. 9, the example shows agent \(a_1\) not capable of playing role \(r_1\). In this case, role \(r_1\) requires two capabilities, \(c_1\) and \(c_2\), but \(a_1\) only possesses capability \(c_2\). Therefore \(a_1\) is not capable of playing \(r_1\).
Definition: Assigned  During the organization process, a specific agent is selected to play a particular role in order to satisfy a specific goal. This relationship is captured by the assigned function, which includes a real valued score that captures how well an agent, playing a specific role, can satisfy a given goal.

During the organization process, a specific agent is selected to play a particular role in order to satisfy a specific goal. This relationship is captured by the assigned function, which includes a real valued score that captures how well an agent, playing a specific role, can satisfy a given goal.

Having specific requirements does not imply that an agent can achieve a role or is the best to achieve the role.

\[
\text{assigned}(a, r, g) = \text{capable}(a, r) \times \text{achieves}(g, r)
\]  

In Fig. 10, the assigned relation is shown, \( \text{assigned}(a_1, r_1, g_1) = 0.67 \mid a_1 \in A, r_1 \in R, g_1 \in G \).
3.4 Transition

The Organization Transition Function ($O_{transition}$) defines the ability of the organization to reorganize from an instance state to the next instance state over the organization life span. From the initial organization, through its termination, the organization may transition its state model numerous times as shown in Fig. 11.

The organization transition function defines how the organization may transition from one organizational state to another over the lifetime of the organization. Since the team members (agents) as well as their individual capabilities may change over time, this function cannot be predefined, but must be computed based on the current state and the goals that are still being pursued. In our present research with purely autonomous teams, we have only considered reorganization that
involves the state of the organization. However, we have defined two distinct types of reorganization: state reorganization, which only allows the modification of the organization state, and structure reorganization, which allows modification of the organization structure (and may require state reorganization to keep the organization consistent).

The initial step in organizing a multiagent team is to use the existing goals to establish the required organizational roles. At the same time, the set of agents making up the team must assess their individual and collective capabilities to determine whether they can fulfill the required roles [MD03]. If the required roles can be filled, then the capabilities exist to satisfy the organization goals and the team assigns the necessary roles to agents (effectively defining the state of the team’s organization). Once the assignments are made, the team may initiate action to satisfy the team information production goals.

An assumption with any organization is that the most capable, available agents will be assigned to roles where the overall organization utility is maximized. In human organizations, this is not always the case due to non-computational aspects of the decision process. In an agent system, we will use this as a standard assumption throughout this research. The criteria for an optimal organization is the maximal computed set of relationships between all employed organization objects and relationships.

Transition is the process in which an organization propagates to a new state. There are two forms of transition, initial organization and reorganization. The difference in these processes are the preconditions, inputs and outputs.

\[
\text{transition} = \begin{cases} 
\text{state} = 0 & \text{initial organization} \\
\text{state} > 0 & \text{reorganization}
\end{cases}
\]  

(10)

Definition: Initial Organization A new or initial organization creates an organized team of agents from a set of available agents. The initial step is to consider a pre-defined set of goals and roles of the proposed team. At the same time, the group must assess its individual and collective
capabilities, based on the available agents. There is a determination made at this point to evaluate if the set of available agents can satisfy the set of goals through playing the set of roles. This capability evaluation will result in one of three outcomes; satisfaction, relaxation or abandonment. Once tasks, roles and capabilities are considered, the assignment of player to role is completed. At this point, the new organization state initiates action to satisfy the team goal(s). The free agents exist in a state where they are available to join an organization if they possess the proper capabilities.

The process of initial organization begins at an initial state $O_0$ where there exists no pre-existing organization and an organizational structure results $O_1$. The resulting organization structure is the first organization instance.

\[
\text{initial organization}(O) \equiv \text{transition}(O_0) \Rightarrow O_1
\]  

(11)

Initial organization and reorganization differ in the computational steps required to complete each process. Conceptually, the first step to initial organization is the formation of individual sets of $G, R, C$ and $A$ into a structural map as shown in Fig. 12.

Figure 12: Initial Computation

Once the structural map has been assembled, the computation of the optimal assignments for the organization can be computed. This is the second step of initial organization. This is shown in Fig. 13.
**Definition: Reorganization**  The process of reorganization begins with an already existing organization $O_n$ and transitions to a new state $O_{n+1}$, representing a different and unique organization instance, as shown in Fig. 14.

$$ reorganization(O) \equiv transition(O_n) \Rightarrow O_{n+1} $$ (12)
3.4.1 Transition Results

There are different effects of transitions. Some transitions will cause no structural change to the organization and externally appear as nothing has changed, while other, more drastic changes, will cause the organization to fail or go to an end state. In this section, the differences between a minimal, moderate and a catastrophic transition are described.

Minimal Effect Transition is a transition in which the structural elements are not added or subtracted from the organization graph. An example of minimal effect transition is the value change of an achieves relationship.

Moderate Effect Transition is a transition where structural or state elements are physically changed and the organization graph is impacted by an addition or loss of objects or relationships. An example of moderate effect change is the loss of an agent from an organization.

Catastrophic Effect Transition occurs when the transition property outcome disables the organization from continuation and sends the organization to an end state. An example is the loss of an agent required to play a role to satisfy a goal that is critical to the organization.

Organization Process Optimization A major assumption in organization theory is that an organization should operate at an optimal state given what resources it may employ. The process of organization or reorganization does not always require the entire set of agents and roles to be considered. If there is a non-trivial organization with 1000 agents and 500 roles, the time to reorganize will become impractical if the entire set of agents and roles has to be considered and reorganized. Minimalization is needed to only reorganize the agent and role combinations necessary to provide a viable organization.
3.4.2 Theoretical Problems and Issues

There are a number of complex theoretical issues that present problems when considering multiagent organizations. As with human organizations, typically the larger the organization, the more intrinsic complexity. To fully consider transition, the analysis must consider potential and known theoretical issues such as temporal complexity, which deals with the arrival, timing and computation of new organizations.

Temporal Issues To understand the temporal effects of organization transition, the basic model of transition over time must be well specified[62]. In Fig. 15, the base case of transition is shown where the organization transitions from \( state_n \) to \( state_{n+1} \). The transition takes an elapsed total time, \( t_{\text{transition}} \) from \( t_0 \) to \( t_1 \), so we can define transition time as:

\[
t_{\text{transition}} : t_0 \rightarrow t_1
\]  

(13)

where the transition function of:

\[
\delta(O_a, \phi_i, s_0) \vdash s_{n+1}
\]  

(14)

where the general transition function representation states that \( organization_a \) transitions from \( state_n \) to \( state_{n+1} \) where \( \phi_i \) is the organization transition property that triggers the transition. This transition occurs over the discrete period from \( t_0 \) to \( t_1 \). The time of transition, \( t_{\text{transition}} \), begins at the time the transition property \( \phi_i \) is realized and stops at the time the organization has completed the processing necessary for a new organization. Each organization transition has a discrete and calculable time.

Using the temporal definition of agent organization transition, there are some specific problems that can occur. In this paper, two specific temporal problems are defined. The first and second problems defined are \( \text{intermediate transition} \) and \( \text{simultaneous transition} \), respectively.
Intermediate Transition is described as the arrival of transition properties that will trigger the organization to transition, when a previous transition is already underway. It is possible that a transition property will arrive during the time that a reorganization is already occurring. This problem is described by Fig. 16. This will create a situation in which a reorganization tries to begin again, while the organization is already propagating through a transition cycle.

This problem also has an extended case where there is a sequence of transition properties arriving during the intermediate stages of a transition cycle. Each new arrival will compound the
effects and eventually will cause a circular spiral which is unrecoverable.

*Simultaneous Transition* is defined as when two transition properties arrive at the same time. A choice must be made as to which to select. This is partly an organization decision problem, but more so a temporal problem. As the propagation time for transition to a new state takes time, having two or more arrive at the same time will multiply the effects of increasing the time to complete the transition, thereby delaying the reactivity of the organization to return to accomplishing the set of goals for which it is responsible. This is described in Fig. 17, where transition properties $\phi_a$ and $\phi_b$ arrive simultaneously and, in addition to the organization having to decide which one to start first, there is also the problem of the overall transition taking

$$t_{total} = t_\delta(O_a, \phi_a, s_n) - s_{n+1} + t_\delta(O_b, \phi_a, s_0) - s_{n+1}$$

if both transition properties are utilized and acted upon. The extended case of this problem is when there is $n$ number of organization transition properties that arrive simultaneously. In that case, the same problem occurs but is magnified by the number of transition property arrivals.

$$t_{total} = \sum_{i=0}^{m} t_\delta(O_a, \phi_i, s_n) - s_{n+1} : m = |\phi|$$

where $m$ is the number of organization properties simultaneously arriving.

### 3.4.3 Transition Foundations

Formally, the abstract notion of transition represents two distinct processes, *initial organization* and *reorganization*.

$$O_{state(0)} \rightarrow O_{state(1)}$$

The reorganization process follows the same basic steps as the organization process; however,
it differs in the point of initiation. Reorganization is initiated by a trigger event, such as sensor loss, during the execution of an already existing organization. When such an event occurs, the team must determine if it still has the capabilities to satisfy team information production goals or whether it must reorganize to do so.

\[ O_{state(n)} \rightarrow O_{state(n+1)} \]  \hspace{1cm} (18)

**Structural Transition**  The structure changes, meaning one or more of the structural elements or relationships are altered.

**State Transition**  The structure does not change, just the instances of agents playing roles in the organization change.

### 3.5 Formal Transition

Organization and reorganization are transition processes. To understand these processes, we must first formalize transition. Transition can occur for a number of reasons such as loss of agent,
evolving capabilities possessed and required, sub-optimal performance or goal changes.

We can reason about transition mechanics being similar to a finite state automaton [48]. An automaton has a set of finite start states, a transition function and a set of reachable states. An organization does not use an input symbol from an alphabet to transition. It will use a triggering organization property, \( \phi \in \Phi \), which is further described in the next section and can be a transition, such as loss of agent or change in goals. The general form of our theoretic organization model approach is expressed by:

\[
O_{\text{transition}} = (O, \Phi, \delta, s_n, S_{\text{optimal}}, S_{\text{possible}}, S_{\text{final}}) \tag{19}
\]

where \( O \) is the organization over which the transition will occur, \( \Phi \) is the set of properties that can trigger a transition of the organization, \( \delta \) is the transition function, \( s_n \) is the set of relative states of the organization, \( S_{\text{optimal}} \) is the set of optimal states that result from transition and \( S_{\text{possible}} \) are states that are possible to reach from the current state. \( S_{\text{final}} \) is a set of organization states where all goals are satisfied, or the last goal is satisfied, or it is determined that not all goals can be satisfied. Even though the outcomes are different, each final state draws a conclusion to the organization’s set of transitions. Because an organization can only exist as a single entity or instance, the current state \( s_n \) is always a unique value.

An organization must transition for a specific and valid reason, otherwise serious side effects can occur, such as instantaneous transition looping, where an organization toggles between instances without progression. It relies on properties of the organization to be dynamically changed for triggering transition. The properties are defined by the problem domain and environment and are thus dependent on these factors.

When an property is consumed by an organization, a frontier of new possible states are generated. The most optimal of the new is selected and becomes the new organization state. The basic transition is defined as a product of the \( O, \Phi \) and \( S \) resulting in a set of reachable organization states:
\[ \delta : O \times \Phi \times S \rightarrow S \] (20)

So the transition function will be of the form:

\[ \delta(O, \phi, s_n) \rightarrow S' \] (21)

where transition function \( \delta \) takes the organization \( O \), a specific transition property \( \phi \), and a state of the organization \( s_n \) and can transition to a set of new states \( S' \) where \( S_{optimal} \subseteq S_{possible} \) and \( S_{optimal} \subseteq S' \). \( S_{final} \subseteq S_{possible} \) for both finite and infinite transition organizations, with the added constraint that \(| S_{final} | \geq 1 \) for finite transitions and \(| S_{final} | = 0 \) for infinite transitions.

Where a finite state automata uses a string of symbols to transition, as normally used to validate languages, our transition function takes as input a string of transition properties \( \phi \). This string of properties is not predetermined, but will be generated dynamically as the organization interacts in its environment as represented by \( \{ \phi_0, \phi_1 \ldots \phi_n \mid \phi \in \Phi, n \in \mathbb{N} \} \).

Anytime any element of the organization changes, a new organization state will be instantiated. Even the smallest change in the organization state changes the structural integrity of the organization state, and therefore, a transition must occur.

When a property change initiates a reorganization, the organization may transition to a new state \( s_{n+1} \) or it can also result in being in the same state \( (s_n) \) where, even though the property changed, the values and relationships of the organization instance did not change.

The graphical view of an example organization transition machine is shown in Fig. 18. Stating the same transitions, using the transition function, is defined by:

\[ O_{transition} = \{ O, \{ A, B, C, D \}, \delta, \{ s_0, s_1, s_2, s_3, s_4, s_{final} \}, S_{optimal}, S_{possible}, S_{final} \} \]
and the following organization state transitions:

\[
\begin{align*}
\delta(O_1, \phi_A, s_0) &\vdash s_1 \\
\delta(O_1, \phi_B, s_0) &\vdash s_2 \\
\delta(O_1, \phi_C, s_1) &\vdash s_2 \\
\delta(O_1, \phi_C, s_1) &\vdash s_4 \\
\delta(O_1, \phi_A, s_2) &\vdash s_4 \\
\delta(O_1, \phi_D, s_2) &\vdash s_{final} \\
\delta(O_1, \phi_A, s_3) &\vdash s_{final} \\
\delta(O_1, \phi_B, s_4) &\vdash s_3 \\
\delta(O_1, \phi_A, s_4) &\vdash s_{final}
\end{align*}
\]
Due to the model non-deterministic definition, we can use an alternate shorthand notation and express transitions by:

\[ \delta(O_1, \phi_C, s_1) \vdash \{ s_2, s_4 \} \]

### 3.5.1 Transition Scope

Some organizations have a definite and discrete number of goals to complete before the useful life of the organization comes to an end. Other organizations have an undefined number of goals to satisfy. Examples of these are a project team and a company, respectively. To differentiate, we define the terms of *finite* and *infinite* organizations.

**Definition: Finite Organization** A finite organization exists when the number of goals the organization is tasked to solve is discrete, where \(| G | = n, n \in N\) and \(N\) is strictly finite. An example of a finite organization is a project, where there is a definite start, a definite end and a discrete number of tasks or goals within the start and end.

**Definition: Infinite Organization** An infinite organization is when the number of goals are indefinite or potentially infinite \(| G | = \infty\). An example of an infinite organization is a company. Since a company does not plan for its own demise, it will continue indefinitely with no set end.

Based on the definitions of finite and infinite organizations, there must exist finite and infinite transitions for an organization. While even a finite organization can theoretically transition an infinite number of times, we reason that a finite organization will transition a finite number of times and an infinite organization will transition an infinite amount of times.

**Definition: Finite Transition** To formally express the definition of a finite organization transition, the condition must hold for the set of transitions to eventually result in a final state being reached. The transition:
\[
\delta(O_i, \phi, S) \vdash^* S_{final}
\]

states that there is a set of transitions that will lead from some initial machine state to a \(S_{final}\) or final state. This will also indicate that, if designed correctly, a finite transition organization possibly is a successful organization where all goals have been achieved.

In our example, a finite transition scope indicates that we have some idea of the length of our organization usefulness. We may only track the phenomena for a certain length of time or cap the number of potential transitions.

**Definition: Infinite Transition** An infinite organization transition can be expressed in a similar manner to the finite. By definition, an infinite organization never reaches a final state, so the expression will result in \(S_\infty\), instead of \(S_{final}\).

\[
\delta(O_i, \phi, S) \vdash^* S_\infty
\]

This indicates that an infinite organization will not reach a final state and will continue on transitioning indefinitely. For this reason, \(| S_{final} | = 0\) will be an enforced constraint for any infinite transition organization.

### 3.5.2 Organization Properties

An organization property \(\Phi\) is somewhat of an abstract theoretical term. It is abstract to capture the generic nature of what it can define. In general terms, an organization will need a set of properties \(\Phi\), for example, capabilities or agents, which by their existence can be the reason for a transition. A major element of defining transition will be the definition of these properties such that individual properties \(\phi\) can be identified as transition triggers. Any individual property \(\phi\) in \(\Phi\) is eligible to act as a reorganization trigger. Some examples of \(\phi\) include a change in the real
value of a capability, the loss of overall capability or agent function, loss of a agent, the reentry of an agent, or the addition of a new agent.

Changes in organization structure and participants will drive transition activities. Transition properties can be triggered internally or externally, as shown in Fig. 19. The general transition properties can be split into properties that are external and those that are internal.

Internal Force represents any stimulus that is part of the organization which causes the organization to transition. An example is an agent who loses capability, thereby not being able to fulfill the requirements of some role. External Force is any stimulus that is not part of the organization which causes the organization to transition. An example is the entry of a new agent, more capable to play a role than an agent already playing that role.

A transition property may result from the relationship between an organization and the environment or simply because of activity within the organization. Fig. 20 shows the relationship between the environment and organization, which allows a external organization property. An example of an external organization property is the addition of a new agent to the organization. An internal
organization property is generated from within the organization and consumed by the organization. An example of an internal organization property is the solution of a goal within the organization.

Organization transition properties must be developed in a formal description. First, we describe what transition properties are in an abstract sense. In this section, we will describe abstract organization properties and specific organization properties in an intuitive manner. Then, we will proceed to translate the intuition into a formal set of properties.

The basic assumption is the system will act rationally as it will always act in its own best interest. Its collective best interest will be defined as always maintaining the highest capability score.

The intuition of $\phi$ is some intrinsic or extrinsic force which will trigger a potential transition in the organization such that $\phi \in \Phi$ where $\phi$ represents an individual transition property.

There are a number of task specific transition properties that exist in any domain problem. What we intend to capture here are general transition properties that can be applied to any organization. These general properties can be instantiated to fit specific examples, as will be shown in the next section. The general transition properties are:
1. Loss of an agent participating in the organization

2. An agent loses capability required to play some role

3. A new agent becomes available

4. Capability of an agent increases

5. Capability of an agent decreases

6. A goal is removed

7. A goal is added

8. A goal is relaxed (changed)

9. Change in goals to roles achieves relationship

10. Change in role to capability requires relationship

### 3.5.3 Transition Predicates

Using transition predicates allows a way of formalizing the individual transition properties of a unique organization.

A key assertion in formalizing transition predicates is that there is no requirement or need for temporal specifications to model the predicates. As each predicate represents a reason to consider reorganization, there is no need to apply a larger temporal language. While there are temporal problems associated with transition, the granularity of properties and the associated predicates remove the need to attach temporal specifications.

Transition predicates can be abstracted in several forms:

\[ \Phi = \{ \phi_1 \ldots \phi_n \} \] (24)
In general, $\Phi$ can be expressed as a set of standard, abstract predicates:

$$
\Phi = \{\phi_{\text{lose}}, \phi_{\text{add}}, \phi_{\text{change}}\} \tag{25}
$$

where $\phi_{\text{lose}}$ is the abstract property dealing with loss, such as losing an agent from the organization or an agent losing capability to play a role. The add property $\phi_{\text{add}}$ describes the action when an object or relationship is added to the organization. For example, with $\phi_{\text{add} \text{agent}}$, a new agent is added to the organization. The change property $\phi_{\text{change}}$ can either be an increase or decrease and further specializes the change predicate:

$$
\phi_{\text{change}} = \{\phi_{\text{decrease}}, \phi_{\text{increase}}\} \tag{26}
$$

**Definition: Primitive Predicates** can be used to formalize single properties. The *primitive predicates* exhibit polymorphic behavior as each can be applied to different organization elements to capture different properties. If there is a loss of an agent participating in the organization, it can be formalized as the predicate $\phi_{\text{lose} \text{agent}}(a)$. An agent $a$ losing some capability can be captured as $\phi_{\text{lose} \text{capability}}(c, a)$.

**Definition: Complex Predicates** are the combination of primitive predicates. Some predicates will encompass others, but in some cases two properties can be successfully combined to form a single property of transition. The complex predicates will be logically constructed using primitive predicates and the common *and* ($\land$) binary relation. Examples of complex predicates are shown in Table 1.

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_{\text{add} \text{agent}}(a) \land \phi_{\text{add} \text{agent}}(b)$</td>
<td>Add new agents $a$ and $b$</td>
</tr>
<tr>
<td>$\phi_{\text{add} \text{goal}}(g) \land \phi_{\text{add} \text{goal}}(h) \land \phi_{\text{add} \text{subgoal}}(g, h)$</td>
<td>Add goals $g, h$ and a subgoal relation</td>
</tr>
<tr>
<td>$\phi_{\text{add} \text{role}}(r) \land \phi_{\text{lose} \text{role}}(s)$</td>
<td>Add role $r$ and delete role $s$</td>
</tr>
</tbody>
</table>

There are constraints on complex predicates. An *add* predicate can either contain just *objects*...
or objects and relationships. A delete predicate can be just a relationship or object and relationship. A change predicate can be either objects or relationships or a combination.

In the case that an agent exits an organization, it can be reasoned that all capability of that agent will also exit. Combining the two previous predicates of losing an agent and losing a capability by an agent are redundant, in respect to the capability predicate $\phi_{\text{lose agent}}(a) \land \phi_{\text{lose capability}}(c, a)$, as long as the capabilities are not possessed by another agent or required by a role. Deleting an object will involve deleting any relationships which are dependent on the object.

In another situation, an organization may lose two agents simultaneously. If agents $a$ and $b$ both leave, we can capture that by $\phi_{\text{lose agent}}(a) \land \phi_{\text{lose agent}}(b)$, where one primitive predicate does not contain the other. Complex predicates are unlimited in their scope. They may be used to create a set of relationships and objects greater than the size of the existing organization.

As there are primitive and complex predicates, predicates can be further defined as either object or relationship predicates. Object predicates represent objects of the organization such as goals, roles, capabilities or agents. Relationship predicates represent the link between two objects such as achieves, possesses or requires.

**Definition: Object Predicates** are defined as predicates where the property represents an object of the organization, such as an agent being added or a goal being deleted. An example of an object predicate is a goal addition, $\phi_{\text{add goal}}(g)$. This is a single predicate only involving an organization object. Examples of object predicates are shown in Table 2.

**Definition: Relationship Predicates** are defined as properties where a relationship between two objects is added, lost or altered. Relationship predicates can be primitive, as long as the objects in which they bind already exist in the organization. Object predicates may be complex as the object must collaborate with a relationship to connect to the organization. Object and relationship predicates will typically be combined in complex predicates. Relationship predicates such as when $g_i$ is a subgoal of $g_j$ is $\exists_{g_i, g_j} \phi_{\text{add subgoal}}(g_i, g_j)$ where the relationship can only exist if both $g_i$ and
### Table 2: Object Predicates

<table>
<thead>
<tr>
<th>φ</th>
<th>Object</th>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>agent</td>
<td>φ_{addagent}(a)</td>
<td>Add a new agent a</td>
</tr>
<tr>
<td></td>
<td>goal</td>
<td>φ_{addgoal}(g)</td>
<td>Add a new goal g</td>
</tr>
<tr>
<td></td>
<td>role</td>
<td>φ_{adddrole}(r)</td>
<td>Add a new role r</td>
</tr>
<tr>
<td></td>
<td>capability</td>
<td>φ_{addcapability}(c)</td>
<td>Add a new capability c</td>
</tr>
<tr>
<td></td>
<td>law</td>
<td>φ_{addlaw}(a, max(r, 2))</td>
<td>Add law l where an agent can play only 2 roles</td>
</tr>
<tr>
<td>lose</td>
<td>agent</td>
<td>φ_{loseagent}(a)</td>
<td>Delete an existing agent a</td>
</tr>
<tr>
<td></td>
<td>goal</td>
<td>φ_{losegoal}(g)</td>
<td>Delete an existing goal g</td>
</tr>
<tr>
<td></td>
<td>role</td>
<td>φ_{loserole}(r)</td>
<td>Delete an existing role r</td>
</tr>
<tr>
<td></td>
<td>capability</td>
<td>φ_{losecapability}(c)</td>
<td>Delete an existing capability c</td>
</tr>
<tr>
<td></td>
<td>law</td>
<td>φ_{loselaw}(a, max(r, 2))</td>
<td>Delete law l</td>
</tr>
<tr>
<td>change</td>
<td>agent</td>
<td>φ_{increasecapability}(c, a)</td>
<td>Increase capability of an agent</td>
</tr>
<tr>
<td></td>
<td>goal</td>
<td>φ_{decreasecapability}(c, a)</td>
<td>Decrease capability of an agent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>φ_{changegoal}(g)</td>
<td>A goal is changed</td>
</tr>
</tbody>
</table>

$g_j$ exist prior. Another relationship predicate $g$ is achieved by $r$ if $\exists_{g,r} \phi_{addachieves}(g, r)$ where the relationship can only exist if both $g$ and $r$ exist prior. Examples of relationship predicates are shown in Table 3. The general constraint for a relationship predicate is stated by $\exists_{x,y} \phi_{relationship}(x, y)$

### 3.6 Organization Segmentation

Segmentation of an organization is splitting the global organization structure into segments. Segmentation requires that each division does not share objects or relationships with other segments.

There are several motivations to segment an organization. By definition, an organization segment does not share resources with other segments. A segment is smaller than the global organization, therefore it has less elements and can compute a transition faster. Computation is local to a segment and other segments are not considered. The reduction of individual segment size reduces the computationally intensive requirements of transition. In concept, a segmented organization is simply an organization of organizations, bound by a common abstract goal $g_{root}$.

Segments can transition in parallel as shown in Fig. 21. This is conceptually thread-safe, as each segment computes individually, as they are disjointed. Each segment can compute individually,
Table 3: Relationship Predicates

<table>
<thead>
<tr>
<th>φ</th>
<th>Relationship</th>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>achieves</td>
<td>( \phi_{\text{add achieves}}(r, g) )</td>
<td>Add a new achieves relationship between goal ( g ) and role ( r )</td>
</tr>
<tr>
<td></td>
<td>requires</td>
<td>( \phi_{\text{add requires}}(r, c) )</td>
<td>Add a new requires relationship between role ( r ) and capability ( c )</td>
</tr>
<tr>
<td></td>
<td>possesses</td>
<td>( \phi_{\text{add possesses}}(a, c) )</td>
<td>Add a new possesses relationship between agent ( a ) and capability ( c )</td>
</tr>
<tr>
<td>lose</td>
<td>achieves</td>
<td>( \phi_{\text{lose achieves}}(r, g) )</td>
<td>Delete the achieves relationship between goal ( g ) and role ( r )</td>
</tr>
<tr>
<td></td>
<td>requires</td>
<td>( \phi_{\text{lose requires}}(r, c) )</td>
<td>Delete the requires relationship between role ( r ) and capability ( c )</td>
</tr>
<tr>
<td></td>
<td>possesses</td>
<td>( \phi_{\text{lose possesses}}(a, c) )</td>
<td>Delete the possesses relationship between agent ( a ) and capability ( c )</td>
</tr>
<tr>
<td>change</td>
<td>achieves</td>
<td>( \phi_{\text{change achieves}}(r, g) )</td>
<td>Change the achieves relationship between goal ( g ) and role ( r )</td>
</tr>
<tr>
<td></td>
<td>possesses</td>
<td>( \phi_{\text{change possesses}}(a, c) )</td>
<td>Change the possesses relationship between agent ( a ) and capability ( c )</td>
</tr>
</tbody>
</table>

Figure 21: Segmentation Concept

handling numerous parallel transition properties \( \phi \) simultaneously. This solves a temporal problem of organization transition properties arriving simultaneously.

**Definition: Segmentation**  The organization \( O \) can be segmented into \( n \) segments, \( O_1 \) to \( O_n \), such that \( O = \bigcup_{i=1}^{n} O_i \).

\( O_i \subset O \) and \( O_j \subset O \). The organization \( O \) is composed of \( G, R, L, C \) and \( A \). Each object must be addressed in terms of segmentation. Segmentation is defined over the objects of the organization. By definition, the relationships of the organization, can only exist if the objects exist. Relationships,
however, must be constrained by segmentation so as not to connect two individual segments.

**Definition: Goal Segmentation** Because an organization, by definition, exists to solve a common goal or set of goals, goals cannot be completely segmented. So, \( \exists g_{root} \mid g_{root} \in G, g_{root} \neq \emptyset \), meaning that \( g_{root} \) must be the top level goal in the organization. In this definition, the goal hierarchy of goals and subgoals relationships can be described as \( \exists G \mid g_{root} \in G, G_{int} \subset G, G_{leaf} \subset G \). The set of goals is defined by a root, a set of internal nodes and a set of leaf nodes, where \( G = \{g_{root}, G_{int}, G_{leaf}\} \).

An organization segment defines rules from another perspective. The root goal \( g_{root} \) must exist, but instead of the previously defined internal and leaf node sets, segmentation splits the goals by segment.

Because the organization is bound by the goals it is to satisfy, the goals are segmented into unique goal spaces, \( G_1 \) through \( G_n \). Because \( G_1 \) through \( G_n \) are sets of goals and subgoals of \( G \), \( G = \{g_{root}, \bigcup_{i=1}^{n} G_i\} \). Additionally, \( \forall G_i \in G \mid \bigcup_{i=1}^{n} G_i, G_i \not\subseteq G_{i+1}, G_{i+1} \not\subseteq G_n \).

Fig. 22 shows an example of goal segmentation. The root goal, \( g_{root} \), in this case is \( g_1 \). There are 3 distinct segments \( G_1, G_2, G_3 \) where \( G = \{g_1, G_1, G_2, G_3\} \). Additionally, \( G_1 = \{g_2\}, G_2 = \{g_3, g_5, g_6\}, G_3 = \{g_4, g_7, g_8\} \). Each goal segment is disjoint from each of the others.

![Goal Segmentation Diagram](image-url)
**Definition: Role Segmentation**  A requirement of an organization is the existence of at least one role \( r \) where \( r \in R \). Roles are completely distinct within segmentation, meaning that a role cannot be shared across more than a single segment. This requirement stipulates that each segment will require at least one distinct role.

Within \( R \) there are \( n \) subsets \( R_1 \) to \( R_n \), such that \( R = \bigcup_{i=1}^{n} R_i \). \( R_i \neq \emptyset \) and \( R_j \neq \emptyset \), such that \( R_i \neq \emptyset \) and \( R_j \neq \emptyset \). So \( \forall g \exists r \mid r \in R_i, g \in G_i, achieves(g, r) \) and \( \forall g \exists r \mid r \in R_j, g \in G_j, achieves(g, r) \). Furthermore, \( R_j \not\subseteq R_j \) and \( R_j \not\supseteq R_j \). \( R_j \) and \( R_j \) are disjoint and neither is a subset of the other.

**Definition: Capability Segmentation**  It is trivial to reason about capability segmentation, as capabilities only exist if there is an agent who possesses them or an role that requires them. It is satisfactory to define segmentation based on roles and agents, instead of capabilities. Each segment must contain at least one capability.

**Agent Segmentation**  As agents can enter and exit an organization, segmentation allows agents to move from one segment \( O_i \) to another segment \( O_j \) freely. In fact, the basis of segmentation allows neighboring segments to exchange agents to best fit the needs and requirements.

**Relationship Segmentation**  Relationships cannot cross a segment line as defined by the objects, as shown by Fig. 23. For example, there cannot be an \( achieves(g_2, r_2) \), as the relationship crosses a segment.

**Optimization Utilizing Segmentation**  Segmentation defines how an organization can be dissected into smaller organizations to reduce the need for computation during transition. Arriving at optimization using segmentation has two perspectives. The first perspective is when to initially segment the organization. When an initial organization is computed, the decision will have to made of whether or not a larger organization should be segmented if possible.
The overall goal is to develop the best strategy and create the most optimal segmentation scheme for $O$. 

Figure 23: 3 Segment Organization
3.7 Conference Organization Example

In this section, we generate a simple, but not trivial, organization and then show a progression through a number of transitions. Each transition is modeled with a specific predicate. After each transition the impact will be shown, with changes reflected from the previous predicate. To give the organization model life, we use an example of a small organization organizing a workshop at a conference.

3.7.1 Organization Description

To demonstrate how we apply the formalizations of transition, properties and predicates, we define an organization and then apply the predicates to the organization. As each transition predicate is applied, a new organization state is formed which is different from its predecessor. The example organization is shown in Fig. 24, the Organization Graph.

![Organization Graph](image)

Figure 24: Organization Graph

Once the organization structure is defined, the initial organization step is shown. The initial organization transitions from $O_0 \Rightarrow O_1$ and is a single transition process. Beyond state $O_1$ all transitions will be reorganizations where the organization transitions from some state to the next state, $O_n \Rightarrow O_{n+1}$. 
3.7.2 Organization Structure

The goals, roles and capabilities of the organization, \( O_{example} \), must be defined first. There are no defined related relationships. The organization in charge of managing the conference workshop will have the following elements:

The goals of the organization are: \( G = \{ g_0, g_1, g_2, g_3, g_{2.1}, g_{2.2}, g_{3.1}, g_{3.2}, g_{3.3} \} \)

Where:

- \( g_0 \) is the goal to organize a conference workshop.
- \( g_1 \) is the goal of completing a workshop proposal.
- \( g_2 \) is the goal of promoting the workshop
- \( g_{2.1} \) is the goal of successfully advertising the workshop
- \( g_{2.2} \) is the goal of publishing the workshop details on the web
- \( g_3 \) is the goal of preparing the program
- \( g_{3.1} \) is the goal evaluating papers
- \( g_{3.2} \) is the goal of paper selection
- \( g_{3.3} \) is the goal of organizing the final workshop program

Only the subgoal relationships, shown in Fig. 25, where the result is true are listed. The subgoal relationships are:

\[
\text{subgoal}(g_0, g_1) \rightarrow \text{true} \\
\text{subgoal}(g_0, g_2) \rightarrow \text{true} \\
\text{subgoal}(g_0, g_3) \rightarrow \text{true} \\
\text{subgoal}(g_2, g_{2.1}) \rightarrow \text{true} \\
\text{subgoal}(g_2, g_{2.2}) \rightarrow \text{true} \\
\text{subgoal}(g_3, g_{3.1}) \rightarrow \text{true} \\
\text{subgoal}(g_3, g_{3.2}) \rightarrow \text{true} \\
\text{subgoal}(g_3, g_{3.3}) \rightarrow \text{true}
\]
There are conjunctive goal relationships, shown in Fig. 26, in this organization. Only relationships where the result is true are listed. The conjunctive goals are:

\[
\text{conjunctive}(g_0) \rightarrow \text{true} \quad \text{conjunctive}(g_2) \rightarrow \text{true} \quad \text{conjunctive}(g_3) \rightarrow \text{true}
\]

The roles of the organization, shown in Fig. 27 are:

\[ R = \{ r_1, r_2, r_3, r_4 \} \]

- \( r_1 \) is the workshop chairperson.
- \( r_2 \) is on the workshop committee.
- \( r_3 \) is on the workshop committee.
- \( r_4 \) is on the workshop committee.

The achieves relationships, shown in Fig. 28 defined are:

\[
\begin{align*}
\text{achieves}(r_1, g_1) & \rightarrow .8 \\
\text{achieves}(r_2, g_2) & \rightarrow .1 \\
\text{achieves}(r_2, g_{2.1}) & \rightarrow .2 \\
\text{achieves}(r_3, g_3) & \rightarrow .6 \\
\text{achieves}(r_3, g_{3.1}) & \rightarrow .8 \\
\text{achieves}(r_4, g_{3.3}) & \rightarrow .5
\end{align*}
\]
The capabilities added to the organization, shown in Fig. 29, are:

\[ C = \{c_1, c_2, c_3, c_4, c_5, c_6\} \]

- \( c_1 \) requires management skills.
- \( c_2 \) is the skill of communication.
- \( c_3 \) is organizing the workshop message.
- \( c_4 \) is knowledge to evaluate research.
- \( c_5 \) is discerning the best papers.
- \( c_6 \) is ability to work with the general workshop chair.

There are a number of requires relationships shown in Fig. 30. Only the relationships resulting in true are listed. The requires relationships are:

\[
\text{requires}(r_1, c_1) \rightarrow \text{true} \quad \text{requires}(r_1, c_2) \rightarrow \text{true} \\
\text{requires}(r_2, c_2) \rightarrow \text{true} \quad \text{requires}(r_2, c_3) \rightarrow \text{true} \\
\text{requires}(r_3, c_3) \rightarrow \text{true} \quad \text{requires}(r_3, c_4) \rightarrow \text{true}
\]
3.7.3 Organization State

The section is based on the previously defined organization state.

When an initial organization is computed, the agents must be added to the structural elements such as goals, roles and capabilities. The agents, shown in Fig. 31, and the possesses relationships, shown in Fig. 32, for our example organization are:

\[ A = \{ a_1, a_2, a_3, a_4 \} \]
The \textit{possesses}, \textit{capable} and \textit{assigned} relations will be stated. The \textit{possesses} are:

\begin{align*}
\text{possesses}(a_1, c_1) &\rightarrow .8 & \text{possesses}(a_1, c_2) &\rightarrow .7 \\
\text{possesses}(a_1, c_3) &\rightarrow .9 & \text{possesses}(a_2, c_2) &\rightarrow .8 \\
\text{possesses}(a_2, c_3) &\rightarrow .3 & \text{possesses}(a_2, c_4) &\rightarrow .3 \\
\text{possesses}(a_3, c_3) &\rightarrow .9 & \text{possesses}(a_3, c_4) &\rightarrow .8 \\
\text{possesses}(a_3, c_5) &\rightarrow .7 & \text{possesses}(a_4, c_4) &\rightarrow .2 \\
\text{possesses}(a_4, c_5) &\rightarrow .5 & \text{possesses}(a_4, c_6) &\rightarrow .8
\end{align*}

For an \textit{agent} to play a \textit{role}, they must be capable. Capability is considered based on any capable
score greater than 0. The capable function score is generated by the *role capability function* (ref).

The *capable* relations are:

\[
\begin{align*}
\text{capable}(a_1, r_1) & \rightarrow .45 \\
\text{capable}(a_2, r_2) & \rightarrow .55 \\
\text{capable}(a_3, r_3) & \rightarrow .8 \\
\text{capable}(a_4, r_4) & \rightarrow .65 
\end{align*}
\]

The final step in the computation of an organization is the assignment of an *agent* to play a *role* that is charged with achieving a *goal*. The computational aspects of this process are described in a previous work [25]. We have to compute the assignment scores for each possible combination of *capable* and *achieves* relationships, respectively.

To calculate an assigned score, we use the equation:

\[\text{assigned}(a, r, g).score = \text{capable}(a, r) \times \text{achieves}(g, r)\]

Calculating an example assignment, we use the assignment of \(a_1\) to \(r_1\) for the achievement of \(g_1\). This calculation is:

\[
\text{assigned}(a_1, r_1, g_1).score = \text{capable}(a_1, r_1) \times \text{achieves}(g_1, r_1)
\]

\[\text{assigned}(a_1, r_1, g_1).score = .45 \times .8\]

### 3.7.4 Computing an Optimal State

To compute an *optimum* organization, we must find the highest possible organization score that obeys the defined *laws* of the organization. To compute the organization score all valid assignments are summed, \(\sum \text{assigned}(a, r, g)\) where \(a \in A, r \in R, g \in G\). If there are more than one possible configuration of assignments, all possible assignment combinations are considered. The configuration with the highest organization score is the *optimal* state. Each control algorithm will contain procedures to reduce the number of configurations calculated. An example is:

\[\text{assigned}(a_1, r_1, g_1) = .45 \times .8 = .36\]
assigned(\(a_2, r_2, g_{2.1}\)) = .55 \times .1 = .055 \\
assigned(\(a_2, r_2, g_{2.2}\)) = .55 \times .2 = .11 \\
assigned(\(a_3, r_3, g_{3.1}\)) = .8 \times .6 = .48 \\
assigned(\(a_3, r_3, g_{3.2}\)) = .8 \times .8 = .64 \\
assigned(\(a_4, r_4, g_{3.3}\)) = .65 \times .5 = .325 \\

Once each assigned score is individually calculated, the total organization score can be summed. It is a basic summation of all assigned relations. The total organization score is 1.97. For this example, we will take this organization score as optimal.

### 3.7.5 Transition - Initial Organization

The assigned relations capturing the assignments computed for the initial organization state, shown in Fig. 33, are:

\[
\begin{align*}
\text{assigned}(a_1, r_1, g_1) & \quad \text{assigned}(a_2, r_2, g_{2.1}) \\
\text{assigned}(a_2, r_2, g_{2.2}) & \quad \text{assigned}(a_3, r_3, g_{3.1}) \\
\text{assigned}(a_3, r_3, g_{3.2}) & \quad \text{assigned}(a_4, r_4, g_{3.3})
\end{align*}
\]

These assignments are assigned by a single agent \(a_1\) using the central control algorithm. In this case, \(a_1\) is responsible for computing the organization and making assignments. The responsibility of managing the transition is placed on \(a_1\). In future states, \(a_1\) will retain the responsibility to manage the transition computation and assignment.

### 3.7.6 Transition - Reorganizations

From the definitions, it is apparent that there are not a great number of different combinations of agents to play the different roles. Based on the various relationships, \(a_1\) is the only agent able to play the role of chairman \(r_1\) which is the only role able to achieve the goal of completing a workshop proposal, \(g_1\). This organization is constructed as a simple model to show the effects of predicate based transitions.
There are different effects of transitions. Some transitions will cause no structural change to the organization and externally appear as nothing has changed, while other, more drastic changes, will cause the organization to fail or go to an end state. In this section, we will show the effects of a *minimal*, then a *moderate* and finally, a *catastrophic* transition.

The first transition is a *minimal effect* transition. It is shown by the transition equation:

\[ \delta(O_{\text{example}}, \phi_{\text{change}}(c_1, a_1), s_0) \Rightarrow s_1 \]

The effects of this transition reflect no change to the organization structure. The relationship from the agent \( a_1 \) to the capability \( c_1 \) does not force the organization to change structure or assignment. The organization must be computed, because there is the potential that a change in capability may change assignments of agents to roles. This is an internal effect, as the capability is changed by the agent involved, without external intervention.

The second transition is a *moderate effect* transition. It is shown by the transition equation:

\[ \delta(O_{\text{example}}, \phi_{\text{lose}}(g_1), s_1) \Rightarrow s_2 \]

In this transition, with results shown in Fig. 34, the organization deletes one of the goals, \( g_1 \). In real terms the goal, \( g_1 \) is no longer required and drops from the organization. Since \( g_0 \),
the overall workshop organization does not conjunctively require $g_1$, the proposal, a transition can be made. The agent $a_1$ is still attached to the workshop chairperson role, although neither is currently utilized in the organization. The agent $a_1$ is still available to manage the transition assignments. This changes the structure of the organization but has no effects of making the organization non-functional to complete the other goals. So while it changes the structure, it does not have a catastrophic effect. This is an external effect as the goals are not developed within the organization.

![Figure 34: Organization at $s_2$](image)

The third transition, as shown in Fig. 35, is a catastrophic effect transition. It is represented by the transition expression:

$$
\delta(O_{\text{example}}, \phi_{\text{lose}}(a_1) \land \phi_{\text{lose}}(a_2), s_2) \Rightarrow s_3
$$

This transition removes agents $a_1$ and $a_2$ from the structure. The chairperson and one of the program committee members are no longer available for the organization. While the loss of $a_1$, the chairperson, has no real state effect, the loss of $a_2$, a program committee member, has dramatic consequences. The loss of $a_1$ will not allow the organization to transition, as $a_1$ is the only agent capable of making assignments. This is an internal effect and is catastrophic to the organization. The organization cannot survive with the loss of both agents $a_1$ and $a_2$. Even though $g_1$ is already deleted, which was dependent on $r_1$, played by $a_1$, the $g_2$ structure is the problem. The $r_2$ satisfies
both $g_{2.1}$ and $g_{2.2}$. With both $a_1$ and $a_2$ gone, the capability no longer exists to play $r_2$ and therefore no way to achieve either $g_{2.1}$ and $g_{2.2}$, which are conjunctive under $g_2$. Because $g_2$ cannot be achieved, the overall goal $g_0$ cannot be achieved, which renders the organization ineffective. By definition, if a required element of an organization is missing, the organization is not viable. For example, if all goals are deleted, the organization has no reason to exist. On another front, if there are no agents to play roles to satisfy goals, the organization is not viable.

Figure 35: Organization at $s_3$
3.8 Summary

This chapter describes the basic organization model in which the transitional approach is applied. While the model is basic in approach, it captures the generic elements of many organizations. The generic approach allows this model to be utilized in numerous application task domains. The next chapter will focus on the transitional algorithms which extend the model for purposes of initial organization and reorganization.
4 Structural Transition Algorithms

This chapter begins with an introduction on basic transition algorithms. It then shows the basic transition algorithms of initial organization and reorganization. The orientation of the agents involved in the organization are covered. The centralized, distributed and segmented algorithms are then described. Finally, the chapter concludes with a summary.

4.1 Introduction

Transition is the process in which an organization propagates to a new state. There are two forms of transition, initial organization \( \text{init. organization}(O) \equiv \text{transition}(O_0) \Rightarrow O_1 \) and reorganization, \( \text{reorganization}(O) \equiv \text{transition}(O_n) \Rightarrow O_{n+1} \). There are differences in these processes in terms of preconditions, inputs and outputs, but the most important difference is in terms of computational requirements.

\[
\text{transition} = \begin{cases} 
\text{state} = 0 & \text{initial organization} \\
\text{state} > 0 & \text{reorganization} 
\end{cases}
\]

(27)

The organization model and transition processes developed in two different forms, structural and knowledge-based. The structural form takes the perspective that an organization consists of a set of components, such as agents, roles and goals, and a set of relationships between the components. The knowledge-based perspective specifies an organization as a set of facts, rules and assertions. The structural model computes transitions in a algorithmic sense, whereas the logical model computes using rules to infer the new state. This chapter describes structural transition while the next chapter covers knowledge-based transition. Both approaches represent the same idea, but use different techniques to compute organizations.

There are three structural transition algorithms that are formally defined and implemented as a core element of this research dissertation. The three algorithms define centralized, distributed and a segmented control strategies of transition. Each of the three strategies has a initial organization.
as well as a reorganization algorithm. All three variations are based upon the general algorithms as discussed in chapter 3.

The three algorithms each arrive at an organization via a different process and computational expense. With the central control algorithms, one of the agents decides who is to play the role of each sensor agent and then assign accordingly. With the distributed control algorithm, each of the five agents negotiate on its own behalf to determine who is most capable to be assigned a particular role. The segmented model allows the agents to form sub-teams of 1 to n agents to organize in smaller teams and, in turn, organize the smaller organization. The segmented model allows only a part of the organization to reorganize if required by a transition property.

Within the agent research community, there is often a disconnect between formalisms that are expressed logically versus those which are expressed algorithmically. While this research requires that algorithms contain the previously defined formalisms, there is a mix of logical and set-based practicalities. The further implementations of all algorithms are completed in both procedural and logically-oriented formats.

Then, each of the three structural algorithms are defined in this section. This section begins by describing generic transition algorithms, for initial organization and reorganization, to frame the more specific control methods. The discussion of how the agents interact is presented and then each of the algorithms is described.

### 4.2 Transition Computation

There are distinct differences in the two general algorithms. The general initial organization algorithm is described in algorithm 1 and the general initial reorganization algorithm is shown in algorithm 6. The initial organization algorithms requires the existence of goals G, roles R, capabilities C and agents A. It does not specifically require a transition property φ, as it is assumed. The reorganization requires an organization already existing and a transition property φ relevant to that organization.
4.2.1 Computing Initial Organization


Procedures are used as they can take inputs but do not return a single value. Within this organization context, all algorithm elements which do not take a formal parameter or return a value are considered to be visible to the organization as a whole.

**Algorithm 1 General Initial Organization Algorithm**

<table>
<thead>
<tr>
<th>Require:</th>
<th>(\exists G,R,L,C,A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require:</td>
<td>(\exists ACH,REQ,\text{SUB,CON,POS,CAP,ASN})</td>
</tr>
<tr>
<td>Require:</td>
<td>(</td>
</tr>
<tr>
<td>Require:</td>
<td>(</td>
</tr>
<tr>
<td>Require:</td>
<td>(</td>
</tr>
</tbody>
</table>

1: INPUT: \(G,R,C\)
2: INPUT: \(ACH,REQ,\text{SUB,CON}\)
3: INPUT: \(A\)
4: INPUT: \(POS,CAP,ASN\)
5: OUTPUT: organization\(_1\), the initial state of the organization
6: OrgScore = 0
7: *AssembleGoalTree*(\(G,\text{SUB}\))
8: for all \(g \in G_{\text{leaf}}\) do
9: assignment\(_g\) = *FindAssignment*(\(g\))
10: \(ASN \leftarrow ASN \cup \{\text{assignment}_g\}\)
11: OrgScore = OrgScore + assignment\(_g\).score
12: end for
13: if \(ASN = \emptyset\) then
14: end organization
15: else
16: return organization\(_1\)
17: end if

The *Generic Initial Organization Algorithm* requires several sets to form a valid organization. The first requirement is the existence of organization objects which are goals \(G\), roles \(R\), capabilities \(C\) and agents \(A\). The second requirement is the existence of organization relationships, achieves

67
ACH, requires REQ, subgoal SUB, conjunctive CON, possesses POS, capable CAP and assignment ASN. The third requirement indicates that the organization object sets must not be null to form an organization. If any object sets are null, the organization will not form. The fourth requirement specifies that the ACH, REQ and POS relationship sets must not be empty in order to form an organization. The third and fourth requirements do not indicate that the elements of the objects sets, G,R,C or A necessarily correspond to the relationship sets, ACH, REQ or POS of the fourth requirement. The correspondence is computed later in the algorithm. These requirements simply serve as a precondition. The last requirement is that the capable, CAP and assignment, ASN, sets are empty prior to computation.

The inputs to the algorithm are listed in lines 1-4. Line 1 shows the structural objects, line 2 the structural relationships, line 3 the state objects and line 4 the state relationships. Line 5 indicates the output as the first organization instance, organization1. Line 7 calls the AssembleGoalTree function which is only performed on the initial organization. After, goals and subgoal relations are added to the tree. Line 8 - 12 loops through all g ∈ Gleaf to determine the assignment for each leaf goal. The organization score is also computed in this section of the algorithm. Finally, lines 13 - 17 test the computation to determine if a valid organization exists. If there are no assignments, there is not a valid organization1.

Algorithm 2, AssembleGoalTree, takes the set of Goals, G, and the subgoal relationship set, SUB, and assembles the organization’s goal tree. The establishment of root, groot, internal, Gin, and leaf goals, Gleaf is completed by the usage of the subgoal relationship set, SUB. Lines 2 - 4 initializes Gin, the set of internal nodes, Gleaf, the set of leaf nodes and groot, the root node. Lines 5 - 16 loop through all nodes to determine which g are in Gin or in Gleaf. After, groot is determined in lines 17 - 28.

Algorithm 3, FindAssignment, determines the best assignment candidate for a leaf goal. For each r ∈ R achieves for the passed goal this algorithm returns the best assignment based on the best role and agent assignment.

Algorithm 4, FindRoleCandidate, determines the best agent candidate to play a role, based on
Algorithm 2 Establish Goal Root, Internal Goals and Leaf Goals

1: PROCEDURE AssembleGoalTree(G, SUB)
2: $G_{\text{int}} = \emptyset$
3: $G_{\text{leaf}} = \emptyset$
4: $g_{\text{root}} = \emptyset$
5: for all $g \in G$ do
6:     internal $\leftarrow$ false
7:     for all $g_2 \in G - \{g\}$ do
8:         if subgoal($g, g_2$) $\rightarrow$ true then
9:             $G_{\text{int}} \leftarrow G_{\text{int}} \cup \{g\}$
10:                internal $\leftarrow$ false
11:         end if
12:     end for
13:     if internal $\leftarrow$ false then
14:         $G_{\text{leaf}} \leftarrow G_{\text{leaf}} \cup \{g\}$
15:     end if
16: end for
17: for all $g \in G_{\text{int}}$ do
18:     root $\leftarrow$ true
19:     for all $g_2 \in G_{\text{int}} - \{g\}$ do
20:         if subgoal($g_2, g$) $\rightarrow$ true then
21:             root $\leftarrow$ false
22:         end if
23:     end for
24: end for
25: if root $\rightarrow$ true then
26:     $g_{\text{root}} \leftarrow$ root
27:     exit for
28: end if

the collective capability of an agent to play a role.

Algorithm 5 calculates the RoleCapabilityFunction ($rcf$) previously defined in chapter 3.
Algorithm 3 Find the best assignment candidate for a leaf goal

1: PROCEDURE FindAssignment(goal)
2: bestAssignment = ∅
3: maxAssignedScore = 0
4: maxCapable = ∅
5: for all \( r \in R_{\text{achieves}}(goal) \) do
6: candidate = FindRoleCandidate\( (r_g) \)
7: assignedScore = 0
8: if assignedScore > maxAssignedScore then
9: maxAssignedScore = assignedScore
10: maxCapable = candidate
11: bestAssignment = assign\( (g,r_g,candidate) \)
12: end if
13: end for
14: return bestAssignment

Algorithm 4 Find the best agent candidate to play a role

1: PROCEDURE FindRoleCandidate\( (role) \)
2: bestCandidate = ∅
3: maxRCFScore = 0
4: mostCapable = ∅
5: for all \( a \in A \) do
6: candidate = true
7: rcfScoreSum = 0
8: mostCapable = RoleCapabilityFunction\( (role,a) \)
9: end for
10: return mostCapable
Algorithm 5 Role Capability Function (rcf)

1: PROCEDURE RoleCapabilityFunction(role, a)
2:   capSET ← ∅
3:   reqSET ← ∅
4: for all c ∈ C do
5:   if requires(r, c) → true then
6:     reqSET ← reqSET ∪ {c}
7:   end if
8: end for
9: for all a ∈ A do
10:   boolean possessesCapability ← true
11:   score = 0
12:   for all c ∈ reqSET do
13:     if possesses(a, c) = 0 then
14:       possessesCapability ← false
15:       exit for
16:     else
17:       score = score + possesses(a, c)
18:     end if
19:   end for
20:   if possessesCapability = false then
21:     capSET ← capSET ∪ {< r, a, score >}
22:   end if
23: end for
24: return capable(a, r) = max(capSET)
4.2.2 Computing Reorganization

The *Generic Reorganization Algorithm*, algorithm 6, uses as a required input an existing organization $organization_n$, computed originally by the initial organization algorithm or itself. A second requirement is a transition property $\phi$. The output is a new organization state $organization_{n+1}$.

To accomplish reorganization, the organization must update the organization with the predicate or predicates of $\phi$, using the $Install\Phi(\phi)$ function. Then, the organization is tested and if any single one of the four object sets are empty, the computation ceases, as there is no way to create a new organization state. Otherwise, the $Compute\Phi$ function is called to compute the new state.

Another test, to determine if there are valid assignments, is executed. If there are no valid assignments, there cannot be a new organization state. If valid assignments exist, the new organization state is generated.

**Algorithm 6 General Reorganization Algorithm**

**Require:** $\exists \ organization_n$

1: INPUT : $organization_n$, an organization state  
2: INPUT : $organization property \phi$, the property used to transition  
3: OUTPUT : $organization_{n+1}$, a new state of the organization  
4: GoalsToCompute = $\emptyset$  
5: GoalsToCompute = $Install\Phi(\phi)$  
6: if $G = \emptyset$ or $R = \emptyset$ or $C = \emptyset$ or $A = \emptyset$ then  
7: exit organization  
8: else  
9: ComputePhi(GoalsToCompute)  
10: end if  
11: if $ASN = \emptyset$ then  
12: end organization  
13: else  
14: return $organization_{n+1}$  
15: end if

Algorithm 7, $Install\Phi$, first calls the function to *parse* the transition predicates and add them to the organization. Then, the $GoalsToComputeList$ is initialized and later tracks all $goal_{leaf}$ whose structures need to be recomputed. Finding the $goal_{leaf}$ for each $\phi_i$ is trivial, so there is no algorithm included detailing the steps. For each simple predicate $\phi_i \in \phi$, the $goal_{leaf}$ is added to
Algorithm 7 Install Organization Predicates

1: PROCEDURE \texttt{installPhi}(\phi) 
2: \hspace{1em} \texttt{parsePhi}(\phi) 
3: \hspace{1em} \texttt{GoalsToComputeList} = \emptyset 
4: \hspace{1em} \textbf{for} \ \phi_i \in \phi \ \textbf{do} 
5: \hspace{2em} \texttt{goal} = \texttt{FindLeafGoal}(\phi_i) 
6: \hspace{2em} \texttt{GoalsToComputeList} = \texttt{GoalsToComputeList} \cup \{\texttt{goal}\} 
7: \hspace{1em} \textbf{end for} 
8: \hspace{1em} \textbf{return} \ \texttt{GoalsToComputeList} 

the \texttt{GoalsToComputeList} set. In the end, \texttt{GoalsToComputeList} is returned.

Algorithm 8 Compute the new organization

1: PROCEDURE \texttt{ComputePhi}(\texttt{GoalsToCompute}) 
2: \hspace{1em} \textbf{if} \ | \phi_{relationship} | > 0 \ \textbf{and} \ | \phi_{object} | = 0 \ \textbf{then} 
3: \hspace{2em} \textbf{for} \ g \ \in \texttt{GoalsToCompute} \ \textbf{do} 
4: \hspace{3em} \texttt{assignment}_g = \texttt{FindAssignment}(g) 
5: \hspace{3em} \texttt{ASN} \leftarrow \texttt{ASN} \cup \{\texttt{assignment}_g\} 
6: \hspace{3em} \texttt{GoalsToCompute} \leftarrow \texttt{GoalsToCompute} \setminus \{g\} 
7: \hspace{2em} \textbf{end for} 
8: \hspace{1em} \textbf{else if} \ | \phi_{relationship} | > 0 \ \textbf{and} \ | \phi_{object} | > 0 \ \textbf{then} 
9: \hspace{2em} \textbf{for} \ g \ \in \texttt{GoalsToCompute} \ \textbf{do} 
10: \hspace{3em} \texttt{assignment}_g = \texttt{FindAssignment}(g) 
11: \hspace{3em} \texttt{ASN} \leftarrow \texttt{ASN} \cup \{\texttt{assignment}_g\} 
12: \hspace{3em} \texttt{GoalsToCompute} \leftarrow \texttt{GoalsToCompute} \setminus \{g\} 
13: \hspace{2em} \textbf{end for} 
14: \hspace{1em} \textbf{else if} \ | \phi_{relationship} | = 0 \ \textbf{and} \ | \phi_{object} | > 0 \ \textbf{then} 
15: \hspace{1em} \textbf{do not recompute} 
16: \hspace{1em} \textbf{end if} 

Algorithm 8, \texttt{ComputePhi}, takes the goals to compute from \texttt{InstallPhi} and computes the new assignments from the transition property \phi. The algorithm has fours possible branches. There is one trivial branch, where \(| \phi_{relationship} | = 0 \ \textbf{and} \ | \phi_{object} | > 0\). In this case, no new relationships have been added, which alleviates the requirement to compute. This is considered a minimal impact transition. Otherwise, if \phi has new valid relationships, either with no new objects, \(| \phi_{relationship} | > 0 \ \textbf{and} \ | \phi_{object} | = 0\), or new objects and relationships are added, \(| \phi_{relationship} | > 0 \ \textbf{and} \ | \phi_{object} | > 0\), a recompute is required for \phi.
Algorithm 9 Parse Organization Predicates

1: PROCEDURE parsePhi(φ)
2: for ∀φ_add, φ_lose, φ_change ∈ φ do
3: if φ_change then
4: changePhi(φ_change)
5: else if φ_add then
6: addPhi(φ_add)
7: else if φ_lose then
8: losePhi(φ_lose)
9: end if
10: end for

Algorithm 9, ParsePhi, looks at each simple predicate, decomposed by being either φ_change, φ_lose or φ_add and calls the appropriate algorithm to change lose or add objects and relationships to the organization.

Algorithm 10 Change Predicates

1: PROCEDURE changePhi(φ_change)
2: if φ_change achieves(r,g) then
3: achieves(r,g) = φ_change achieves(r,g)
4: else if φ_change possesses(a,c) then
5: possesses(a,c) = φ_change possesses(a,c)
6: else if φ_change requires(r,c) then
7: requires(r,c) = φ_change requires(r,c)
8: end if

Algorithms 10, 11 and 12 add, delete and change elements from the organizations object and relationship set based on the predicates of φ, respectively. Algorithm 10 takes a φ_change predicate and applies it to one of the three numerical relationships of the organization, achieves, possesses or requires. The change serves to alter, increasing or decreasing the numerical score or this relationship.

To add an object or relationship to the organization, addPhi is utilized, which is algorithm 11. Any of the structural or state objects or relationships can be added.

Algorithm 12, LosePredicates, is more complex than the other predicate add or change algorithms. When an object is lost, the relationships it maintains to other objects cannot remain. In terms of deletion, the object and its relationships must be deleted. When an internal goal g ∈ g_int is
Algorithm 11 Add Predicates

1: PROCEDURE addPhi(\(\phi_{add}\))
2: if \(\phi_{addgoal}(g)\) then
3: \(G_{leaf} \leftarrow G_{leaf} \cup \{g\}\)
4: else if \(\phi_{addgoal}(g)\) then
5: \(G_{int} \leftarrow G_{int} \cup \{g\}\)
6: else if \(\phi_{addrole}(r)\) then
7: \(R \leftarrow R \cup \{r\}\)
8: else if \(\phi_{addcapability}(c)\) then
9: \(C \leftarrow C \cup \{c\}\)
10: else if \(\phi_{addagent}(a)\) then
11: \(A \leftarrow A \cup \{a\}\)
12: else if \(\phi_{addachieves}(r, g)\) then
13: \(ACH \leftarrow ACH \cup \{achieves(r, g)\}\)
14: else if \(\phi_{addrequires}(r, c)\) then
15: \(REQ \leftarrow REQ \cup \{requires(r, c)\}\)
16: else if \(\phi_{addpossesses}(a, c)\) then
17: \(POS \leftarrow POS \cup \{possesses(a, c)\}\)
18: else if \(\phi_{addsubgoal}(g_1, g_2)\) then
19: \(SUB \leftarrow SUB \cup \{subgoal(g_1, g_2)\}\)
20: else if \(\phi_{addconjunctive}(g)\) then
21: \(CON \leftarrow CON \cup \{conjunctive(g)\}\)
22: end if

deleted the subgoal and conjunctive relationships must also be deleted. When a leaf goal \(g \in g_{leaf}\) is deleted, its achieves relationships must be deleted. When a role \(r \in R\) is deleted, all achieves and requires relationships must be deleted. Capability \(c \in C\) deletions require requires and possesses relationships to be deleted. Agent \(a \in A\) deletions require possesses relationships to be deleted. The capable and assignment relationships are generated at the reorganization and are deleted, at that time.
Algorithm 12 Lose Predicates

1: PROCEDURE losePhi(φlose)
2: if φlose goal(g) then
3:   G ← G − {g}
4:   ACH ← ACH − {achieves(x, g)}
5:   SUB ← SUB − {subgoal(g, gn)}
6:   CON ← CON − {conjunctive(g)}
7: else if φlose role(r) then
8:   R ← R − {r}
9:   ACH ← ACH − {achieves(r, x)}
10:  REQ ← REQ − {requires(r, x)}
11: else if φlose capability(c) then
12:   C ← C − {c}
13:   REQ ← REQ − {requires(x, c)}
14:   POS ← POS − {possesses(x, c)}
15: else if φlose agent(a) then
16:   A ← A − {a}
17:   POS ← POS − {possesses(a, x)}
18: else if φlose achieves(r, g) then
19:   ACH ← ACH − {achieves(r, g)}
20: else if φlose requires(r, c) then
21:   REQ ← REQ − {requires(r, c)}
22: else if φlose possesses(a, c) then
23:   POS ← POS − {possesses(a, c)}
24: else if φlose subgoal(g1, g2) then
25:   SUB ← SUB − {subgoal(g1, g2)}
26: Change Gint or Gleaf
27: else if φlose conjunctive(g) then
28:   CON ← CON − {conjunctive(g)}
29: end if
4.2.3 Evaluation of Generic Initial Organization

Determination of a successful initial organization consists of two measures. The first is the completeness measure to determine whether an organization can be formed from the sets of objects and relationships available. If the organization cannot complete successfully, the second part is inconsequential. If an organization can be formed, the time to compute the organization is of paramount importance. The time taken to compute an initial organization is a function of the overall organization size, where $Org_{size}$ is given as:

$$Org_{size} = Objects_{size} + Relationships_{size}$$  \hspace{1cm} (28)

The organization size consists of the addition of the $Objects_{size}$ and the $Relationships_{size}$ defined by the sizes of all objects sets:

$$Objects_{size} = |G| + |R| + |C| + |A|$$  \hspace{1cm} (29)

and all relationship sets:

$$Relationships_{size} = |ACH| + |REQ| + |POS| + |SUB| + |CON|$$  \hspace{1cm} (30)

The time to compute is driven by the size of the total organization size. Fig. 36 shows the progression through 4 initial organization scenarios. Scenario 1 is the most trivial organization possible, where there is one possible assignment. The computation of this trivial case establishes a baseline for a computable organization for an organization with $Org_{size} = 9$. Scenarios 2 and 3 add another set of objects and relationships, although the organizations remain rather simple. Scenario 4 involves more complexity as the number of relationships between the objects increases. This naturally increases the computation as more capable relationships are computed and more possible assignment relationships are generated. Table 4 shows the scenario, $Objects_{size}$, $Relationships_{size}$ and total $Org_{size}$ for the 4 scenarios.
Table 5 shows the data for a set of initial organization computations, differentiated by $\text{Org}_{\text{size}}$. The data is similar to Table 4 with the addition of $\text{Time}_{\text{avg}}(1000)$. The times for each initial organization execution are shown with the related $\text{Objects}_{\text{size}}$, $\text{Relationships}_{\text{size}}$ and total $\text{Org}_{\text{size}}$. Fig. 37 shows the graphical representation of the 10 initial organization executions. $\text{Time}_{\text{avg}}(1000)$ was computed by executing each initial organization 1000 times and taking the average.

Each initial organization and reorganization algorithm has a micro-evaluation small organizations to view nuance information not seen in larger organization evaluations. The complete algorithm set for each initial organization and reorganization is developed in Java jdk1.6.003. The evaluations were executed on Intel Core 2 Pentium CPU T7400 computer clocked to 2.16 GHz with 2 GB of main memory. The initial organization times for each algorithm is compared to expose small scale differences. The reorganization algorithms are executed with increasing numbers of objects and relationships in add format as well as decreasing numbers of objects and relationships in delete format. Each algorithm is executed using the exact sizes, adds and deletions under similar operating conditions. The basis thesis of the micro evaluation is that the addition and deletion of
relationships to an organization is much costlier, in terms of compute time, than the addition or deletion of objects. A second assertion is there is not a large scale divergence in runtime due to the small scale of organization.

Table 5: Generic Initial Organization

<table>
<thead>
<tr>
<th>Initial Organization</th>
<th>Objects size</th>
<th>Relationships size</th>
<th>Org size</th>
<th>Time_avg(1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>0.00041044</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>17</td>
<td>0.00062692</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>12</td>
<td>25</td>
<td>0.00064342</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>18</td>
<td>33</td>
<td>0.00076881</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>20</td>
<td>41</td>
<td>0.00083314</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>25</td>
<td>46</td>
<td>0.00085765</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>40</td>
<td>61</td>
<td>0.00086810</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>45</td>
<td>66</td>
<td>0.00088114</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>60</td>
<td>81</td>
<td>0.00095459</td>
</tr>
<tr>
<td>10</td>
<td>21</td>
<td>65</td>
<td>86</td>
<td>0.00096183</td>
</tr>
</tbody>
</table>

Figure 37: Initial Generic Organization
4.2.4 Evaluation of Generic Reorganization

For reorganization, the size of the objects and relationships in $organization_n$ is one key element. The second element is the size of transition property, $\phi$, in size of objects and relationships to be added. There is no restriction on the size of $\phi$ or $Org_{size}$ in relation to each other. It is possible that $\phi_{size}$ is larger the existing $Org_{size}$.

The process of reorganization considers the total organization size, $Org_{size} = Objects_{size} + Relationships_{size}$, with which it begins. Then, $\phi$ adds additional objects and relationships, so we consider the size of $\phi$ in these terms:

$$\phi_{size} = \sum_{i=1}^{n} \phi_{add_i} + \sum_{j=1}^{m} \phi_{lose_j}$$

(31)

Change predicates $\phi_{change}$ do not add or delete objects or relationships and therefore do not change the size. The size of the organization property in terms of predicates is not necessarily additive. A $\phi_{lose}$ predicate reduces the number of relationships and/or properties in the next organization state. A $\phi_{lose}$ can reduce the total $Org_{size}$ by greater than it size if it contains the loss of objects that are also cause the removal of linked relationships.

$$Org_{size_{n+1}} = Org_{size_{n}} + \phi_{size}$$

(32)

The resulting reorganization size is $Org_{size_{n+1}}$, based upon the combination of the existing $Org_{size_{n}}$ and the addition or subtraction of the elements of the predicate $\phi$.

In Table 6, the additions to the organization are chronicled starting with an existing initial organization and transitioning through 10 new organization states. The additions to the organization are similar to the additions made to the initial organization data previously. In this case, the initial organization is trivial with an $Org_{size} = 9$ where $Object_{size} = 5$ and $Relationship_{size} = 4$. Each transition property then adds a distinct number of objects and relationships through the remainder
of the transitions.

<table>
<thead>
<tr>
<th>Organization State</th>
<th>Orgsizeₙ</th>
<th>φₙadd</th>
<th>φₙdelete</th>
<th>Orgsizeₙ₊₁</th>
<th>Timeavg(1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>8</td>
<td>0</td>
<td>17</td>
<td>0.00006396</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>8</td>
<td>0</td>
<td>25</td>
<td>0.00002442</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>8</td>
<td>0</td>
<td>33</td>
<td>0.00001558</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
<td>8</td>
<td>0</td>
<td>41</td>
<td>0.00001752</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
<td>5</td>
<td>0</td>
<td>46</td>
<td>0.00001564</td>
</tr>
<tr>
<td>6</td>
<td>46</td>
<td>15</td>
<td>0</td>
<td>61</td>
<td>0.00003755</td>
</tr>
<tr>
<td>7</td>
<td>61</td>
<td>5</td>
<td>0</td>
<td>66</td>
<td>0.00001730</td>
</tr>
<tr>
<td>8</td>
<td>66</td>
<td>15</td>
<td>0</td>
<td>81</td>
<td>0.00004761</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>5</td>
<td>0</td>
<td>86</td>
<td>0.00001448</td>
</tr>
<tr>
<td>10</td>
<td>86</td>
<td>5</td>
<td>0</td>
<td>91</td>
<td>0.00001299</td>
</tr>
</tbody>
</table>

Fig. 38 depicts the reorganizations, based upon the addition of new organization relationships and objects. The reorganizations at states 2,3,4,5,7,9,10 are relatively similar in time. That is due to what was added. The number of objects or relationships was low in each case. For reorganizations 6 and 8, the addition or 15 new relationships causes a large spike in the time to compute due to all of the new capable and assignment relationships that must be considered.

Figure 38: Generic Reorganization with increase in organization size
In Table 7, deletions from the organization are shown starting with the organization of 91 elements and going in reverse order of the previous add executions and transitioning through 10 new organization states. The runs begin with an $Org_{size} = 91$ and result in an $Org_{size}$ of 9 after the 10th transition.

<table>
<thead>
<tr>
<th>Organization State</th>
<th>$Org_{size_n}$</th>
<th>$\phi_{add}$</th>
<th>$\phi_{delete}$</th>
<th>$Org_{size_{n+1}}$</th>
<th>$Time_{avg}(1000)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>91</td>
<td>0</td>
<td>5</td>
<td>86</td>
<td>0.00002446</td>
</tr>
<tr>
<td>2</td>
<td>86</td>
<td>0</td>
<td>5</td>
<td>81</td>
<td>0.00001174</td>
</tr>
<tr>
<td>3</td>
<td>81</td>
<td>0</td>
<td>15</td>
<td>66</td>
<td>0.00002898</td>
</tr>
<tr>
<td>4</td>
<td>66</td>
<td>0</td>
<td>5</td>
<td>61</td>
<td>0.00001020</td>
</tr>
<tr>
<td>5</td>
<td>61</td>
<td>0</td>
<td>15</td>
<td>46</td>
<td>0.00002654</td>
</tr>
<tr>
<td>6</td>
<td>46</td>
<td>0</td>
<td>5</td>
<td>41</td>
<td>0.00000984</td>
</tr>
<tr>
<td>7</td>
<td>41</td>
<td>0</td>
<td>8</td>
<td>33</td>
<td>0.00001246</td>
</tr>
<tr>
<td>8</td>
<td>33</td>
<td>0</td>
<td>8</td>
<td>25</td>
<td>0.00001143</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>0</td>
<td>8</td>
<td>17</td>
<td>0.00001086</td>
</tr>
<tr>
<td>10</td>
<td>17</td>
<td>0</td>
<td>8</td>
<td>9</td>
<td>0.00001106</td>
</tr>
</tbody>
</table>

In Fig. 39, the highest average reorganization times are for 7, 9 and 10. The reduction of organization objects causes complete reassignment in these cases maximizing computation.

Figure 39: Generic Reorganization with decrease in organization size
4.3 Agent Perspective

It is important to understand the perspective of agents participating in an organization. Agents can be a physical manifestation, either in space or memory, or they can be virtual depending on their specific organization perspective. Can an agent see only his assignments? Can an agent see all assignments and organization data? In general, the extent to what an agent knows about the organization and where it fits into the organization is required. This section discusses the dual nature of an agent, both as internal and external to the organization.

Fig. 40 and Fig. 41 shows a structural map and a state map of an organization, respectively. In each, the agents are a part of the organization. From the perspective of this research, it is of paramount importance to consider the dual nature of an agent. An agent exists in the organization, but the maps are strictly an artifact within the memory of the agent containing all of the information the agent knows about the organization it participates within. An example of this is a worker in a company’s organization chart. From the chart, the worker knows the relationships to all other workers and where he fits into the organization, but the worker is not physically a part of the organization chart.

The agent’s perspective can be split into distinct views of the organization, as shown in Fig. 42. A global view indicates an agent can see all objects and relationships of the organization. Local view

Figure 40: Structural Map

Figure 41: State Map
is defined by an agent only having access to what concerns them, such as their specific assignments. A \textit{segmented view} allows an agent access to information within the segment to which they belong. An agent within a segment can have a global or local view of that segment.

![Figure 42: Agent’s Dual Nature](image)

A generic organization agent, $a \in A$ was defined in chapter 3. This definition is the agent’s representation within the organization. This representation defines the internal facet of the agent’s dual nature. The external facet of an operational agent is a set of knowledge cores as defined by:

$$AGENT = \langle \text{Organization}, \text{Communication}, \text{Task} \rangle \quad (33)$$

where \text{Organization} is an instantiated organization to which the agent belongs, \text{Communications} is a set of communications protocols to communicate with other agents in the organization and \text{Task} a set of tasks for the agent to complete, based upon the agent’s capabilities. The organization core is the central focus of this research, although elements of the Communications and Task cores are introduced where required. The elements of an organization reside in the \textit{Organization Core}, as shown in Fig. 43. Each $agent \in AGENT$ have a subset of the organization core for the organization to which it belongs. The content of the agent’s organization subset may span from empty to complete information.

The complete depiction of an agent’s cores is shown in Fig. 44. Each of the cores can be
considered a knowledge base of information for that particular aspect of an agent. The organization core contains all information about the organization. The organization elements exchange with the communicational aspects of the agent so that agents can exchange information.

The final aspect of an agent is the perspective or view of the agent, in relation to the organization. An agent may possess a global view, indicating that the agent can see all objects and relationships of the organization. Because the agent has full organizational knowledge, it can compute a new organization. An agent may also have a local view, limiting its perspective to only seeing organization objects and relationships relating to itself, but no others. Finally, an agent can have a segment view. The segment view can be either global to the segment or local to the segment. These are similar to general global and local views, except they are limited to the specific segment...
in which the agent belongs.

Transition properties $\Phi$ are not given to the organization as a whole. They are perceived by a single agent, $agent \in AGENT$, in the organization and propagated to the remainder of the agents depending on the transition algorithm model that is employed.
4.4 Central Control Transition Algorithm

Central Control Transition Algorithm indicates that a single agent dictates decisions on which agents play what roles for each organization state. The advantage of central control is that the computation to derive a new organization is done by a single agent indicating a single computation of the organization. The positive effects of this approach are minimal computation, propagation of n-1 messages to update all other agents and n-1 updates to each agents organization core. This can also have negative outcomes. If the controlling agent exits the organization, a problem is created in losing the ability to compute the organization and make assignments to all other agents. In other words, there exists a single point of failure.

Prior to computation, there is a defined set of agents, $AGENT$ containing a leader agent $agent_{leader}$ and a set of subordinate agents $AGENTS_{subordinate} = \{agent_1, agent_2, \ldots, agent_n\}$, where $AGENT = \{agent_{leader}, AGENTS_{subordinate}\}$. The $agent_{leader}$ contains complete knowledge of all aspects of the organization, communication and task cores. Each subordinate agent begins in the following state where the agent, $agent_i = <\emptyset, COMMUNICATION, TASK>$. Each subordinate agent has knowledge of tasks it may complete and how to communicate with a set of other agents, but no knowledge of what it is supposed to do. Prior to commencement, each $agent_{subordinate}$ operates only on future organization knowledge sent by the $agent_{leader}$. A diagram of a organization process using the Central Control algorithm is displayed in Fig. 45, where $agent_1$ is the $agent_{leader}$.

There is a potential organization recovery strategy, involving designation of a new control agent, recreating the inputs to the organization, computing the organization and updating all other agents with a new set of assignments.

4.4.1 Initial Organization

The basic intuition of central control initial organization the $agent_{leader}$ takes all input to compute an organization. The $agent_{leader}$ then computes an initial organization state. Then, for each
Figure 45: Central Control

agent ∈ AGENT, the agent leader sends assignments to each agent assigned some task during the computation. Alternatively, the agent leader may only send information where an agent has changes in assignments to further reduce messaging. Each $a \in AGENT$ loads the assignment information. Finally, recomputation terminates and the initial state of execution begins.

Algorithm 13, the Central Control Initial Organization Algorithm is an extension of Algorithm 1, the Generic Initial Organization Algorithm, placed in the singular perspective of a control agent, agent leader. Algorithm 13 takes as input all organization elements, used by the agent leader, which computes the initial organization state. Line 7 calls the ComputeInitialCentralOrganization.

Algorithm 14 which assemble the goal tree, computes all assignments and returns the complete organization, if there is a viable organization which is computable. The algorithms common to the Generic Initial Organization algorithm, such as AssembleGoalTree and FindAssignment are common.

Line 8 tests if the organizations assignment set $ASN = \emptyset$. If this is true, there is not a valid organization state to form. Otherwise, there is a valid set of assignments. In case of an
$ASN \neq \emptyset$, the $\text{agent}_{\text{leader}}$ sends the assignments from $ASN_{ORG}$ to each $agent \in AGENT$ using the $\text{Communicate}$ function. The $\text{Communicate}$ function is trivial and dependent upon protocols and platform implementation, so is not detailed here.

When the $\text{agent}_{\text{leader}}$ has complete propagation of all assignments to each utilized $agent$, the organization computation is complete and the organization instance may begin its execution phase.

**Algorithm 13** Central Control Initial Organization Algorithm

**Require:** $\exists G, R, L, C, A$

**Require:** $\exists ACH, REQ, SUB, CON, POS, CAP, ASN$

**Require:** $\mid G \mid > 0, \mid R \mid > 0, \mid C \mid > 0, \mid A \mid > 0$

**Require:** $\mid ACH \mid > 0, \mid REQ \mid > 0, \mid POS \mid > 0$

**Require:** $\mid CAP \mid = 0, \mid ASN \mid = 0$

1: INPUT : $G, R, C$

2: INPUT : $ACH, REQ, SUB, CON$

3: INPUT : $A$

4: INPUT : $POS, CAP, ASN$

5: INPUT : $\text{agent}_{\text{leader}}$

6: OUTPUT : $\text{organization}_1$, the initial state of the central control organization

7: $ORG = \text{ComputeInitialCentralOrganization}(G, R, C, A, ACH, REQ, POS, CAP, SUB)$

8: if $ASN_{ORG} \neq \emptyset$ then

9: for all $agent \in AGENT$ do

10: $\text{Communicate}(agent, ASN_{ORG}(agent))$

11: end for

12: else

13: end organization

14: end if

**Algorithm 14** Compute Initial Central Organization Algorithm

1: PROCEDURE $\text{ComputeInitialCentralOrganization}(G, R, C, A, ACH, REQ, POS, CAP, SUB)$

2: OrgScore = 0

3: $\text{AssembleGoalTree}(G, SUB)$

4: for all $g \in G_{\text{leaf}}$ do

5: $\text{assignment}_g = \text{FindAssignment}(g)$

6: $ASN \leftarrow ASN \cup \{\text{assignment}_g\}$

7: $\text{OrgScore} = \text{OrgScore} + \text{assignment}_g.\text{score}$

8: end for

9: return $ORG$
4.4.2 Evaluation of Central Control Initial Organization

The data for the *Central Control Initial Organization* is shown in Table 8. The size of each initial organization execution is the same as for the previous generic evaluation. With centralized organization, messaging is low and there is only a single compute cycle, although the organization must be passed to each other agent from the *agent leader*.

Table 8: Central Control Initial Organization

<table>
<thead>
<tr>
<th>Initial Organization</th>
<th>Objects size</th>
<th>Relationships size</th>
<th>Org size</th>
<th>Time ave (1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>0.00046433</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>17</td>
<td>0.00061624</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>12</td>
<td>25</td>
<td>0.00073011</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>18</td>
<td>33</td>
<td>0.00084872</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>20</td>
<td>41</td>
<td>0.00099805</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>25</td>
<td>46</td>
<td>0.00097791</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>40</td>
<td>61</td>
<td>0.00098502</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>45</td>
<td>66</td>
<td>0.00102578</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>60</td>
<td>81</td>
<td>0.00103335</td>
</tr>
<tr>
<td>10</td>
<td>21</td>
<td>65</td>
<td>86</td>
<td>0.00136323</td>
</tr>
</tbody>
</table>

Figure 46: Central Control Initial Organization
The time of computation for each execution for *Central Control Initial Organization* is shown in Fig. 46. While there are different increments, the time taken to organize rises consistently with the size increase.
### 4.4.3 Reorganization

Algorithm 15, the *Central Control Reorganization Algorithm* is an extension of Algorithm 6, the *Generic Reorganization Algorithm*, placed in the singular perspective of a control agent, \(agent_{\text{leader}}\).

The basic intuition for Algorithm 15, Central Control Reorganization Algorithm, is the \(agent_{\text{leader}}\) receives a transition property \(\phi\), indicating a change to the organization and a transition. The \(agent_{\text{leader}}\) takes the current organization state and the \(\phi\) as input and computes a new organization state. Then, for each \(agent \in AGENT\), the \(agent_{\text{leader}}\) sends assignments to each agent assigned some task during the computation. Alternatively, the \(agent_{\text{leader}}\) may only send where an agent has changes in assignments to further reduce messaging. Each \(a \in AGENT\) loads the assignment information. Finally, recomputation terminates and the initial state of execution begins.

#### Algorithm 15 Central Control Reorganization Algorithm

**Require:** \(\exists \text{organization}_n, \exists agent_{\text{leader}}\)

1: INPUT : \(\text{organization}_n\), an organization state

2: INPUT : organization property \(\phi\), the property used to transition

3: OUTPUT : \(\text{organization}_{n+1}\), a new state of the organization

4: GoalsToCompute = \(\emptyset\)

5: GoalsToCompute = \(\text{InstallPhi(}\phi)\)

6: if \(G = \emptyset\) or \(R = \emptyset\) or \(C = \emptyset\) or \(A = \emptyset\) then

7: exit organization

8: else

9: ComputePhi(GoalsToCompute)

10: end if

11: if \(ASN_{\text{ORG}} \neq \emptyset\) then

12: for all \(agent \in AGENT\) do

13: \(\text{Communicate}(agent, ASN_{\text{ORG}}(agent))\)

14: end for

15: else

16: end organization

17: end if

Algorithm 15, the *Compute Central Control Reorganization Algorithm* is similar to Algorithm 6, the *Generic Reorganization Algorithm*. The first difference is a precondition of \(\exists agent_{\text{leader}}\). The second difference is the distribution of \(ASN_{\text{ORG}}\) at the end by the \(agent_{\text{leader}}\) to each \(a \in A\).
4.4.4 Evaluation of Central Control Reorganization

The data for Central Control Reorganization is shown in Table 9 and the graph is shown in Fig. 47. This reorganization evaluation adds objects and relationships to the organization with each \( \phi \) applied. The first reorganization has the highest time with the addition of 8 new objects. The other high time to complete reorganizations are 6 and 8, which are to be expected, as they add 15 relationships each. The addition of relationships increase the complexity of reorganization as it does with initial organization.

<table>
<thead>
<tr>
<th>Organization State</th>
<th>Org Size, ( n )</th>
<th>( \phi ) Add</th>
<th>( \phi ) Delete</th>
<th>Org Size, ( n+1 )</th>
<th>Time Avg (1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>8</td>
<td>0</td>
<td>17</td>
<td>0.00006542</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>8</td>
<td>0</td>
<td>25</td>
<td>0.00002340</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>8</td>
<td>0</td>
<td>33</td>
<td>0.00002239</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
<td>8</td>
<td>0</td>
<td>41</td>
<td>0.00001680</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
<td>5</td>
<td>0</td>
<td>46</td>
<td>0.00001538</td>
</tr>
<tr>
<td>6</td>
<td>46</td>
<td>15</td>
<td>0</td>
<td>61</td>
<td>0.00003604</td>
</tr>
<tr>
<td>7</td>
<td>61</td>
<td>5</td>
<td>0</td>
<td>66</td>
<td>0.00001540</td>
</tr>
<tr>
<td>8</td>
<td>66</td>
<td>15</td>
<td>0</td>
<td>81</td>
<td>0.00004261</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>5</td>
<td>0</td>
<td>86</td>
<td>0.00001442</td>
</tr>
<tr>
<td>10</td>
<td>86</td>
<td>5</td>
<td>0</td>
<td>91</td>
<td>0.00001460</td>
</tr>
</tbody>
</table>

The data for Central Control Reorganization where \( \phi \) deletes objects and relationships is shown in Table 10 and Fig. 48. The interesting reorganizations are 3 and 5 whose times are larger than the other reorganization executions. These reorganizations instances have a \( \phi \) which deletes 15 relationships each time. As with the addition of relationships, the deletion of relationships shows a much higher reorganize time.
Figure 47: Central Reorganization with increase in organization size

Table 10: Central Control Reorganization - Delete

<table>
<thead>
<tr>
<th>Organization State</th>
<th>Org Size</th>
<th>Add</th>
<th>Delete</th>
<th>Org Size</th>
<th>Time Avg (1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>91</td>
<td>0</td>
<td>5</td>
<td>86</td>
<td>0.00002264</td>
</tr>
<tr>
<td>2</td>
<td>86</td>
<td>0</td>
<td>5</td>
<td>81</td>
<td>0.00001228</td>
</tr>
<tr>
<td>3</td>
<td>81</td>
<td>0</td>
<td>15</td>
<td>66</td>
<td>0.00002879</td>
</tr>
<tr>
<td>4</td>
<td>66</td>
<td>0</td>
<td>5</td>
<td>61</td>
<td>0.00001019</td>
</tr>
<tr>
<td>5</td>
<td>61</td>
<td>0</td>
<td>15</td>
<td>46</td>
<td>0.00002664</td>
</tr>
<tr>
<td>6</td>
<td>46</td>
<td>0</td>
<td>5</td>
<td>41</td>
<td>0.00000995</td>
</tr>
<tr>
<td>7</td>
<td>41</td>
<td>0</td>
<td>8</td>
<td>33</td>
<td>0.00001231</td>
</tr>
<tr>
<td>8</td>
<td>33</td>
<td>0</td>
<td>8</td>
<td>25</td>
<td>0.00001120</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>0</td>
<td>8</td>
<td>17</td>
<td>0.00001078</td>
</tr>
<tr>
<td>10</td>
<td>17</td>
<td>0</td>
<td>8</td>
<td>9</td>
<td>0.00001095</td>
</tr>
</tbody>
</table>
Figure 48: Central Reorganization with decrease in organization size
4.5 Distributed Control Transition Algorithm - Complete Knowledge

There are two distinct algorithms for Distributed Control Transition. The first is a model where each agent $\in AGENT$ has a complete representation of the organization. This is considered as complete knowledge. The second option is where each agent $\in AGENT$ has only local knowledge of the organization. This section describes the complete knowledge distributed transition algorithms.

Distributed algorithms have no single point of failure as all knowledge is either contained within each agent or can be reconstructed from the collection of agents, assuming reliable communication between agents. Whereas with a central control model, there is a single point of failure. The distribution of knowledge translates into higher storage and messaging during transition. Messaging is increased due to the negotiation that must take place for agents to agree. Storage is higher as agents necessarily keep more information.

As a contrast to the Central Control algorithms where the agentleader is the only agent who can receive $\phi$, the distributed control algorithms allow any agent $\in AGENT$ to receive a transition property. The distributed model takes the agent who receives the transition property $\phi$ to be the agentorganizer for this specific transition. The agent $\in AGENT$ acting as the agentorganizer can change each transition.

4.5.1 Distributed Control Initial Organization - Complete Knowledge

Fig. 49, shows the basic premise of the Distributed Control Complete Knowledge Initial Organization. The agent $a_1$, as the agentorganizer, computes the organization and then communicate a copy of the organization to each of the other agents in the organization. Because there is full knowledge, there is no incentive to negotiate roles as any agent designated as the agentorganizer arrive at the same initial organization outcome.

Algorithm 16, the Distributed Control Complete Knowledge Initial Organization Algorithm is an extension of Algorithm 1, the Generic Initial Organization Algorithm. A key difference is the distributed algorithm utilizes an agentorganizer, which can change, instead of an agentleader, utilized
in Algorithm 13, the Central Control Initial Organization Algorithm. A second extension is the entire ORG computed is sent to each agent $\in AGENT$, if $ASN_{ORG} \neq NULL$.

Algorithm 16 Distributed Control Complete Knowledge Initial Organization Algorithm

Require: $\exists G, R, L, C, A$

Require: $\exists ACH, REQ, SUB, CON, POS, CAP, ASN$

Require: $|G| > 0$, $|R| > 0$, $|C| > 0$, $|A| > 0$

Require: $|ACH| > 0$, $|REQ| > 0$, $|POS| > 0$

Require: $|CAP| = 0$, $|ASN| = 0$

1: INPUT: $G, R, C, A$
2: INPUT: $ACH, REQ, SUB, CON, POS, CAP, ASN$
3: INPUT: $agent_{leader}$
4: OUTPUT: $organization_{1}$, the initial state of the central control organization
5: $ORG = ComputeInitialDistCompleteOrganization(G, R, C, A, ACH, REQ, POS, CAP, SUB)$
6: if $ASN_{ORG} \neq NULL$ then
7:     for all $agent \in AGENT$ do
8:         Communicate($agent, ORG$)
9:     end for
10: end if

Algorithm 17, the Compute Distributed Complete Knowledge Organization Algorithm takes the organization objects and relationships as input, computes and then returns an initial ORG.

Table 11 contains the data for the initial 10 organization executions and Fig. 50 shows the...
Algorithm 17 Compute Initial Distributed Complete Organization Algorithm

1: \( \text{PROCEDURE ComputeInitialDistCompleteOrganization}(G, R, C, A, ACH, REQ, POS, CAP, SUB) \)
2: \( \text{OrgScore} = 0 \)
3: \( \text{AssembleGoalTree}(G, SUB) \)
4: \( \text{for all } g \in G_{\text{leaf}} \text{ do} \)
5: \( \text{assignment}_g = \text{FindAssignment}(g) \)
6: \( \text{ASN} \leftarrow \text{ASN} \cup \{\text{assignment}_g\} \)
7: \( \text{OrgScore} = \text{OrgScore} + \text{assignment}_g.\text{score} \)
8: \( \text{end for} \)
9: \( \text{return } ORG \)

Graphical display of the table data. With the exception of initial organizations 1, 9 and 10, each of the executions follows a trend of increasing as the size of the organization increases, although the rate of increase diminishes as the organizations get larger.

4.5.2 Evaluation of Distributed Control Initial Organization - Complete Knowledge

<table>
<thead>
<tr>
<th>Initial Organization</th>
<th>Objects_{size}</th>
<th>Relationships_{size}</th>
<th>Org_{size}</th>
<th>Time_{avg}(1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>0.00061228</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>17</td>
<td>0.00057401</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>12</td>
<td>25</td>
<td>0.00076759</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>18</td>
<td>33</td>
<td>0.00097748</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>20</td>
<td>41</td>
<td>0.00101506</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>25</td>
<td>46</td>
<td>0.00104660</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>40</td>
<td>61</td>
<td>0.00106133</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>45</td>
<td>66</td>
<td>0.00106395</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>60</td>
<td>81</td>
<td>0.00104839</td>
</tr>
<tr>
<td>10</td>
<td>21</td>
<td>65</td>
<td>86</td>
<td>0.00100368</td>
</tr>
</tbody>
</table>
Figure 50: Distributed Initial Organization - Complete Knowledge
4.5.3 Distributed Control Reorganization - Complete Knowledge

Algorithm 18, the *Distributed Control Complete Knowledge Reorganization Algorithm* is an extension of Algorithm 6, the *Generic Reorganization Algorithm*.

**Algorithm 18** Distributed Control Complete Reorganization Algorithm

```
Require: ∃ organization_n, ∃ agent_organizer
1: INPUT : organization_n, an organization state
2: INPUT : organization property φ, the property used to transition
3: OUTPUT : organization_{n+1}, a new state of the organization
4: GoalsToCompute = ∅
5: GoalsToCompute = InstallPhi(φ)
6: if G = ∅ or R = ∅ or C = ∅ or A = ∅ then
7:   exit organization
8: else
9:   ComputePhi(GoalsToCompute)
10: end if
11: if ASNORG ≠ NULL then
12:   for all agent ∈ AGENT do
13:     Communicate(agent, ORG)
14:   end for
15: else
16: end organization
17: end if
```

Fig. 51, shows the basic premise of the Distributed Control Complete Knowledge Reorganization. The agent a_1, as the agent_organizer, computes the complete knowledge reorganization and if a valid ASN ≠ ∅ results, a_1 communicates the organization to each of the other agents. Once again, as there is full knowledge, there is no computational incentive to negotiate roles as any agent designated to be the agent_organizer arrives at the same outcome.

4.5.4 Evaluation of Distributed Control Reorganization - Complete Knowledge

Table 12 and Fig. 52 indicate the addition of large φ with greater number of φ_{relationships} increases the amount of time to reorganize in a complete knowledge distributed formation. Likewise, Table 13 and Fig. 53 show the data for the reorganizations based on φ with only deletions. The largest
Figure 51: Distributed Control Complete Knowledge Reorganization

Table 12: Distributed Control Complete Knowledge Reorganization - Add

<table>
<thead>
<tr>
<th>OrganizationState</th>
<th>Org(_{size_n})</th>
<th>(\phi_{add})</th>
<th>(\phi_{delete})</th>
<th>Org(<em>{size</em>{n+1}})</th>
<th>Time(_{avg}(1000))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>8</td>
<td>0</td>
<td>17</td>
<td>0.00010277</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>8</td>
<td>0</td>
<td>25</td>
<td>0.00001643</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>8</td>
<td>0</td>
<td>33</td>
<td>0.00001525</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
<td>8</td>
<td>0</td>
<td>41</td>
<td>0.00001506</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
<td>5</td>
<td>0</td>
<td>46</td>
<td>0.00001568</td>
</tr>
<tr>
<td>6</td>
<td>46</td>
<td>15</td>
<td>0</td>
<td>61</td>
<td>0.00003853</td>
</tr>
<tr>
<td>7</td>
<td>61</td>
<td>5</td>
<td>0</td>
<td>66</td>
<td>0.00001494</td>
</tr>
<tr>
<td>8</td>
<td>66</td>
<td>15</td>
<td>0</td>
<td>81</td>
<td>0.00003741</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>5</td>
<td>0</td>
<td>86</td>
<td>0.00002506</td>
</tr>
<tr>
<td>10</td>
<td>86</td>
<td>5</td>
<td>0</td>
<td>91</td>
<td>0.00001338</td>
</tr>
</tbody>
</table>

spikes in time to reorganize are when the deletion of \(\phi_{relationships}\) is greatest at reorganizations thereby forcing the large recomputation at 3 and 5.
Figure 52: Distributed Reorganization - Complete Knowledge with increase in organization size

Table 13: Distributed Control Complete Knowledge Reorganization - Delete

<table>
<thead>
<tr>
<th>Organization State</th>
<th>Org\textsubscript{size\textsubscript{n}}</th>
<th>(\phi\text{add} )</th>
<th>(\phi\text{delete} )</th>
<th>Org\textsubscript{size\textsubscript{n+1}}</th>
<th>Time\textsubscript{avg}(1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>91</td>
<td>0</td>
<td>5</td>
<td>86</td>
<td>0.00002249</td>
</tr>
<tr>
<td>2</td>
<td>86</td>
<td>0</td>
<td>5</td>
<td>81</td>
<td>0.00001195</td>
</tr>
<tr>
<td>3</td>
<td>81</td>
<td>0</td>
<td>15</td>
<td>66</td>
<td>0.00002817</td>
</tr>
<tr>
<td>4</td>
<td>66</td>
<td>0</td>
<td>5</td>
<td>61</td>
<td>0.00001041</td>
</tr>
<tr>
<td>5</td>
<td>61</td>
<td>0</td>
<td>15</td>
<td>46</td>
<td>0.00002669</td>
</tr>
<tr>
<td>6</td>
<td>46</td>
<td>0</td>
<td>5</td>
<td>41</td>
<td>0.00000995</td>
</tr>
<tr>
<td>7</td>
<td>41</td>
<td>0</td>
<td>8</td>
<td>33</td>
<td>0.00001223</td>
</tr>
<tr>
<td>8</td>
<td>33</td>
<td>0</td>
<td>8</td>
<td>25</td>
<td>0.00001148</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>0</td>
<td>8</td>
<td>17</td>
<td>0.00001065</td>
</tr>
<tr>
<td>10</td>
<td>17</td>
<td>0</td>
<td>8</td>
<td>9</td>
<td>0.00001117</td>
</tr>
</tbody>
</table>
Figure 53: Distributed Reorganization - Complete Knowledge with decrease in organization size
4.6 Distributed Control Transition Algorithm - Local Knowledge

The Distributed Control Transition Algorithm acts as a peer to peer system where all role assignments are negotiated for each transition. Each agent negotiates with all other affected agents to play the role for which it is most capably suited. With this algorithm, the question must be asked, "What does one agent know about all of the others?". With this algorithm, the agents know nothing about each other, except that they exist in the same organization and have the capacity to communicate effectively and reliably.

As with the Complete Knowledge algorithms, the local knowledge distributed control algorithms allow any agent $\in AGENT$ to receive a transition property. The distributed model takes the agent who receives the transition property $\phi$ to be the $agent_{organizer}$ for this specific transition. The $agent \in AGENT$ acting as the $agent_{organizer}$ can change each transition.

4.6.1 Distributed Control Initial Organization - Local Knowledge

The Distributed Control Initial Organization Algorithm is similar to the Central Control Initial Organization Algorithm with the exception that the central control algorithm has a designated leader, $agent_{leader}$, whereas the distributed control algorithm has an arbitrary agent, $agent_{organizer} \in AGENT$ which computes the initial state of the organization and then send the required information to each agent.

With a distributed knowledge concept, the $agent_{organizer}$ has two options. The first option is to send the entire computed organization core to each agent in the organization, as with the Distributed Control Complete Knowledge Algorithm. This method gives the appearance of perfect information, but is computationally more expensive and redundant. The second option is to send only the assignments to each agent, which is the idea of Distributed Control Local Knowledge. In this case $a_1$ acts as the $agent_{organizer}$. Fig. 54, shows the basic premise of the Distributed Control Complete Knowledge Initial Organization.

Algorithm 19, the Compute Initial Distributed Organization Algorithm computes the initial
Figure 54: Distributed Control Local Knowledge Initial Organization

distributed organization. The initial organization is computed much in the same way as Algorithm 16, the Distributed Control Complete Knowledge Initial Organization Algorithm. At the end, what is sent to each agent ∈ AGENT differs from the complete information algorithm. Whereas Algorithm 16 sends a complete copy of the organization state, Algorithm 19 only sends the specific assignments ASN_{agent} for the agent and its capabilities C_{agent}. With this algorithm, it is important to consider what each agent ∈ AGENT has knowledge of before and after initial organization.

Prior to initial organization, each agent has only the knowledge of their own capabilities, C_{agent} and agent_{organizer} has complete organization knowledge. After initial organization, each agent ∈ AGENT has complete local knowledge, meaning ASN_{agent} and updated C_{agent}. The initial agent_{organizer} retains knowledge of G_{int} and r_{root}, but deletes all assignments except its own.

Algorithm 20, Compute Initial Distributed Local Organization Algorithm first assembles the goal tree then compute the assignments for each g ∈ G_{leaf}, computing the OrgScore as each assigned relationship is determined.
Algorithm 19 Distributed Control Initial Local Organization Algorithm

Require: $\exists G, R, L, C, A$

Require: $\exists ACH, REQ, SUB, CON, POS, CAP, ASN$

Require: $|G| > 0, |R| > 0, |C| > 0, |A| > 0$

Require: $|ACH| > 0, |REQ| > 0, |POS| > 0$

Require: $|CAP| = 0, |ASN| = 0$

1: INPUT : $G, R, C, A$

2: INPUT : $ACH, REQ, SUB, CON, POS, CAP, ASN$

3: INPUT : agent\text{\_\textit{organizer}}

4: OUTPUT : organization\text{\_\textsubscript{1}}, the initial state of the central control organization

5: $ORG = \text{ComputeInitialDistCompleteOrganization}(G, R, C, A, ACH, REQ, POS, CAP, SUB)$

6: if $ASN_{ORG} \neq \text{NULL}$ then

7: for all $agent \in A\text{\textit{G\textsubscript{0}}}$ do

8: $\text{Communicate}(agent, ASN_{agent}, C_{agent})$

9: end for

10: end if

Algorithm 20 Compute Initial Distributed Local Organization Algorithm

1: \textit{PROCEDURE}\text{\textit{Com}}pute\text{\textit{Initial}}\text{\textit{Distributed}}\text{\textit{Organization}}$(G, R, C, A, ACH, REQ, POS, CAP, SUB)$

2: OrgScore = 0

3: AssembleGoalTree($G, SUB$)

4: for all $g \in G_{\text{leaf}}$ do

5: $assignment_g = \text{FindAssignment}(g)$

6: $ASN \leftarrow ASN \cup \{assignment_g\}$

7: $OrgScore = OrgScore + assignment_g.score$

8: end for

9: return $ORG$
### 4.6.2 Evaluation of Distributed Control Initial Organization - Local Knowledge

Table 14: Initial Organization

<table>
<thead>
<tr>
<th>Initial Organization</th>
<th>Objects size</th>
<th>Relationships size</th>
<th>Org size</th>
<th>Time$_{avg}$(1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>0.00068014</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>17</td>
<td>0.00065087</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>12</td>
<td>25</td>
<td>0.00072368</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>18</td>
<td>33</td>
<td>0.00089832</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>20</td>
<td>41</td>
<td>0.00105152</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>25</td>
<td>46</td>
<td>0.00109230</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>40</td>
<td>61</td>
<td>0.00098608</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>45</td>
<td>66</td>
<td>0.00109100</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>60</td>
<td>81</td>
<td>0.00101394</td>
</tr>
<tr>
<td>10</td>
<td>21</td>
<td>65</td>
<td>86</td>
<td>0.00106512</td>
</tr>
</tbody>
</table>

Table 14 and Fig. 55 show the tabular and graphical representation of the local knowledge initial organization. As the initial organization grows, the time to compute the initial organization also grows until 6. From 7 on there are reductions in the time to compute at 7, then an increase at 8 and the a decrease again at 9. The overall trend at this section looks as though the empirical data may be problematic at points 5 and 6 where the compute time rises more dramatically than previous or after.
Figure 55: Distributed Initial Organization
4.6.3 Distributed Control Reorganization - Local Knowledge

In the Distributed Control Initial Organization - Local Knowledge, the agent organizer, \( \text{agent}_{\text{organizer}} \in \text{AGENT} \), computes the initial organization instance and then sends the assignments, \( \text{ASN}_{\text{agent}} \), to each other agent. The reorganization begins with a set of \( \text{agent} \in \text{AGENT} \), where none are designated as the \( \text{agent}_{\text{organizer}} \). Each \( \text{agent} \) has only a local knowledge of its assignments, \( \text{ASN}_{\text{agent}} \). The assignment contains the goal, role, and capability for each \( \text{agent} \in \text{AGENT} \).

When a transition property, \( \phi \), is received by the organization, it is received by a single agent, which then becomes the \( \text{agent}_{\text{organizer}} \) for that specific transition. A new \( \text{agent}_{\text{organizer}} \) can be set for each new \( \phi \), although this is dependent on which \( \text{agent} \in \text{AGENT} \) perceives the \( \phi \) as a signal to transition.

The resolution of which \( a \in A \) plays \( r \in R \) depends on a series of conversations between two agents \( a_i \) and \( a_k \). The first message is a proposal. The sending agent proposes it plays a role to the receiving agent. The receiving agent can either rebut or accept the proposal. For example, if an agent \( \text{agent}_i \) proposes it plays a role \( r \) to agent \( \text{agent}_k \). If \( \text{capable}(\text{agent}_i, r) \geq \text{capable}(\text{agent}_k, r) \), then \( \text{agent}_k \) sends an accept message. If \( \text{capable}(\text{agent}_i, r) < \text{capable}(\text{agent}_k, r) \), then \( \text{agent}_k \) sends a rebut message. The proposal process determines which agent is more capable of playing a role \( r \in R \). Accept indicates the other agent agrees the proposing agent is more capable, whereas the rebuttal indicates the receiving agent is more capable. Fig. 56, shows the basic proposal diagram.

\[\text{Figure 56: Proposal Process with Distributed Agents}\]
Fig. 57, shows the basic premise of the Distributed Control Complete Knowledge reorganization. An initiating agent, $a_i$, proposes to a another agent, $a_k$ that it is better at playing a role, than is the agent being proposed to. The second agent takes that proposal and either accept the proposal, acknowledging $a_i$ is better or rebut the proposal indicating that it is better than $a_i$. Since the agents do not have any knowledge of each others capabilities, the proposal and accept/rebut scenario occurs for each pair of agents.

Algorithm 21 Distributed Local Reorganization Algorithm

| Require: $\exists \text{organization}_n$ |
| 1: INPUT : $\text{organization}_n$, an organization state |
| 2: INPUT : $\text{organization}$ property $\phi$, the property used to transition |
| 3: INPUT : $\text{agentorganizer}$, agent receiving $\phi$ |
| 4: OUTPUT : $\text{organization}_{n+1}$, a new state of the organization |
| 5: $\text{PropagatePhi}(\phi)$ |
| 6: $\text{CheckEachAgent}()$ |
| 7: $\text{Proposal(agentorganizer)}$ |
| 8: $\text{ComputeDistPhi}()$ |
| 9: $\text{CheckAssignments}()$ |
| 10: return $\text{organization}_{n+1}$ |

Algorithm 21, the Distributed Local Reorganization Algorithm initiates the transition pro-
cess. It requires an agent_{organizer}, which is an agent that receives the \( \phi \). The algorithm then calls a set of algorithms to PropagatePhi, CheckEachAgent, Proposal, ComputeDistPhi and CheckAssignments which completes the distributed local reorganization.

Algorithm 22, the Propagate Phi algorithm starts with the initial loop adding new agents to the organization. In this case, this must occur first as then each agent \( \in AGENT \) allocates the new predicates, the special case is when a new agent is added. A new agent does not appear in any of the other agent \( \in AGENT \) so a new agent is created.

Algorithm 23, CheckEachAgent, checks to see if each agent has sufficient organization objects to reorganize, as a precondition, otherwise the reorganization terminates.

Algorithm 24, the Proposal process algorithm is the peer to peer negotiation to determine which agent is best to play a specific \( r \in R \) to satisfy a specific \( g \in G_{leaf} \). Whichever agent, proposer or receiver, is better, takes on the role proposed.

Algorithm 25, ComputeDistPhi computes the assignments for each agent \( \in AGENT \) throughout the organization. All negotiations begin with a proposal initiated by the agent_{organizer} who is the recipient of the \( \phi \).

Algorithm 26, CheckAssignments, checks to see if each agent has a computable, valid assignment and can accomplish its goals. This is required as the goals are no longer shared between agents.
Algorithm 22 Propagate Phi

1: PROCEDURE PropagatePhi(\(\phi\))

2: for \(\forall \phi_i \in \phi\) do

3: if \(\phi_i\) is A then

4: \(AGENT \leftarrow AGEN + \{\phi_i\}\)

5: end if

6: end for

7: for \(\forall agent \in AGEN\) do

8: for \(\forall \phi_i \in \phi\) do

9: if \(\phi_i\) is object then

10: if \(\phi_i\) is \(G_{leaf}\) then

11: \(agent_{Goal_{leaf}} \leftarrow agent_{Goal_{leaf}} + \{\phi_i\}\)

12: else if \(\phi_i\) is Role then

13: \(agent_r \leftarrow agent_r + \{\phi_i\}\)

14: else if \(\phi_i\) is Capability then

15: \(agent_c \leftarrow agent_c + \{\phi_i\}\)

16: end if

17: end if

18: end for

19: end for

20: for \(\forall agent \in AGEN\) do

21: if \(\phi_i\) is relationship then

22: if \(\phi_i\) is POS then

23: if \(possess_{\phi}(c_agent, agent)\) then

24: \(agent_{possess} \leftarrow agent_{possess} + \{\phi_i\}\)

25: end if

26: else if \(\phi_i\) is REQ then

27: if \(requires_{\phi}(role, c_agent)\) then

28: \(agent_{requires} \leftarrow agent_{requires} + \{\phi_i\}\)

29: end if

30: else if \(\phi_i\) is ACH then

31: if \(achieves_{\phi}(role_{agent}, goal_{agent})\) then

32: \(agent_{achieves} \leftarrow agent_{achieves} + \{\phi_i\}\)

33: end if

34: end if

35: end if

36: end if

37: end for

38: removePhi(agent)

39: end for
Algorithm 23 Check Each Agent for objects
1: PROCEDURE CheckEachAgent()
2: for all agent ∈ AGENT do
3: if agent_G = ∅ or agent_R = ∅ or agent_C = ∅ or agent_A = ∅ then
4: exit organization
5: end if
6: end for

Algorithm 24 Proposal
1: PROCEDURE Proposal(agent)
2: for ∀ r ∈ R do
3: mostCapable = ∅
4: done = ∅
5: for ∀ agent_k ∈ AGENT do
6: if capable(agent_i, r) ≥ capable(agent_k, r) then
7: accept()
8: mostCapable = agent_i
9: done = done ∪ {agent_k}
10: else if capable(agent_i, r) < capable(agent_k, r) then
11: rebut()
12: mostCapable = agent_k
13: done = done ∪ {agent_k}
14: end if
15: end for
16: end for
17: return capable(mostCapable, r)

Algorithm 25 Compute Distributed Phi
1: PROCEDURE ComputeDistPhi()
2: for all agent ∈ AGENT do
3: GoalsToComputeList = ∅
4: for all φ_i ∈ φ_agent do
5: goal = FindLeafGoal(φ_i)
6: GoalsToComputeList = GoalsToComputeList ∪ {goal}
7: end for
8: ComputePhi(GoalsToComputeList)
9: end for
Algorithm 26 Check Assignments

1: PROCEDURE CheckAssignments()
2: for all agent ∈ AGENT do
3:   if ASN_{agent} = ∅ then
4:     end organization
5:   end if
6: end for
4.6.4 Evaluation of Distributed Control Reorganization - Local Knowledge

Table 15: Distributed Control Local Knowledge Reorganization - Add

<table>
<thead>
<tr>
<th>Organization</th>
<th>Org_{size_n}</th>
<th>φ_{add}</th>
<th>φ_{delete}</th>
<th>Org_{size_{n+1}}</th>
<th>Time_{avg}(1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>8</td>
<td>0</td>
<td>17</td>
<td>0.00007098</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>8</td>
<td>0</td>
<td>25</td>
<td>0.00002072</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>8</td>
<td>0</td>
<td>33</td>
<td>0.00002186</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
<td>8</td>
<td>0</td>
<td>41</td>
<td>0.00006072</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
<td>5</td>
<td>0</td>
<td>46</td>
<td>0.00002985</td>
</tr>
<tr>
<td>6</td>
<td>46</td>
<td>15</td>
<td>0</td>
<td>61</td>
<td>0.00003593</td>
</tr>
<tr>
<td>7</td>
<td>61</td>
<td>5</td>
<td>0</td>
<td>66</td>
<td>0.00001536</td>
</tr>
<tr>
<td>8</td>
<td>66</td>
<td>15</td>
<td>0</td>
<td>81</td>
<td>0.00003397</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>5</td>
<td>0</td>
<td>86</td>
<td>0.00001355</td>
</tr>
<tr>
<td>10</td>
<td>86</td>
<td>5</td>
<td>0</td>
<td>91</td>
<td>0.00001340</td>
</tr>
</tbody>
</table>

Table 15 and Fig. 58 show that there is no consistency of reorganization times for adding to the organization at a micro scale. While the expectation is the heightened time with the addition of large groups of new relationships, the distributed model does not follow the other algorithms in an add format.

Figure 58: Distributed Reorganization Local with increase to organization size

Table 16 and Fig. 59 indicate that for a local reorganization application the premise holds that
the longest times to reorganize appear with the addition of a large group of relationships.

<table>
<thead>
<tr>
<th>Organization</th>
<th>OrgSize</th>
<th>φ_add</th>
<th>φ_delete</th>
<th>OrgSize</th>
<th>Time_{avg}(1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>91</td>
<td>0</td>
<td>5</td>
<td>86</td>
<td>0.00003738</td>
</tr>
<tr>
<td>2</td>
<td>86</td>
<td>0</td>
<td>5</td>
<td>81</td>
<td>0.00001473</td>
</tr>
<tr>
<td>3</td>
<td>81</td>
<td>0</td>
<td>15</td>
<td>66</td>
<td>0.00003541</td>
</tr>
<tr>
<td>4</td>
<td>66</td>
<td>0</td>
<td>5</td>
<td>61</td>
<td>0.00001096</td>
</tr>
<tr>
<td>5</td>
<td>61</td>
<td>0</td>
<td>15</td>
<td>46</td>
<td>0.00003031</td>
</tr>
<tr>
<td>6</td>
<td>46</td>
<td>0</td>
<td>5</td>
<td>41</td>
<td>0.00001078</td>
</tr>
<tr>
<td>7</td>
<td>41</td>
<td>0</td>
<td>8</td>
<td>33</td>
<td>0.00001485</td>
</tr>
<tr>
<td>8</td>
<td>33</td>
<td>0</td>
<td>8</td>
<td>25</td>
<td>0.00001358</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>0</td>
<td>8</td>
<td>17</td>
<td>0.00001242</td>
</tr>
<tr>
<td>10</td>
<td>17</td>
<td>0</td>
<td>8</td>
<td>9</td>
<td>0.00001362</td>
</tr>
</tbody>
</table>
Figure 59: Distributed Reorganization Local with decrease to organization size
4.7 Segmented Control Transition Algorithm

Segmented Control Transition Algorithm is a combination of the central and distributed models. Often, when an organization needs to reorganize, the entire organization does not need to change, but only a small subset of the organization needs to be affected. For example, if a single, but critical programmer leaves a 10,000 person company, the entire company is not reorganized, but only the immediate team and maybe some surrounding teams. Of all control algorithms, the segmented scheme is the most complex, but more realistic model for larger and more useful organization structures and domain problems.

4.7.1 Segmented Control Initial Organization

The segmented control initial organization is computationally more expensive than a generic initial organization due to the necessity to segment the organization. The intuition is that the organization requires less computation do to only having to recompute a segment when a $\phi$ is encountered. The general segmentation starts with all $g \in G_{leaf}$, split into segments. Then, install each leaf as a segment, until proven otherwise which results initially as $\{N_{segments} \mid N = |G_{leaf}| \}$. Then, go through each level of the structure and resegment based upon the achieves, requires and possesses relationships as described by segment discovery process shown in Fig. 60. There are 5 initial segments based upon $|Goal_{leaf}|$, then achieves relationships in the role segmentation step reduces the number of segments to 3. Capability segmentation further reduces the number to 2 segments. Finally, the agent segmentation step does not reduce the number of segments.

Algorithm 27, Segmented Initial Organization Algorithm, is an extension of the Generic Initial Organization algorithm. The key differences are the requirement to segment the organization and then to compute each segment, instead a single organization instance. The organization segmentation is initiated in the call to SegmentOrganization() at line 8. This function returns, $SEG$, a set of segments. This must be done after the initial AssembleGoalTree, because the $|Goal_{leaf}|$ must be known prior to segmentation. To compute each segment, the algorithm loops through each
$seg_n \in SEG$. The computation for each segment is done the same as for a complete organization.

The core premise of Algorithm 28, Segment Organization Algorithm, is to maximize segmentation. This is consistent with keeping computation minimized with the fewest objects and relationships. Maximization of segments intuitively minimizes the average size of each segment in terms of object and relationships. Minimization of size reduces the computation time for each segment.

The first responsibility of Algorithm 28 is to decompose each $g \in Goal_{leaf}$ into a segment, which occurs in lines 4 - 8. After that, the algorithm calls, in order, the functions to resegment the organization by roles $ComputeRoleSegmentation(SEG)$, capabilities $ComputeCapabilitySegmentation(SEG)$ and agents $ComputeAgentSegmentation(SEG)$ in that order. The algorithm invokes Algorithm
Algorithm 27 Segmented Initial Organization Algorithm

Require: ∃G, R, L, C, A

Require: ∃ACH, REQ, SUB, CON, POS, CAP, ASN, SEG

Require: |G| > 0, |R| > 0, |C| > 0, |A| > 0

Require: |ACH| > 0, |REQ| > 0, |POS| > 0

Require: |CAP| = 0, |ASN| = 0, |SEG| = 0

1: INPUT : G, R, C
2: INPUT : ACH, REQ, SUB, CON
3: INPUT : A
4: INPUT : POS, CAP, ASN, SEG
5: OUTPUT : organization_1, the initial state of the organization
6: OrgScore = 0
7: AssembleGoalTree(G, SUB)
8: SEG = SegmentOrganization()
9: for all seg_n ∈ SEG do
10:   OrgScoreSegment_n = ComputeSegment(seg_n)
11:   if OrgScoreSegment_n = 0 then
12:     end organization
13:   else
14:     OrgScore = OrgScore + OrgScoreSegment_n
15:   end if
16: end for

29, Algorithm 30 and 31.

Algorithm 29, Compute Role Segmentation is to resegment the organization based upon the roles and their achieves relations with the Goal_leaf set. Algorithms 30, ComputeCapabilitySegmentation(SEG), and 31, ComputeAgentSegmentation(SEG), are similar in steps. Each looks at the specific object in which they are referring, roles r, capabilities c or agents a, and must go through the set for each object element. If each only has a single goal in its specific goalSET found through using the trivial findLeafGoal(object) function, it gets the segment using getSegment{g ∈ G} and adds the object to that segment seg_i. If |goalSET| > 1, then the algorithm goes through each goal in goalSET and determines if the segments need to be reduced.

Algorithm 32, Compute Segment Algorithm, has the same function as the computation of an organization. The operation is computed only on a segment of an organization instead of the entire organization. The allows the segmentation scheme to process computations in parallel, thereby
Algorithm 28 Segment Organization Algorithm

1: PROCEDURE SegmentOrganization()
2: i=0
3: SEG = ∅
4: for all $g \in G_{leaf}$ do
5:   $seg_i = \{g\}$
6:   SEG = SEG $\cup\{seg_i\}$
7:   $i = i + 1$
8: end for
9: SEG = ComputeRoleSegmentation(SEG)
10: SEG = ComputeCapabilitySegmentation(SEG)
11: SEG = ComputeAgentSegmentation(SEG)
12: return SEG

reducing the time to recompute an organization.
Algorithm 29 Compute Role Segmentation

1: PROEDURE ComputeRoleSegmentation(SEG)
2: for all \( r \in R \) do
3: \( \text{goalSET} = \text{findLeafGoal}(r) \)
4: if \( |\text{goalSET}| = 1 \) then
5: \( \text{seg}_i = \text{getSegment}(g \in \text{goalSET}) \)
6: \( \text{seg}_i = \text{seg}_i \cup \{r\} \)
7: else if \( |\text{goalSET}| > 1 \) then
8: \( \text{saveSEG} = \emptyset \)
9: for all \( g \in \text{goalSET} \) do
10: \( \text{seg}_i = \text{getSegment}(g) \)
11: if \( \text{saveSEG} = \text{seg}_i \) then
12: \( \text{seg}_i = \text{seg}_i \cup \{r\} \)
13: else
14: \( \text{seg}_{\text{new}} = \{\text{seg}_i\} \cup \{\text{saveSEG}\} \)
15: \( \text{SEG} = \text{SEG} - \{\text{seg}_i\} \)
16: \( \text{SEG} = \text{SEG} - \{\text{saveSEG}\} \)
17: \( \text{SEG} = \text{SEG} \cup \{\text{seg}_{\text{new}}\} \)
18: \( \text{saveSEG} = \text{seg}_{\text{new}} \)
19: end if
20: end for
21: end if
22: end for
23: return SEG
Algorithm 30 Compute Capability Segmentation

1: PROCEDURE ComputeCapabilitySegmentation(SEG)
2: for all $c \in C$ do
3: \hspace{1em} $goalSET = \text{findLeafGoal}(c)$
4: if $|goalSET| = 1$ then
5: \hspace{2em} $seg_i = \text{getSegment}(g \in goalSET)$
6: \hspace{2em} $seg_i = seg_i \cup \{r\}$
7: else if $|goalSET| > 1$ then
8: \hspace{2em} $saveSEG = \emptyset$
9: \hspace{3em} for all $g \in goalSET$ do
10: \hspace{4em} $seg_i = \text{getSegment}(g)$
11: \hspace{4em} if $saveSEG = seg_i$ then
12: \hspace{5em} $seg_i = seg_i \cup \{c\}$
13: \hspace{4em} else
14: \hspace{5em} $seg_{new} = \{seg_i\} \cup \{saveSEG\}$
15: \hspace{4em} $SEG = SEG \setminus \{seg_i\}$
16: \hspace{4em} $SEG = SEG \cup \{seg_{new}\}$
17: \hspace{4em} $saveSEG = seg_{new}$
18: \hspace{3em} end if
19: \hspace{2em} end for
20: end if
21: end for
22: return $SEG$
Algorithm 31 Compute Agent Segmentation

1: PROCEDURE ComputeAgentSegmentation(SEG)
2: for all $a \in A$ do
3:    $goalSET = findLeafGoal(A)$
4:    if $| goalSET | = 1$ then
5:        $seg_i = getSegment(g \in goalSET)$
6:        $seg_i = seg_i \cup \{ r \}$
7:    else if $| goalSET | > 1$ then
8:        $saveSEG = \emptyset$
9:        for all $g \in goalSET$ do
10:           $seg_i = getSegment(g)$
11:           if $saveSEG = seg_i$ then
12:              $seg_i = seg_i \cup \{ a \}$
13:           else
14:              $seg_{new} = \{ seg_i \} \cup \{ saveSEG \}$
15:              $SEG = SEG - \{ seg_i \}$
16:              $SEG = SEG \cup \{ seg_{new} \}$
17:              $saveSEG = seg_{new}$
18:           end if
19:        end for
20:    end if
21: end for
22: return $SEG$

Algorithm 32 Compute Segment Algorithm

1: PROCEDURE ComputeSegment(seg_n)
2: for all $g \in G_{leaf}$ do
3:    assignment_g = FindAssignment(g)
4:    $ASN \leftarrow ASN \cup \{ assignment_g \}$
5:    $OrgScore = OrgScore + assignment_g.score$
6: end for
7: if $ASN = \emptyset$ then
8:    $OrgScore = 0$
9:    return NULL
10: else
11:    return $ORGSEG$
12: end if
### 4.7.2 Evaluation of Segmented Control Initial Organization

Table 17: Initial Organization

<table>
<thead>
<tr>
<th>Initial Organization</th>
<th>Objects size</th>
<th>Relationships size</th>
<th>Org size</th>
<th>Time(_{avg}) (1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>0.00056140</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>17</td>
<td>0.00061053</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>12</td>
<td>25</td>
<td>0.00079199</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>18</td>
<td>33</td>
<td>0.00101445</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>20</td>
<td>41</td>
<td>0.00099428</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>25</td>
<td>46</td>
<td>0.00119437</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>40</td>
<td>61</td>
<td>0.00108820</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>45</td>
<td>66</td>
<td>0.00121285</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>60</td>
<td>81</td>
<td>0.00098753</td>
</tr>
<tr>
<td>10</td>
<td>21</td>
<td>65</td>
<td>86</td>
<td>0.00125055</td>
</tr>
</tbody>
</table>

Table 17 and Fig. 61 describe the data of applying segmentation to initial organization. The escalation of compute times is increased as the organization size increases dramatically from 1 to 4 as the \(\text{Relationships size}\) grows at a higher rate. At 5, the \(\text{Relationships size}\) increases by 2 but \(\text{Org time}\) decreases slightly. From 5 on, the times toggle with a large decrease at 9, which is an anomaly due to the increase in \(\text{Relationships size}\) from 66 to 81.
Figure 61: Segmented Initial Organization
4.7.3 Segmented Control Reorganization

Once the initial organization is segmented, reorganization is computationally similar to segmented organization, although segmentations can be performed in parallel.

Algorithm 33, Segmented Reorganization Algorithm

| Require: | ∃ organization
| 1: INPUT : organization, an organization state | | |
| 2: INPUT : organization property φ, the property used to transition | | |
| 3: OUTPUT : organization, a new state of the organization | | |
| 4: for seg ∈ SEG do | | |
| 5: φseg = SegmentPhi(φ) | | |
| 6: GoalsToCompute = {} | | |
| 7: GoalsToCompute = InstallSegPhi(φseg) | | |
| 8: if Gseg = ∅ or Rseg = ∅ or Cseg = ∅ or Aseg = ∅ then | | |
| 9: exit organization | | |
| 10: else | | |
| 11: ComputeSegmentedPhi(GoalsToCompute) | | |
| 12: end if | | |
| 13: if ASNseg = ∅ then | | |
| 14: end organization | | |
| 15: end if | | |
| 16: end for | | |
| 17: return organization

Algorithm 33, Segmented Reorganization Algorithm, requires an organization exists, takes a property φ prior to execution and results in a new organization state. The extended function is that it only reorganizes segments affected by new objects and relationships in φ. The salient step of the algorithm, is the SegmentPhi(φ), Algorithm 34 which segments the transition property to apply to the current segments of the organization. The second key line is the call of InstallSegPhi(φseg), installing the φseg segmented predicates created by SegmentPhi(φ).

Algorithm 34, SegmentPhi, splits the initial transition property φ into specific simple or complex predicates based on the elements of each segment.

Algorithm 35, InstallSegPhi, first calls the function to parse the transition predicates and add them to the organization. Then, the GoalsToComputeList is initialized and later tracks all goal.
Algorithm 34 Segment the Transition Property

1: PROCEDURE SegmentPhi(φ)
2: for φᵢ ∈ φ do
3:     if φᵢ is φ_object then
4:         if φ_objectᵢ ∈ segᵢ then
5:             φ segᵢ ← φ segᵢ ∪ {φ_objectᵢ}
6:     end if
7:     else if φᵢ is φ_relationship then
8:         if φ_relationshipᵢ ∈ segᵢ then
9:             φ segᵢ ← φ segᵢ ∪ {φ_relationshipᵢ}
10:    end if
11: end if
12: end for
13: return φ seg₁ ... φ segₙ

Algorithm 35 Install Organization Segment Predicates

1: PROCEDURE InstallSegPhi(φ seg)
2: parsePhi(φ seg)
3: GoalsToComputeList = ∅
4: for φᵢ ∈ φ seg do
5:     goal = FindLeafGoal(φᵢ)
6:     GoalsToComputeList ← GoalsToComputeList ∪ {goal}
7: end for
8: return GoalsToComputeList

whose structures need to be recomputed. Finding the goal leaf for each φᵢ is trivial, so there is no algorithm included detailing the steps. For each simple predicate φᵢ ∈ φ, the goal leaf is added to the GoalsToComputeList set. In the end, GoalsToComputeList is returned.

Algorithm 36, Compute Segmented Phi examines the | φ_relationship | and | φ_object | to determine how to proceed. If both | φ_relationship | = 0 and | φ_object | = 0, there is no reason to continue as the φ is trivial. If | φ_relationship | > 0 and | φ_object | = 0, then new relationships are added to existing objects and a recomputation takes place. The same situation occurs if | φ_relationship | > 0 and | φ_object | > 0. Even though a φ where | φ_relationship | = 0 and | φ_object | > 0 is valid, there is no reason to compute, as the relationships between objects do not change.

Table 18 and Fig. 62 show the data for evaluation of segmented reorganization where objects
Algorithm 36 Compute the new organization segment

1: PROCEDURE ComputeSegmentedPhi(GoalsToCompute)
2: if $|\phi_{\text{relationship}}| > 0$ and $|\phi_{\text{object}}| = 0$ then
3: for $g \in \text{GoalsToCompute}$ do
4:  $\text{assignment}_g = \text{FindAssignment}(g)$
5:  $\text{ASN} \leftarrow \text{ASN} \cup \{\text{assignment}_g\}$
6:  $\text{GoalsToCompute} \leftarrow \text{GoalsToCompute} - \{g\}$
7: end for
8: else if $|\phi_{\text{relationship}}| > 0$ and $|\phi_{\text{object}}| > 0$ then
9: for $g \in \text{GoalsToCompute}$ do
10:  $\text{assignment}_g = \text{FindAssignment}(g)$
11:  $\text{ASN} \leftarrow \text{ASN} \cup \{\text{assignment}_g\}$
12:  $\text{GoalsToCompute} \leftarrow \text{GoalsToCompute} - \{g\}$
13: end for
14: else if $|\phi_{\text{relationship}}| = 0$ and $|\phi_{\text{object}}| > 0$ then
15: do not recompute
16: end if

and relationships are added to the organization. In this case, the initial time 1 is high as there is a lot of computation with segmentation. Transitions 6 and 8 are the largest compute times which agrees with previous results which show that transitions with high numbers of relationships increase the time to transition.

Table 19 and Fig. 63 shows the data for the segmented reorganization where objects and relationships are deleted from a large initial organization and gets smaller over its life. Consistently,

<table>
<thead>
<tr>
<th>OrganizationState</th>
<th>Orgsize$_n$</th>
<th>$\phi_{\text{add}}$</th>
<th>$\phi_{\text{delete}}$</th>
<th>Orgsize$_{n+1}$</th>
<th>Time$_{avg}(1000)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>8</td>
<td>0</td>
<td>17</td>
<td>0.00009983</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>8</td>
<td>0</td>
<td>25</td>
<td>0.00002794</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>8</td>
<td>0</td>
<td>33</td>
<td>0.00001799</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
<td>8</td>
<td>0</td>
<td>41</td>
<td>0.00001393</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
<td>5</td>
<td>0</td>
<td>46</td>
<td>0.00001445</td>
</tr>
<tr>
<td>6</td>
<td>46</td>
<td>15</td>
<td>0</td>
<td>61</td>
<td>0.00007589</td>
</tr>
<tr>
<td>7</td>
<td>61</td>
<td>5</td>
<td>0</td>
<td>66</td>
<td>0.00001349</td>
</tr>
<tr>
<td>8</td>
<td>66</td>
<td>15</td>
<td>0</td>
<td>81</td>
<td>0.00003298</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>5</td>
<td>0</td>
<td>86</td>
<td>0.00001318</td>
</tr>
<tr>
<td>10</td>
<td>86</td>
<td>5</td>
<td>0</td>
<td>91</td>
<td>0.00001399</td>
</tr>
</tbody>
</table>
Figure 62: Segmented Reorganization with increase to organization size

The highest times to transition are in reorganizations 3 and 5 which reflect a high number of relationships.
<table>
<thead>
<tr>
<th>OrganizationState</th>
<th>Orgsizeₙ</th>
<th>φadd</th>
<th>φdelete</th>
<th>Orgsizeₙ₊₁</th>
<th>Timeavg(1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>91</td>
<td>0</td>
<td>5</td>
<td>86</td>
<td>0.000002260</td>
</tr>
<tr>
<td>2</td>
<td>86</td>
<td>0</td>
<td>5</td>
<td>81</td>
<td>0.00001195</td>
</tr>
<tr>
<td>3</td>
<td>81</td>
<td>0</td>
<td>15</td>
<td>66</td>
<td>0.00002062</td>
</tr>
<tr>
<td>4</td>
<td>66</td>
<td>0</td>
<td>5</td>
<td>61</td>
<td>0.00001023</td>
</tr>
<tr>
<td>5</td>
<td>61</td>
<td>0</td>
<td>15</td>
<td>46</td>
<td>0.000001982</td>
</tr>
<tr>
<td>6</td>
<td>46</td>
<td>0</td>
<td>5</td>
<td>41</td>
<td>0.00000998</td>
</tr>
<tr>
<td>7</td>
<td>41</td>
<td>0</td>
<td>8</td>
<td>33</td>
<td>0.00001225</td>
</tr>
<tr>
<td>8</td>
<td>33</td>
<td>0</td>
<td>8</td>
<td>25</td>
<td>0.00001143</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>0</td>
<td>8</td>
<td>17</td>
<td>0.00001184</td>
</tr>
<tr>
<td>10</td>
<td>17</td>
<td>0</td>
<td>8</td>
<td>9</td>
<td>0.00001105</td>
</tr>
</tbody>
</table>

Figure 63: Segmented Reorganization with decrease to organization size
4.8 Summary

The algorithms to initially organize and reorganize vary over a number of features. Table 20 summarizes the main features that differentiate the algorithms. Each algorithm has specific strengths and weaknesses, depending on task domain. The *Generic* algorithms are not listed as they are the basis for all other algorithms.

<table>
<thead>
<tr>
<th>Algorithm Model</th>
<th>Knowledge</th>
<th>Messaging</th>
<th>$\phi$ received by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central</td>
<td>Complete</td>
<td>$\text{agent}_{\text{leader}}$</td>
<td>$\text{agent}_{\text{leader}}$</td>
</tr>
<tr>
<td>DistComplete</td>
<td>Complete</td>
<td>$\text{agent}_{\text{organizer}}$</td>
<td>$\text{agent} \in \text{AGENT}$</td>
</tr>
<tr>
<td>DistLocal</td>
<td>Local</td>
<td>$\text{agent}_{\text{organizer}}$</td>
<td>$\text{agent} \in \text{AGENT}$</td>
</tr>
<tr>
<td>Segmented</td>
<td>Local/Complete</td>
<td>multiple</td>
<td>$\text{agent} \in \text{AGENT}_{\text{SEG}}$</td>
</tr>
</tbody>
</table>

Figure 64: Comparison of Initial Organization Control Times

A comparison of all initial organization algorithms is shown in Fig. 64. The key aspect of
comparing these algorithms on a micro-scale is the trend line each represents. While the trend line of $\text{ORG}_{\text{size}}, | \phi_{\text{add}} |$ or $| \phi_{\text{delete}} |$ is not large enough to apply theoretical analysis of each algorithm, it does show the growth over a trivial to small size organization progression.

Figure 65: Comparison of Reorganization Control Times - Increasing Organization Size

A comparison of all reorganization algorithms with additions is shown in Fig. 65. The trend lines for each are fairly consistent and indicate that when $| \phi_{\text{relationships}} |$ is high, the time to reorganize is significantly higher than if the reorganization is predicated on objects.

A comparison of all reorganization algorithms with deletes is shown in Fig. 66. As with the previous comparison for additions, the trend lines for deletion are fairly consistent and indicate that when $| \phi_{\text{relationships}} |$ is high, the time to reorganize is again significantly higher than if the reorganization is predicated on objects.

The results of the micro-analysis of organization change shows the impact of relationships versus
the impact of objects on reorganization time over $\phi$. This analysis was shown on a micro scale, as the details tend to disappear when large-scale organizations are transitioned. The difference in transition time using relationships and objects is a critical determinant.
5 Analysis and Validation

This chapter describes the analysis and validation of all algorithms. The first section describes the analysis of each initial organization and reorganization algorithm. The second section compares each initial organization and reorganization algorithm with all others over similar measures. The final section, validation, describes scenarios of realistic simulations and domain problems and how they can be modeled using the organization model and appropriate transition algorithms.

5.1 Algorithm Analysis

The algorithms are driven by the changes that affect the organizations on size, number of objects or relationships added or deleted and the various changes that are possible, over the life span of the organization. In this section, an empirical analysis is conducted for each algorithm.

This analysis sets a benchmark with which to compare future algorithms and extensions to these algorithms. While many have discussed reorganization in capability based organizations, there are no complete model with corresponding algorithms which formalize transition. All algorithms are compared using the same evaluation criteria with the growth rate used for comparison. This allows others to compare their algorithms, in future algorithms and proposals.

The benchmark also establishes the comparison of algorithm expectations. For instance, intuition dictates that a central control algorithm is less complex than a distributed or segmented, based on the number of messages exchanged. But for survivability and adaptability, the distributed algorithm provides a better solution, due to redundancy.

One interesting comparison is to explore if the algorithms differ in suitability based on the size and complexity of the organization. For example, will the central control algorithm be more efficient than the distributed control algorithm with a small, low-complexity organization structure. Conversely, a large, complex organization may show that the segmented algorithm is more appropriate.
The organization model structurally takes on the form of a graph for computational purposes. With that stated, clarity must be established in the perspective in which we approach the analysis of the algorithms. Traditionally, algorithm analysis, as shown Cormen et al. [16] uses a single variable to compute complexity. However, a differing perspective is presented by Howell [44], who proposes that analysis can be computed over several variables. Each approach was considered in this research. The main question is what is the relevant variable or variables to approach the analysis. The model has goals $G$, roles $R$, capabilities $C$ and agents $A$, each which can grow independently of the others. Each is also bound by a set of relationships to the organization objects. Based upon these variable factors of growth, typical asymptotic analysis cannot be utilized to set bounds for these algorithms. For this analysis, sparse and dense organizations are used to provide the basis for lower and upper bound approximations.

The main goal of the evaluation is to determine the implementation and scalability of the model to differing multiagent systems. While many systems are implemented on specific or trivial systems (less than 10 agents), the intent is to create a generic model that can be applied to any level of complexity with success. The model, transition formalisms and algorithms must be useful from a practical perspective.

As the organization model is fully implemented in Java, it can be evaluated to produce empirical results. The implementation of the organization model allows a deeper understanding of the real effort an organization goes through to form and the behavior it exhibits during its lifecycle.

A standard set of tests evaluate each algorithm. A comparison can be constructed based on these results. Some variables on which growth is measured are the number of agents, capabilities, roles, goals, and relationships between these entities. Changes to an organization effect the ability to transition, so the number of properties included in a change to the complexity of the transition, is measured.

The most complete and valuable evaluation is based upon the fully implemented model, such as the sensor organization validation scenario. This evaluation extends the basic set of empirical evaluations, at each level.
5.1.1 Sparse and Dense Organizations

For evaluation of organization, the difference between upper and lower bounds must be brought into specific relief. We further define \textit{sparse} and \textit{dense} organizations as the two extremes of evaluation. Fig. 67 shows the difference between a sparse and dense organization.

**Figure 67: Sparse and Dense Organizations**

\begin{align*}
\text{Definition: Sparse Organization} & \{ \forall g,r,q \mid \text{achieves}(x,g) = 1, | \text{requires}(r,c) | = 1, | \text{possesses}(a,c) | = 1 \mid g \in G_{\text{leaf}}, r \in R, a \in A, c \in C, x \in R - \{r\} \}\n
\text{Definition: Dense Organization} & \{ \forall g \forall r \exists \text{achieves}(r,g) \land \forall r \forall c \exists \text{requires}(r,c) \land \forall a \forall c \exists \text{possesses}(a,c) \mid g \in G_{\text{leaf}}, r \in R, c \in C, a \in A \}
\end{align*}

5.1.2 Analysis Experimental Setup

The complete algorithm set for each initial organization and reorganization is developed in \textit{Java jdk1.6.003} The evaluations were executed on Intel Core 2 Pentium CPU T7400 computer clocked to
2.16 GHz with 2 GB of main memory. For each algorithm, a standard computer environment was used, with a complete restart of the machine to reset the environment. Each algorithm is executed 10 complete times at each size and then the average time is used. The $ORG_{size}$ is dependent on whether the organization is SPARSE or DENSE. A SPARSE organization is much smaller than a DENSE organization, in terms of $ORG_{size}$, because the number of relationships grows at an exponential instead of a linear rate.

Table 21 shows a difference, for an organization, over a set of $ORG_{size}$ variants. These values are used to evaluate each of the initial organization algorithms. The growth rate of $ORG_{size}$ is a critical part of the evaluation. As in the micro-analysis of Chapter 4, the increase in relationships drives the increase in computational resources, leading to an increase in time to reorganize. Over the 25 simulations, the SPARSE organization varies from an $Object_{size} = [21 \ldots 501]$, where the SPARSE organization total $ORG_{size} = [41 \ldots 1001]$ and the DENSE organization total $ORG_{size} = [101 \ldots 47501]$. The corresponding times to organize vary from 0.01568137 seconds at $ORG_{size} = 1001$ for SPARSE and 1619.348813 seconds $ORG_{size} = 47501$ for DENSE as shown in Table 22.
<table>
<thead>
<tr>
<th>Simulation</th>
<th>$\text{Objects}_{\text{size}}$</th>
<th>$\text{Relationships}_{\text{size}}$</th>
<th>$\text{ORG}_{\text{size}}$</th>
<th>$\text{Relationships}_{\text{size}}$</th>
<th>$\text{ORG}_{\text{size}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>20</td>
<td>41</td>
<td>80</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>40</td>
<td>81</td>
<td>310</td>
<td>351</td>
</tr>
<tr>
<td>3</td>
<td>61</td>
<td>60</td>
<td>121</td>
<td>690</td>
<td>751</td>
</tr>
<tr>
<td>4</td>
<td>81</td>
<td>80</td>
<td>161</td>
<td>1220</td>
<td>1301</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>100</td>
<td>201</td>
<td>1900</td>
<td>2001</td>
</tr>
<tr>
<td>6</td>
<td>121</td>
<td>120</td>
<td>241</td>
<td>2730</td>
<td>2851</td>
</tr>
<tr>
<td>7</td>
<td>141</td>
<td>140</td>
<td>281</td>
<td>3710</td>
<td>3851</td>
</tr>
<tr>
<td>8</td>
<td>161</td>
<td>160</td>
<td>321</td>
<td>4840</td>
<td>5001</td>
</tr>
<tr>
<td>9</td>
<td>181</td>
<td>180</td>
<td>361</td>
<td>6120</td>
<td>6301</td>
</tr>
<tr>
<td>10</td>
<td>201</td>
<td>200</td>
<td>401</td>
<td>7550</td>
<td>7751</td>
</tr>
<tr>
<td>11</td>
<td>221</td>
<td>220</td>
<td>441</td>
<td>9130</td>
<td>9351</td>
</tr>
<tr>
<td>12</td>
<td>241</td>
<td>240</td>
<td>481</td>
<td>10860</td>
<td>11101</td>
</tr>
<tr>
<td>13</td>
<td>261</td>
<td>260</td>
<td>521</td>
<td>12740</td>
<td>13001</td>
</tr>
<tr>
<td>14</td>
<td>281</td>
<td>280</td>
<td>561</td>
<td>14770</td>
<td>15051</td>
</tr>
<tr>
<td>15</td>
<td>301</td>
<td>300</td>
<td>601</td>
<td>16950</td>
<td>17251</td>
</tr>
<tr>
<td>16</td>
<td>321</td>
<td>320</td>
<td>641</td>
<td>19280</td>
<td>19601</td>
</tr>
<tr>
<td>17</td>
<td>341</td>
<td>340</td>
<td>681</td>
<td>21760</td>
<td>22101</td>
</tr>
<tr>
<td>18</td>
<td>361</td>
<td>360</td>
<td>721</td>
<td>24390</td>
<td>24751</td>
</tr>
<tr>
<td>19</td>
<td>381</td>
<td>380</td>
<td>761</td>
<td>27170</td>
<td>27551</td>
</tr>
<tr>
<td>20</td>
<td>401</td>
<td>400</td>
<td>801</td>
<td>30100</td>
<td>30501</td>
</tr>
<tr>
<td>21</td>
<td>421</td>
<td>420</td>
<td>841</td>
<td>33180</td>
<td>33601</td>
</tr>
<tr>
<td>22</td>
<td>441</td>
<td>440</td>
<td>881</td>
<td>36410</td>
<td>36851</td>
</tr>
<tr>
<td>23</td>
<td>461</td>
<td>460</td>
<td>921</td>
<td>39790</td>
<td>40251</td>
</tr>
<tr>
<td>24</td>
<td>481</td>
<td>480</td>
<td>961</td>
<td>43320</td>
<td>43801</td>
</tr>
<tr>
<td>25</td>
<td>501</td>
<td>500</td>
<td>1001</td>
<td>47000</td>
<td>47501</td>
</tr>
</tbody>
</table>
### 5.1.3 Generic Initial Organization

Table 22: Generic Sparse Dense Time Comparison

<table>
<thead>
<tr>
<th>Simulation</th>
<th>$SPARSE$</th>
<th>$DENSE$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00117897</td>
<td>0.00206725</td>
</tr>
<tr>
<td>2</td>
<td>0.00088993</td>
<td>0.00468951</td>
</tr>
<tr>
<td>3</td>
<td>0.00219213</td>
<td>0.03423172</td>
</tr>
<tr>
<td>4</td>
<td>0.00255514</td>
<td>0.15152867</td>
</tr>
<tr>
<td>5</td>
<td>0.00312776</td>
<td>0.47058006</td>
</tr>
<tr>
<td>6</td>
<td>0.00342957</td>
<td>1.18778406</td>
</tr>
<tr>
<td>7</td>
<td>0.00411359</td>
<td>2.63835948</td>
</tr>
<tr>
<td>8</td>
<td>0.00496397</td>
<td>5.2344489</td>
</tr>
<tr>
<td>9</td>
<td>0.00623160</td>
<td>9.3906625</td>
</tr>
<tr>
<td>10</td>
<td>0.00640606</td>
<td>16.21108004</td>
</tr>
<tr>
<td>11</td>
<td>0.00642092</td>
<td>25.49529697</td>
</tr>
<tr>
<td>12</td>
<td>0.00672315</td>
<td>39.67917668</td>
</tr>
<tr>
<td>13</td>
<td>0.00778696</td>
<td>60.19342319</td>
</tr>
<tr>
<td>14</td>
<td>0.00878922</td>
<td>87.00131561</td>
</tr>
<tr>
<td>15</td>
<td>0.00971169</td>
<td>122.6883437</td>
</tr>
<tr>
<td>16</td>
<td>0.00910576</td>
<td>169.5046501</td>
</tr>
<tr>
<td>17</td>
<td>0.01086080</td>
<td>235.9751775</td>
</tr>
<tr>
<td>18</td>
<td>0.01133750</td>
<td>308.8198814</td>
</tr>
<tr>
<td>19</td>
<td>0.01070451</td>
<td>407.1512966</td>
</tr>
<tr>
<td>20</td>
<td>0.01260635</td>
<td>518.0543453</td>
</tr>
<tr>
<td>21</td>
<td>0.01296074</td>
<td>682.1078392</td>
</tr>
<tr>
<td>22</td>
<td>0.01283411</td>
<td>844.6894253</td>
</tr>
<tr>
<td>23</td>
<td>0.01555309</td>
<td>1067.985423</td>
</tr>
<tr>
<td>24</td>
<td>0.01383151</td>
<td>1341.799103</td>
</tr>
<tr>
<td>25</td>
<td><strong>0.01568137</strong></td>
<td><strong>1619.348813</strong></td>
</tr>
</tbody>
</table>

Fig. 68 shows the plot for a sparse generic initial organization. The plot, while toggling mildly, at values over $ORG_{size} = 600$, follows a linear pattern. Fig. 69 shows the plot for a dense generic initial organization. The plot follows the expectations of the exponential rate of relationship growth and the effects caused to the $Time_{InitOrg}$.
Figure 68: Generic Initial Organization - Sparse

Figure 69: Generic Initial Organization - Dense
5.1.4 Generic Reorganization

Fig. 70 shows the reorganization time of an organization which adds 4 new objects and 4 new relationships over 125 transitions which results in a final $ORG_{size} = 1000$. Although the size of the organization grows at a constant rate, the time to reorganize does not increase on a constant basis as the only objects and relationships added are used to reorganize, under their leaf goals. The initial organization, at 0, is the only major outlier in the set of transition times.

![Generic Reorganization - Constant Growth](image)

Figure 70: Generic Reorganization - Constant Growth

As Fig. 70 shows the time to reorganize for an organization with constant growth, while Fig. 71 shows the reorganize times for an organization that has a constant deletion of 4 objects and 4 relationships for each input predicate. The rate of decline is more pronounced, as shown by spikes in the figure, as relationships are removed and more assign relationships must be computed.
Figure 71: Generic Reorganization - Constant Decline
5.2 Algorithm Comparison

This section draws a comparison of algorithms based on the time to compute an initial organization or a reorganization. Because of the differing nature of each algorithm, some are grouped to compare, more specifically, their performance.

5.2.1 Initial Organization Comparison

For the following comparisons, each of the initial organizations, of each $ORG_{size}$ was executed 10 times and then the average of those times is used to determine the $Time_{Avg}$ for each data $ORG_{size}$.

![Graph](image)

Figure 72: Initial Organization Comparison - SPARSE

Fig. 72 shows the constant growth of initial organization algorithms over SPARSE relationships up to an initial the range of $Org_{size} = [41 \ldots 1001]$. The comparison of the algorithms over a SPARSE organization shows that the Generic, Central and Distributed Complete Knowledge algorithms are similar. This is consistent with the similarity they have in the computationally intense functions of each algorithm.
Fig. 73 shows the constant growth of initial organization algorithms over DENSE relationships up to an initial $\text{ORG}_{\text{size}} = [101\ldots4751]$. Once again, the Generic, Central and Distributed Complete Knowledge algorithms are similar in growth. This is consistent with the similarity they have in the computationally intense functions of each algorithm. Because the computations are so large, the lines appear to follow almost the exact line of growth.

While it appears all values are very similar, Fig. 74 shows the last 5 data points of the initial organization dense comparison. There is separation between the points, although the indication is the transfer of organization information a single time is much less of a temporal aspect, than that of organization computation.

At the ends of the spectrum, a segmented SPARSE organization has a relationship where $|\text{SEG}| = |\text{G}_{\text{leaf}}|$, whereas a segmented DENSE organization has $|\text{SEG}| = 1$, due to connectivity at all levels of the organization.

Fig. 75 shows the comparison between the 3 similar algorithms and the segmented algorithm, for initial organization. With a sparse organization, there exists maximum segmentation as each
$G_{leaf}$ represents a segment, where $|G_{leaf}| = |segments|$, so the $Time_{Compute}$ grows at a much faster rate than that of the non-segmented algorithms. The maximal $Time_{Compute}$, at each point, is caused by the requirement of segmenting the organization, prior to computing a new state. As $ORG_{size}$, and more specifically $Relationship_{size}$, grow, the rate of $Time_{Compute}$ also grows.

Fig. 76 shows the comparison with a dense organization. A dense organization computes all segmentation steps, but end with a single segment due to the complete set of relationships, so $|segments| = 1$

While the time to compute a segmented organization is quite high, compared to a non-segmented organization, an organization that lives a long life, such as an infinite organization, benefits by reduced computation to reorganize. This, of course, is dependent on the nature and structure of the organization.
Figure 75: Initial Organization Comparison with Segmentation - SPARSE

Figure 76: Initial Organization Comparison with Segmentation - DENSE
5.2.2 Reorganization Comparison

Figure 77: Reorganization Comparison - Constant Growth

Fig. 77 shows the constant growth of reorganization algorithms starting from an $ORG_{size} = 9$ up to an $ORG_{size} = 10,000$. The Central algorithm is the most linear plot and also the lowest for all $ORG_{size}$. The Generic algorithm, not having some specific advantages of Central, has a higher plot and $Time_{Compute}$ at each point, although it trends onward as the organization grows. Finally, The Distributed Complete algorithm has the highest $Time_{Compute}$ and trends upward. As the $ORG_{size}$ grows the amount of data it must transfer at each reorganization will grow linearly.

Fig. 78 shows the constant decline of reorganization algorithms from an $ORG_{size} = 10,000$ down to an $ORG_{size} = 9$. While at the beginning, there is a dramatic difference in $Time_{Compute}$ for each reorganization, based on $ORG_{size}$, all of the algorithms converge as the $ORG_{size}$ approaches 9.
Figure 78: Reorganization Comparison - Constant Decline
5.3 Validation

This research encompasses a set of implemented algorithms, based on theoretical foundations of transition. The central achievement is the definition, implementation and validation of the central, distributed and segmented algorithms, as extensions of the formal organization model. Each of the three proposed algorithms are viewed as a process to propagate an organization from one state to the next. To complete the research, the algorithms require validation. In order to validate the algorithms, we must first define validation in this domain and context. Then, three example task domains are presented in which to validate the model and three transitional algorithms.

Validation is set over an application domain that exercises the specific characteristics of each algorithm. While there is no perfect algorithm fit, the profiles of some algorithms are more appropriate to certain task domains. This section reviews examples of some domains and the most appropriate algorithms.

Validation Task Domains There are three task domains in which the organization model is implemented and used to validate this generic capability-based organization model. The generic nature of the model allows the instantiation of an organization in any capability-based task domain. Specifically, the three problems to be specified, developed, implemented and tested are a human-based simulation to write software, a sensor organization simulation to track specific phenomena through a two dimensional domain and a simple physical sensor network enabled by the organization model to adapt. The validation of the organization model depends upon whether it can transition to a new state if a new transition state exists. To complete this research effort, a valid implemented organization exists to validate the structural and state aspects of the model. Validation does not determine efficiency or predict how the model acts, it provides the assurance to indicate that the model with structural, state and transition elements work in concert to allow transitional processes to proceed in a correct manner.

If the model can be validated for these domains, it is reasonable to apply the model to any
capability-based problem domain with a high probability of success. As the success of any validation is predicated on the specifications, the instantiated models must be implemented correctly. As theoretical and basic implementation testing are not sufficient to test validation, the model is validated through experimentation with the afore mentioned problems.

The first validation test is a simulation for software development which allows the building of goals, roles and agents to develop contrived software projects. The organization is constructed with the goals of developing software artifacts. The software artifacts are not the main concern. The structure and state of the software team to advance toward completion of their goals is interrupted by a series of changes to the organization, both internal and external. Examples of these changes are people (agents) leaving the software development team and people growing in skill level such that the optimal assignment of agent to role changes, thus requiring a change in the organization. The validation arrives in the ability of the organization to transition to optimal states to continue the development of software.

The second validation scenario is that of a large software organization with many agents, roles and goals. A large software organization is typically split over many locations, with a hierarchical structure and many sub-organizations. In this scenario, the target is the application of the segmented algorithms.

The third scenario is a sensor organization monitors a spatial environment. The agents organize around specific targets that move through their area of perception. As a specific ”bogey” enters an area, an organization of sensors instantiate an organization using themselves to monitor the phenomena until they are no long able to perceive it. Some agents drop and new agents are added to the organization as it propagates through the range of the sensor organization.

These three examples provide excellent insight to determine if the organization model can support the transitional requirements of real organization problems. The scenarios also provide a good foundation for further research and application of the organization model.
5.3.1 Small Software Development Organization

The instantiation of a small software engineering project provides a small but rich example to validate the Central Control and Distributed Complete Knowledge algorithms. The organization structure and state elements are designed. Then each set of algorithms has been applied to each and the results shown through a set of organization state transitions.

The initial goal set $G$ of the organization is comprised of the following elements:

$$G = \{ g_{\text{project}}, g_{\text{plan}}, g_{\text{requirements}}, g_{\text{analysis}}, g_{\text{design}}, g_{\text{construction}}, g_{\text{testing}}, g_{\text{implementation}} \}$$

Where:

- $g_{\text{project}}$: the abstract $g_{\text{root}}$ of the organization
- $g_{\text{plan}}$: plan the project
- $g_{\text{requirements}}$: complete the requirements specification
- $g_{\text{analysis}}$: complete the detailed analysis
- $g_{\text{design}}$: complete the design
- $g_{\text{construction}}$: develop the Java code artifacts
- $g_{\text{testing}}$: unit, integration and product testing
- $g_{\text{implementation}}$: complete the move to production

The set $G = \{ g_{\text{root}}, G_{\text{int}}, G_{\text{leaf}} \}$ where $g_{\text{root}} = g_{\text{project}}$, $G_{\text{int}} = \{ \emptyset \}$ and $G_{\text{leaf}} = \{ g_{\text{plan}}, g_{\text{requirements}}, g_{\text{analysis}}, g_{\text{design}}, g_{\text{construction}}, g_{\text{testing}}, g_{\text{implementation}} \}$. Only the subgoal relationships where subgoal($g_i, g_j$) = true are included. In this example, each $g \in G_{\text{leaf}}$ is subgoal($g, g_{\text{root}}$) = true, so there is no further need of definition. The root goal, $g_{\text{project}}$ is conjunctive over all other goals.

The roles of the organization are:

$$R = \{ r_{\text{Manager}}, r_{\text{SrSE}}, r_{\text{SE1}}, r_{\text{SE2}} \}$$

where:

- $r_{\text{Manager}}$: manager of the project
- $r_{\text{SrSE}}$: senior software engineer
- $r_{\text{SE1}}$: software engineering level 1

152
$r_{SE2}$: software engineering level 2

The achieves relationships defined are:

\[
\begin{align*}
\text{achieves}(r_{Manager}, g_{plan}) & \rightarrow .8 & \text{achieves}(r_{Manager}, g_{requirements}) & \rightarrow .5 \\
\text{achieves}(r_{SrSE}, g_{requirements}) & \rightarrow .9 & \text{achieves}(r_{SrSE}, g_{analysis}) & \rightarrow .85 \\
\text{achieves}(r_{SE1}, g_{design}) & \rightarrow .75 & \text{achieves}(r_{SE1}, g_{construction}) & \rightarrow .65 \\
\text{achieves}(r_{SE1}, g_{test}) & \rightarrow .45 & \text{achieves}(r_{SE2}, g_{construction}) & \rightarrow .5 \\
\text{achieves}(r_{SE2}, g_{test}) & \rightarrow .6 & \text{achieves}(r_{SE2}, g_{implementation}) & \rightarrow .9
\end{align*}
\]

The capabilities are:

\[C = \{c_{MicrosoftProject}, c_{budgeting}, c_{Elicitation}, c_{UML}, c_{Java}, c_{unittest}, c_{integrationtest}, c_{producttest}, c_{versioning}\}\]

where:

- $c_{MicrosoftProject}$: Capability to use Microsoft Project project planning tool
- $c_{budgeting}$: Capability to budget and forecast project finances
- $c_{Elicitation}$: Elicit requirements from customers
- $c_{UML}$: Capability to design software using UML/Unified Process
- $c_{Java}$: Capability to develop Java code
- $c_{unittest}$: Capability to unit test Java
- $c_{integrationtest}$: Capability to integration test a system
- $c_{producttest}$: Final product testing of software
- $c_{versioning}$: Capability to apply versioning to the project implementation

There are a number of requires relationships. Only the relationships resulting in true are listed.

The requires relationships are:

\[
\begin{align*}
\text{requires}(r_{Manager}, c_{MicrosoftProject}) & \rightarrow true & \text{requires}(r_{Manager}, c_{budgeting}) & \rightarrow true \\
\text{requires}(r_{SrSE}, c_{Elicitation}) & \rightarrow true & \text{requires}(r_{SrSE}, c_{UML}) & \rightarrow true \\
\text{requires}(r_{SE1}, c_{UML}) & \rightarrow true & \text{requires}(r_{SE1}, c_{Java}) & \rightarrow true
\end{align*}
\]
The organization state elements are defined. When an initial organization is computed, the agents must be added to the structural elements such as goals, roles and capabilities. The agents for our example organization are:

\[ A = \{a_{\text{Gates}}, a_{\text{Jobs}}, a_{\text{Allen}}, a_{\text{Wozniak}}\} \]

The possesses are:

\[
\begin{align*}
\text{possesses}(a_{\text{Gates}}, c_{\text{MSProject}}) & \rightarrow .8 & \text{possesses}(a_{\text{Gates}}, c_{\text{budgeting}}) & \rightarrow .7 \\
\text{possesses}(a_{\text{Jobs}}, c_{\text{budgeting}}) & \rightarrow .9 & \text{possesses}(a_{\text{Jobs}}, c_{\text{elicitation}}) & \rightarrow .8 \\
\text{possesses}(a_{\text{Jobs}}, c_{\text{UML}}) & \rightarrow .3 & \text{possesses}(a_{\text{Allen}}, c_{\text{UML}}) & \rightarrow .3 \\
\text{possesses}(a_{\text{Allen}}, c_{\text{Java}}) & \rightarrow .9 & \text{possesses}(a_{\text{Allen}}, c_{\text{unittest}}) & \rightarrow .8 \\
\text{possesses}(a_{\text{Allen}}, c_{\text{integrationtest}}) & \rightarrow .7 & \text{possesses}(a_{\text{Allen}}, c_{\text{producttest}}) & \rightarrow .2 \\
\text{possesses}(a_{\text{Allen}}, c_{\text{versioning}}) & \rightarrow .5 & \text{possesses}(a_{\text{Wozniak}}, c_{\text{UML}}) & \rightarrow .8 \\
\text{possesses}(a_{\text{Wozniak}}, c_{\text{Java}}) & \rightarrow .9 & \text{possesses}(a_{\text{Wozniak}}, c_{\text{unittest}}) & \rightarrow .8 \\
\text{possesses}(a_{\text{Wozniak}}, c_{\text{integrationtest}}) & \rightarrow .7 & \text{possesses}(a_{\text{Wozniak}}, c_{\text{producttest}}) & \rightarrow .2
\end{align*}
\]

The small software team participates in a finite organization scenario, with the initial organization assignments shown in Table 23. As a software project has a specific start and end, quantified by the completion of a finite set of goals, the organization has a finite life cycle. In this case, the small project team goes through a set of transitions that lead to its own end.

The small software organization initially contains all needed objects and relationships to accomplish the goals and succeed as an organization. Over time, the organization releases objects (resources) that are no longer required, as they become available. Table 24 shows the predicates, results and transition times for 25 transitions of the small software organization. The first transition adds an agent Maddie to the organization. This predicate does not change the organization as it is
Table 23: Small Software Team Initial Assignments

<table>
<thead>
<tr>
<th>Goal</th>
<th>Role</th>
<th>Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g_{\text{plan}}$</td>
<td>Manager</td>
<td>Gates</td>
</tr>
<tr>
<td>$g_{\text{requirements}}$</td>
<td>SrSE</td>
<td>Jobs</td>
</tr>
<tr>
<td>$g_{\text{analysis}}$</td>
<td>SrSE</td>
<td>Jobs</td>
</tr>
<tr>
<td>$g_{\text{design}}$</td>
<td>SE1</td>
<td>Wozniak</td>
</tr>
<tr>
<td>$g_{\text{construction}}$</td>
<td>SE1</td>
<td>Wozniak</td>
</tr>
<tr>
<td>$g_{\text{testing}}$</td>
<td>SE1</td>
<td>Wozniak</td>
</tr>
<tr>
<td>$g_{\text{implementation}}$</td>
<td>SE2</td>
<td>Allen</td>
</tr>
</tbody>
</table>

an organization object with no connecting relationships. Transition 2 adds a possesses relationship between Maddie and the capability MSProject. Transition 3 adds Budgeting capability to Maddie, which indicates this transition is moderate. Because the possesses relationship allows Maddie to replace Gates in the role of Manager. Transition 4 allows agent Gates to exit the organization, followed by the completion of planning, the deletion of the planning phase, transition 5, and the exit of agent Maddie in transition 6.

Transition 7 allows the deletion of the Budgeting capability as the planning phase is complete. By transition 8, the requirements phase is complete, so the requirements goal is deleted. In transition 9, agent Jobs has an increase in capability, which is reflected in a change to his capability of UML. Agent Jacob is added in transition 10, but it is a minimal impact transition, as there are no relationships added. When the analysis phase is complete, transition 11 deletes the analysis goal. With the requirements phase complete, transition 12, removes the elicitation capability. Often, the achievement level required to for a role to play a goal increases and this is evident in transition 13, as the role SE2 achieves score is increased to 0.89. Transition 14 has the loss of the design goal, along with the loss of UML and Java capabilities and the construction goal, signifying the construction phase of the project is complete.

In transition 18, agent Nate is added, but since no relationships are added the transition impact is minimal. Soon after Nate is added, the team completes testing, so the testing goal is deleted in transition 19, followed by deletion of the unit test, integration test and product test capabilities in transitions 20 - 22. Agent Nate has no assignment, so he is taken out of the organization in
Table 24: Small Software Team Reorganizations

<table>
<thead>
<tr>
<th>State</th>
<th>Predicate</th>
<th>Impact</th>
<th>$t_{central}$</th>
<th>$t_{dist}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial Organization</td>
<td>NA</td>
<td>0.00207735</td>
<td>0.00187722</td>
</tr>
<tr>
<td>1</td>
<td>addagent(Maddie)</td>
<td>minimal</td>
<td>0.00008499</td>
<td>0.00006076</td>
</tr>
<tr>
<td>2</td>
<td>addpossesses(Maddie, MSProject)</td>
<td>minimal</td>
<td>0.00008856</td>
<td>0.00009079</td>
</tr>
<tr>
<td>3</td>
<td>addpossesses(Maddie, Budgeting)</td>
<td>moderate</td>
<td>0.00009834</td>
<td>0.00008856</td>
</tr>
<tr>
<td>4</td>
<td>loseagent(Gates)</td>
<td>moderate</td>
<td>0.00005224</td>
<td>0.0000528</td>
</tr>
<tr>
<td>5</td>
<td>losegoal(Plan)</td>
<td>moderate</td>
<td>0.00006230</td>
<td>0.0000595</td>
</tr>
<tr>
<td>6</td>
<td>loseagent(Maddie)</td>
<td>moderate</td>
<td>0.00010672</td>
<td>0.00010616</td>
</tr>
<tr>
<td>7</td>
<td>losecapability(budgeting)</td>
<td>moderate</td>
<td>0.00005532</td>
<td>0.00005224</td>
</tr>
<tr>
<td>8</td>
<td>losegoal(Requirements)</td>
<td>moderate</td>
<td>0.00007375</td>
<td>0.0000785</td>
</tr>
<tr>
<td>9</td>
<td>changepossesses(Jobs, UML) = 0.35</td>
<td>minimal</td>
<td>0.00005476</td>
<td>0.00005978</td>
</tr>
<tr>
<td>10</td>
<td>addagent(Jacob)</td>
<td>minimal</td>
<td>0.00004693</td>
<td>0.00004721</td>
</tr>
<tr>
<td>11</td>
<td>losegoal(Analysis)</td>
<td>moderate</td>
<td>0.00008213</td>
<td>0.00006537</td>
</tr>
<tr>
<td>12</td>
<td>losecapability(Elicitation)</td>
<td>moderate</td>
<td>0.00008241</td>
<td>0.00007068</td>
</tr>
<tr>
<td>13</td>
<td>changeachieves(SE2, impl) = 0.89</td>
<td>minimal</td>
<td>0.00006258</td>
<td>0.00005224</td>
</tr>
<tr>
<td>14</td>
<td>losegoal(Design)</td>
<td>moderate</td>
<td>0.00006118</td>
<td>0.00005671</td>
</tr>
<tr>
<td>15</td>
<td>losecapability(UML)</td>
<td>moderate</td>
<td>0.00006928</td>
<td>0.00005922</td>
</tr>
<tr>
<td>16</td>
<td>losecapability(Java)</td>
<td>moderate</td>
<td>0.00004638</td>
<td>0.00005112</td>
</tr>
<tr>
<td>17</td>
<td>losegoal(Constructor)</td>
<td>moderate</td>
<td>0.00005727</td>
<td>0.00006342</td>
</tr>
<tr>
<td>18</td>
<td>addagent(Nate)</td>
<td>minimal</td>
<td>0.00005056</td>
<td>0.00004861</td>
</tr>
<tr>
<td>19</td>
<td>losegoal(Testing)</td>
<td>moderate</td>
<td>0.00007571</td>
<td>0.00006202</td>
</tr>
<tr>
<td>20</td>
<td>losecapability(unitTest)</td>
<td>moderate</td>
<td>0.00005643</td>
<td>0.00005392</td>
</tr>
<tr>
<td>21</td>
<td>losecapability(integrationTest)</td>
<td>moderate</td>
<td>0.00006174</td>
<td>0.00011426</td>
</tr>
<tr>
<td>22</td>
<td>losecapability(productTest)</td>
<td>moderate</td>
<td>0.00005476</td>
<td>0.00005839</td>
</tr>
<tr>
<td>23</td>
<td>loseagent(Nate)</td>
<td>minimal</td>
<td>0.00015170</td>
<td>0.00016036</td>
</tr>
<tr>
<td>24</td>
<td>losegoal(Test)</td>
<td>moderate</td>
<td>0.00013242</td>
<td>0.00012823</td>
</tr>
<tr>
<td>25</td>
<td>losegoal(Implementation)</td>
<td>catastrophic</td>
<td>0.00014125</td>
<td>0.00013998</td>
</tr>
</tbody>
</table>

transition 23. Finally, the software development team completes testing and implementation, so the test and implementation goals are deleted, signifying the termination of the organization after the catastrophic impact of the deletion of the implementation goal. Since the implementation goal is the final $g \in G_{leaf}$, the organization has no more attainable leaf goals and terminates.

While this is a short example, it shows the necessary steps through a small organization’s lifecycle. While it is allowable to keep all objects and relationships as part of the organization, a goal is to always reduce the size of the organization, reducing any computation possibilities. That is shown in this case.
Fig. 79 shows the times or initial organization and subsequent reorganizations of the small development team through its software life cycle. The simulation was run using both the Central and Distributed Full Knowledge algorithms. As shown, the initial organization takes far more time than any single organization. Each unique reorganization, whether central or distributed is very similar. Both algorithms are similar in approach and execution.

![Figure 79: Transition Times - Small Software Organization](image)
5.3.2 Large Software Development Organization

The domain of a small organization, as in the small software development organization, is very different than a much larger organization. A software development organization of just 4 or 5 people initially organizes and reorganizes considering all resources available, as those resources are easy to visualize and compute. A software organization with a 1000 persons (agents) is very different in scope and the nature of what each person knows, how much they interact and what is to be considered if someone leaves or joins.

In this domain validation problem, the transitions of a larger software organization are simulated, from the initial organization through a set of reorganization states. Because of the scope and size, all people in a large software organization do not work in a single team. The organization is typically hierarchically decomposed into smaller organizations and teams, based upon a decomposed goal hierarchy. For example, it is common to have a group only working on requirements or user interaction or testing. In this case, the individual workers do not have to possess general capabilities, such as the agent in the small software development organization. The specialization of labor allows for the specialization of roles, \( r \in R \) and, in turn, the specialization of \( g \in G \), more specifically the \( g \in G_{leaf} \). The decomposition and is a natural segmentation of requirements and skills, which leads to a segmentation of sub-organizations within the overall organization. Due to these attributes of a large organization, the large software development organization simulation utilizes the *Segmentation Algorithm*.

The simulation of the large organization validation case creates an initial organization of 1000 agents. There are also 1000 \( goal_{leaf} \), but a limited set of roles. The result here is the comparison of a segmented organization to a non-segmented organization. While a non-segmented organization will only apply the changes contained within \( \phi \), to the specific \( goal_{leaf} \) relationships, there is more computation, than in a segmented organization. Of course, the position in the spectrum between a *sparse* and *dense* organization will define the level of segmentation or any organization. In this case, the average number of relationships for each object is 3. Given this number and the fact the there are 1000 agents, this indicates this organization is much closer to sparse than dense. This is
typical of large organizations.

The initial organization times are vastly different for the two organization type. As a segmented organization must first segment the large organization, its time to complete the initial organization is 2.995 seconds, while the time for the central control organization is: 0.014 seconds. These times are vastly different given the same outcome. The difference is in the long-term usage of a segmented scheme for a large organization over a long period, of many reorganizations. Fig. 80 shows the transition times over the reorganizations of the large software organizations life cycle for a central control organization and a segmented organization. For all reorganizations, starting at simulation 2, through simulation 25, the time to apply $\phi$ and transition to the next organization state is shown. In all cases, the segmented transition is better than the centrally controlled transition, given all other factors are equal.

The investment of a long initial organization time is recouped over the lifetime of the organization in lower over times resulting in less time spent computing new organization states and more

Figure 80: Transition Times - Large Software Organization
time spent working on the domain problem.

5.3.3 Sensor Organizations

This validation example is an agent-based organization of sensors with a global set of goals and utilizes the Distributed Local Algorithms. The goals task the organization to track and evaluate an object in a spatial domain. This validation scenario is the output of work conducted for the Air Force Research Labs [64, 66].

Anytime sensors are employed in a non-simulated environment, there is a distinct chance that one or more sensors will fail or be incapacitated at some point. If this occurs, the sensor organization is potentially required to transition from its current state to a new state in order to complete its goals. If this becomes the case, the organization properties define what environmental effects propagate a transition to a new state.

The concept of a sensor organization is shown in Fig. 81. A spatial domain is covered by sensors of differing type and range. The sensors can sense a number of different individual phenomena with their range and are capable of collecting data based on their positive sensing. In the figure, the domain is split into vertical and horizontal zones. The vertical zones are labeled \{A…F\} and the horizontal zones are labeled \{1…8\}. Each zone, such as (C, 2) potentially has some coverage by a sensor. In this case, (C, 2) is covered by a sensor with a green center. Each sensor type, \{blue, red, green\} has a specific range and capability to sense. Each sensor is bound to an agent. Each agent can participate in the organization instance. The sensor organization can recognize and identify an object as it moves into the boundaries of the spatial domain, as long as the capabilities of the sensors can detect the object. As the object propagates through the domain, agents capable of detecting the object organize. The organization’s goal is to detect and identify the target to the best of its ability, based on the merged capabilities of the sensors engaged.

The integration of a sensor network with a computational organization model potentially yields a more powerful system capable not only of reporting the required actionable information streams
from the sensors network, but also provide the ability to self-configure, adapt and change the employed sensors required to monitor a specific phenomenon occurring in a spatio-temporal environment. To test the idea of a sensor organization, we developed a simulation environment which allows us to alter the scenarios we are able to simulate.

The Sensor Organization Simulator, as shown in Fig. 82, is a platform to create, simulate and validate sensor organizations before moving to the more expensive proposition of constructing a physical system. The simulator is an output of research conducted under a contract for the AFRL [64, 66]. The simulation can model any sensor type through the use of capability-based modeling. If a sensor is to measure heat, then its model is built upon that capability.

The simulator has three graphical elements. The first element is the spatial domain where sensors and specific phenomena, which the sensors are to interpret, are co-located. In the current simulator, there are three basic sensor instance models. The large black circles with a green center point are large 360 degree sensors, capable of sensing a gaseous object. The smaller black circles
with a blue center point are small 360 degree sensors, capable of sensing a physical object within their range. The small triangles with the yellow end point are conical sensors, such as a single sonar, capable of detecting a physical object. Any sensor capability can be modeled.

Figure 82: Sensor Organization

The text display at the bottom of the interface displays the status of events with the spatial domain. Messages pertaining to events are added to the list each time an anomaly is detected in the environment. When a *bogey* is detected in the spatial field, a sensor event triggers a status message to be written containing the sensor agent who detected the anomaly and where the detection took place.

On the right side of the simulator interface, a simple list is displayed showing all agents involved in the sensor organization currently in action. As each sensor agent detects an anomaly, it is automatically included in the interface, even if it is redundant with another agent, reporting the same information.

The spatial domain contains all sensor agents and the anomalies currently under interpretation.
In the example, the anomaly is a cloud. The sensors agents currently detecting the anomaly are shown in red. All agents in red are part of the current instance, executing in the spatial domain.

To understand the interface between sensors, agents and the organization model previously described, we provide an small straight forward example. Fig. 83 shows a more detailed graphic of a spatial domain. The spatial domain is divided into eight specific sections labeled area$_1$...area$_8$. There are sixteen sensor agents in the domain. There are three types of sensor agents. There is a single anomaly in the domain.

![Figure 83: Sensor Example](image)

While Fig. 83 shows the spatial domain view, Fig. 84 shows an organization map of the structural and state elements and the relationships between the elements.

The organization in charge of monitoring the spatial domain has the following elements:

The goals of the organization are:

$$G = \{g_0, g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8\}$$
where:

$g_0$ is monitor environment is an abstract goal to monitor all areas.
$g_1$ is the goal to monitor area$_1$.
$g_2$ is the goal to monitor area$_2$.
$g_3$ is the goal to monitor area$_3$.
$g_4$ is the goal to monitor area$_4$.
$g_5$ is the goal to monitor area$_5$.
$g_6$ is the goal to monitor area$_6$.
$g_7$ is the goal to monitor area$_7$.
$g_8$ is the goal to monitor area$_8$.

The roles of the organization are:
$R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8\}$
where:

$r_1 \ldots r_8$ each role is required to monitor its respective area.

The capabilities are:
$C = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\}$
where:

$c_1$ monitors area$_1$ for solid objects.
$c_2$ monitors area$_1$ for gaseous objects.
$c_3$ monitors area$_2$ for solid objects.
$c_4$ monitors area$_2$ for gaseous objects.
$c_5$ monitors area$_3$ for solid objects.
$c_6$ monitors area$_6$ for solid objects.
$c_7$ monitors area$_6$ for gaseous objects.
$c_8$ monitors area$_7$ for solid objects.
$c_9$ monitors area$_7$ for gaseous objects.
$c_{10}$ monitors area7 for solid objects.

The achieves relationships, describing the numeric score for which a role can achieve a goal, simplified to all be 1.0, are defined by:

\[
\begin{align*}
\text{achieves}(r_1, g_1) & \rightarrow 1.0 \\
\text{achieves}(r_2, g_2) & \rightarrow 1.0 \\
\text{achieves}(r_3, g_3) & \rightarrow 1.0 \\
\text{achieves}(r_4, g_4) & \rightarrow 1.0 \\
\text{achieves}(r_5, g_5) & \rightarrow 1.0 \\
\text{achieves}(r_6, g_6) & \rightarrow 1.0 \\
\text{achieves}(r_7, g_7) & \rightarrow 1.0 \\
\text{achieves}(r_8, g_8) & \rightarrow 1.0
\end{align*}
\]

The 1.0 scores indicate each role has an equal ability to achieve monitoring of a sector in the spatial domain. There are a number of requires relationships. Only the relationships resulting in true are listed. The requires relationships are:

\[
\begin{align*}
\text{requires}(r_1, c_1) & \rightarrow \text{true} \\
\text{requires}(r_1, c_2) & \rightarrow \text{true} \\
\text{requires}(r_2, c_3) & \rightarrow \text{true} \\
\text{requires}(r_2, c_4) & \rightarrow \text{true} \\
\text{requires}(r_3, c_5) & \rightarrow \text{true} \\
\text{requires}(r_6, c_6) & \rightarrow \text{true} \\
\text{requires}(r_7, c_7) & \rightarrow \text{true} \\
\text{requires}(r_7, c_8) & \rightarrow \text{true} \\
\text{requires}(r_7, c_9) & \rightarrow \text{true} \\
\text{requires}(r_7, c_{10}) & \rightarrow \text{true}
\end{align*}
\]

Each of the subgoal relationships where the result is true actually have a subgoal relationship. The subgoal relationships are all of the form, subgoals$(g_0, g_n) \rightarrow \text{true}$ where $n$ is the specific agent number 2...9.

There are conjunctive goal relationships in this organization. All subgoals of $g_0$ are disjunctive. So, for each goal the conjunctive function is similar to, conjunctive$(a_n) \rightarrow \text{false}$.

The section describes the instance information of the organization state. When an initial organization is computed, the agents must be added to the structural elements such as goals, roles and capabilities. The agents for our example organization are:
The \textit{possesses} relationship describes what capabilities are possessed by an agent and the numerical score of the agent. The \textit{possesses} are:

\[
\begin{align*}
\text{possesses}(a_1, c_1) & \rightarrow 0.8 & \text{possesses}(a_2, c_1) & \rightarrow 0.8 \\
\text{possesses}(a_3, c_1) & \rightarrow 0.8 & \text{possesses}(a_4, c_1) & \rightarrow 0.9 \\
\text{possesses}(a_{20}, c_2) & \rightarrow 0.9 & \text{possesses}(a_5, c_3) & \rightarrow 0.8 \\
\text{possesses}(a_5, c_6) & \rightarrow 0.7 & \text{possesses}(a_{16}, c_7) & \rightarrow 0.2 \\
\text{possesses}(a_{17}, c_7) & \rightarrow 0.2 & \text{possesses}(a_7, c_9) & \rightarrow 0.8 \\
\text{possesses}(a_{18}, c_7) & \rightarrow 0.8 & \text{possesses}(a_{18}, c_9) & \rightarrow 0.3 \\
\text{possesses}(a_{19}, c_7) & \rightarrow 0.7 & \text{possesses}(a_{19}, c_9) & \rightarrow 0.3 \\
\text{possesses}(a_9, c_6) & \rightarrow 0.2 & \text{possesses}(a_9, c_8) & \rightarrow 0.7 \\
\text{possesses}(a_{10}, c_4) & \rightarrow 0.8 & \text{possesses}(a_{10}, c_5) & \rightarrow 0.2 \\
\text{possesses}(a_{11}, c_4) & \rightarrow 0.8 & \text{possesses}(a_{11}, c_5) & \rightarrow 0.2 \\
\text{possesses}(a_{12}, c_5) & \rightarrow 0.9 & \text{possesses}(a_{13}, c_{10}) & \rightarrow 0.9 
\end{align*}
\]
The assigned relations, which describe which agent is best to play a role to satisfy a goal are computed for each new organization state $\text{assigned}(a_n, r_m, g_p)$. As the anomaly moves through each sector, the organization transitions. As the transition occurs, then agents, roles and subgoals involved change. In this example, we look at two specific state scenarios. The first scenario is when the anomaly is located in sector 1. The second scenario is based on the anomaly in sector 5.

For each of the scenarios, all structural organization relationships are shown in Fig. 84, such as achieves, requires and possesses, formerly defined. In Fig. 85, we see the first scenario of the simulation where a cloud anomaly is in the first sector of the spatial domain. In this case, we apply the organization structural relationships previously defined to reveal that agents $a_1, a_2, a_3, a_4, a_{20}$ can detect the anomaly. Agents $a_1, a_2, a_3, a_4$ possess different capabilities than does $a_{20}$. In this case, $a_4$ and $a_{20}$ are selected and assigned. Since there are four smaller agents that can detect the anomaly, $a_4$ is selected as it has the highest possesses score of any of the similarly capable agents. The other agent, $a_{20}$ is the only larger agent capable of detecting the anomaly in that area. The role $r_1$ is played by two agents. The resulting assigned relationships are:

$$\text{assigned}(a_4, r_1, g_1) \quad \text{assigned}(a_{20}, r_1, g_1)$$

In Fig. 86, the assigned relationships are shown by red lines from the role to the agent assigned.

![Simulation Example - Sector 1](image-url)
In Fig. 87 and Fig. 88, we see the second simulation scenario where the cloud anomaly has moved to the sixth sector. Again, the organization structural relationships previously defined are applied to reveal agents $a_5$, $a_7$, $a_9$, $a_{16}$, $a_{17}$, $a_{18}$, $a_{19}$ can detect the anomaly. Agents $a_5$, $a_7$ and $a_9$ possess different capabilities than do $a_{16}$, $a_{17}$, $a_{18}$ and $a_{19}$. In this case, $a_7$ and $a_{18}$ are selected and assigned. Since there are three smaller agents that can detect the anomaly, $a_7$ is selected as it has the highest possesses score of any of the similarly capable agents. The other agent, $a_{18}$ has the highest possesses score of the larger sensors. The role $r_6$ is played by two agents. The resulting assigned relationships are:

\[
\text{assigned}(a_7, r_6, g_6) \quad \text{assigned}(a_{18}, r_6, g_6)
\]

Between the two state instances, there are a number of organization transitions formed as the anomaly moves through the spatial domain. These two examples show only two specific instances. The organization model is capable of transitioning through all stimuli from the task domain of the sensor environment.

An important aspect of sensor organizations is the ability to transition from one organization
state to the next, to monitor a spatial environment. Fig. 89 shows the transition times for a longer term simulation including 50 different organization states. The higher times on the first two states involve the initial organization, which requires more computation and thereby a longer time. The next 48 transitions each represent an anomaly moving through the spatial field and causing the sensors to organize and begin to analyze. Each new state represents a successful computation of a new state, where the goals, roles and agents have changed. The time to compute each new organization state does not increase over the life of the simulation. While the result is preliminary, it shows the ability of the system to continuously reorganize to monitor an environment and respond to various anomalies which arise in the environment.
Figure 88: Assignments - Sector 6

Figure 89: Transition Times - Sensor Organization
5.4 Summary

The validation examples show that the model and the various transitional algorithms are applicable to real world task environments. While this is by no means an exhaustive list, the future employment of the model and transition algorithms will be in a number of application task domains, beyond what was shown in this chapter.
6 Conclusions

In this section, the contribution and impact of this research effort is summarized.

6.1 Contribution of Research

The primary contribution is the completion and implementation of an organization model applicable to any capability-based organizational problem. The contribution is greatly enhanced by the ability of the organization to transition to new organizational states to overcome changes in the environment or internal problems with the organization. While there is research in the area of agent organizations and models, there is not a fully realized generic organization model with full transitional capabilities.

The overall model evaluation expresses and compares theoretical and implemented evaluations. The results compare each of the organization model algorithms in reference to a set of distinct test cases to measure the effectivity and efficiency of each. The overall summary then compares all theoretical and empirical evaluations of the model and algorithms.

Specifically the contributions are:

1. Complete transition definition
2. Distinction between initial organization and reorganization
3. Definition of transition properties
4. Complete Algorithm Specification
   - Generic
   - Central Control
   - Distributed Control Complete Knowledge
   - Distributed Control Local Knowledge
5. Complete model and algorithm implementation

- Logically (Java/JESS)
- Structurally (Java)

6. Ability to compute transition of large organizations in acceptable time

6.2 Impact

The impact of this work provides new insights into the formalization of generic transition systems for capability-based agent organizations. These do not currently exist. The use of a complete organization model with the ability to interpret the environment, respond to changes, adapt and continue its mission has application potential in any environment where organizations can be utilized to solve a problem. Examples of specific applications are battlefield evaluation and awareness, sensor networks, space exploration and self-sustaining robotic ecologies.

A second impact is the baseline and inspiration for others to extend this work, further and improve it, and as a benchmark for those who currently study or will study organization theory as applied to agents.

6.3 Future Work

This dissertation creates a basis on which a great amount of additional work can be built. The model and algorithms are applied to a number of application domains such as larger sensor networks and organizations, robotic teams and human simulation scenarios. The model can also be applied to very specific domains such as make-to-order manufacturing and manufacturing workflow task domains.

There are many extensions to the model in applying rules or laws. While these appear in the model, they were not developed in this research. Rules constraining the organization produce...
efficiencies in relationships and can reduce computation times. An interesting theoretical area to extend is that of organizational shapes, where the relationship sets are used to determine the connectivity between organizations and how that impacts overall performance, both from the ability to solve problems, but also to compute efficiently and effectively.
A Knowledge Base Approach

The realization of an organization based implementation can take many forms. This knowledge-based design and implementation is the first step in identifying the required elements, problems and solutions to realize a transitional organization model.

A.1 Realization

A model is not necessarily sufficient to completely explain the exchange of knowledge. The process must also show how the agents interact to share the information. This definition only describes the basic mechanics of the exchange. It must be further explored to answer questions on what basis is information exchanged. Is the information be shared with anyone who asks? Is the information be shared with all agents? Will it be shared with agents who do not specifically ask for it? These questions not only pose a set of philosophical queries, but also pose some practical problems in exchange. Automatically sending data to an agent that does not need it, as it already possesses the information, is wasteful in terms of resources.

Our approach to knowledge exchange is similar to the mind-body problem of Descartes. In the mind-body problem, the mind is differentiated from the matter of the body. The knowledge of an organization, which resides in the individual mind of each agent within the organization, is different than the physical manifestation of the organization. Each agent carries an image of the organization with all components and relationships. The key is for all agents to have the same image of the organization, in other words, perfect information.

The basic premise is that when each transition occurs, all agents need to be updated with the current organization knowledge. When a human organization requires change, a decision is made and the change is then communicated by the decision maker to those affected. As with human organizations, a single agent receives the change, $\phi$, and propagate the change to each of the other agents in the organization.

There exists a risk of a transition property not correctly propagating from the sending agent to
the receiving agent. If this occurs, the receiving agent does not compute a new organization and is different than those agents who successfully received the message. If for some reason, such as an agent being deleted, another agent senses the agent loss and update the others. It can also be said that each of the others can self update in the event of a loss, but questions whether each is required to recompute. A key goal is to minimize the amount of information transferred for each organization transition.

A.2 Design

Each agent optimally has the same organization knowledge. This supports the premise that all agents operate on full information. Fig. 90 shows an organization of 4 agents \{agent_1, agent_2, agent_3, agent_4\}. Agent_1 receives a transition property from either an internal or external force. Agent_1 then propagates the predicate to agent_2, agent_3 and agent_4. The organization core represents the part of the mind of the agent concerned with where it fits in the organization. The agents themselves represent the physical manifestation, or the body.

A.3 Implementation

The organization formalisms and knowledge exchange processes have been implemented to complete this work. The implementation is a combination of Java used as the main development platform with JESS utilized to implement the knowledge bases. JESS has a natural relationship with Java as described by Friedman-Hill [34] and utilizes the rete algorithm of Forgy et al. [35] and Albert [1] shows the computational fit for this algorithm applied to this technical problem.

In this section, the implementation of the structural and state elements as logical constructs in JESS are discussed. Each component and relationship are expressed as logical predicates. This logical expression represents the mind of the organization. Each predicate is sent to each agent in the body and then each agent recomputes a new organization image within their own structure. Thus, the mind of each agent in the body recomputes its own like image of the organization after
each change. All JESS logical functions are constructed with rules and facts, based on templates. The organization object is then embedded inside a Java shell for integration with the body of the organization, written in Java.

The STRUCTURE templates are:

(deftemplate achieves (slot role) (slot goal) (slot score))
(deftemplate related (slot role) (slot role))
(deftemplate requires (slot role) (multislot capability))
(deftemplate subgoal (slot goal) (slot goal))
(deftemplate conjunctive (slot goal))
(deftemplate goal (slot goal))
(deftemplate role (slot role))
(deftemplate capability (slot capability))
(deftemplate agent (slot agent))

Figure 90: Knowledge Transfer
The STATE Templates are:

(deftemplate possesses (slot agent) (slot capability) (slot score))
(deftemplate capable (slot agent) (slot role) (slot score))
(deftemplate assigned (slot agent) (slot role) (slot goal) (slot score))
(deftemplate coord (slot agent) (slot agent))
(deftemplate agentToRole (slot agent) (slot role) (slot score))

An internal template example:

(deftemplate BestAtRole (slot agent) (slot role))

Some basic rules are:

(assert (goal (goal g1)))
(assert (role (role r1)))
(assert (capability (capability c)))
(assert (agent (agent a)))
(assert (subgoal (goal g) (goal subgoal) )))
(assert (conjunctive (goal g)))
(assert (acheives (role r) (goal g) (score s)))
(assert (requires (role r) (capability c)))
(assert (possesses (agent a) (capability c) (score s)))

(defrule deletePossesses (possesses (agent a) (capability c) (score ?))
   ?fact <- (possesses (agent a) (capability c) (score ?))
   => (retract ?fact))

(defrule CheckGoals (exists (goal (goal ?)))
   => (printout t 'Goals...OK' crlf)
(foreach ?l ?rolelist (printout t ?l crlf))

(defrule CheckLeafGoals (or (and (and (goal (goal ?g)) (subgoal(goal ?) (goal ?g)))
   (not(acheives (role ?) (goal ?g) (score ?))) )
   (and (and (goal (goal ?g)) (not (subgoal(goal ?) (goal ?g))))
   (acheives (role ?) (goal ?g) (score ?))))
   => (printout t 'LeafGoals...not OK' ?g crlf)

(defrule AgentToRole (possesses (agent ?agent) (capability ?capability) (score ?s2))
   (requires (role ?role) (capability ?capability)) " +
   =>
   (store ?role (assert (agentToRole (agent ?agent) (role ?role) (score ?s2) ))))

(defrule BestAtRole (agentToRole (agent ?a1) (role ?r1) (score ?s1))
   ?fact<-(agentToRole (agent ?a2) (role ?r1) (score ?s2))
   (test (> ?s1 ?s2 ))
   =>
   (retract ?fact)

(defrule BestAtRole (agentToRole (agent ?a1) (role ?r1) (score ?s1))
   ?fact<-(agentToRole (agent ?a2) (role ?r1) (score ?s2))
   (test (> ?s1 ?s2 ))
   =>
   (retract ?fact)

(defrule assign (and (acheives (role ?role) (goal ?goal) (score ?s1))
   (agentToRole (agent ?agent) (role ?role) (score ?s2)))

179
Functionally the initial organization program looks like:

```java
initialOrganization()
{
    defineTemplates();
    defineFacts();
    checkGoals();
    checkLeafGoals();
    defineAgentToRole();
    defineBestAtRole();
    defineAssignments();
}
```

Functionally the reorganization program looks like:

```java
public void executeReorganization()
{
    checkGoals();
    checkLeafGoals();
    defineAgentToRole();
    defineBestAtRole();
    defineAssignments();
}
```

Structural and State assertions look like:
STRUCTURE Assertions

//*******************************************************
(assert (goal (goal complete_system)))
(assert (goal (goal analyze_system)))
(assert (goal (goal complete_program)))
(assert (goal (goal test)))
(assert (subgoal (goal complete_system) (goal analyze_system)))
(assert (subgoal (goal complete_system) (goal complete_program)))
(assert (subgoal (goal complete_system) (goal test))))
(assert (conjunctive (goal complete_system)))
(assert (achieves (role coder) (goal complete_program) (score .5)))
(assert (requires (role coder) (capability program)))
(assert (achieves (role analyzer) (goal analyze_system) (score .7)))
(assert (requires (role analyzer) (capability analysis)))
(assert (role (role coder)))
(assert (role (role analyzer)))

(assert (possesses (agent eric) (capability program) (score .75)))
(assert (possesses (agent kris) (capability analysis) (score .5)))
(assert (possesses (agent kris) (capability program) (score .87)))
(assert (possesses (agent eric) (capability analysis) (score .7)))

A.4 Initial Results

We must first distinguish between results split by the two transition processes, initial organization and reorganization. The results indicates the initial organization is computationally more intense and is based on the number of components and relationships. Since it is only computed once in each organization’s life, its effect is discounted. Reorganization is much smaller, due to the incremental nature of only having to recompute around new components and relationships of the
\( \phi \) predicate. If \( \phi \) is quite large, it may alter the computational intensity. For example, if the number of components and relationships in \( \phi \) is equal or greater than the existing organization, reorganization may be computationally large.

The computation of a transitioning organization differs from an initial organization to a reorganization. In a strictly structural context, initial organization and reorganization do not differ a great deal. In our mind body approach the difference is significant. For a small organization size of 10 goals, 10 roles, 10 agents and complete relationship set, the time for initial organization is 0.03219 seconds. The time for a reorganization based on one new component and all relationships is 0.01754 seconds. Fig. 91 shows the time to compute a transition against the size of the organization in elements. The initial organization used in this analysis has 10 organization components, such as roles, agents or goals, and 15 relationships between those components. The total number of components, on the lower end, is 25. The data shows the time to compute the transition going from 25 components to 100, which is beyond a trivial organization. The transition process is based on computing an optimal organization configuration. The key is that the time to recompute is not significantly different for the larger organization. This is due to the incremental nature of the computation process. This indicates use of this method is at least initially, is scalable.

If we compare this timing to another result by Zhong it shows the difference. In Zhong’s research[86], based on a similar model, using only the constructive version of the structural model algorithm to transition, the results of a structural computation yields two interesting points. First,
the structural model transition algorithm grows at a fast rate as the number of organization components grow. Secondly, the ability to scale to large organizations are be significantly hindered by a strictly structural approach. This indicates as the size of the organization grows, the difference between our approach and a more structural-based approach grows, in terms of time to compute.

Instantiating an organization and its transition processes in terms of a mind-body approach has advantages over a strictly structural computational approach. While there are also a few disadvantages, these are overcome by the positives.

Computation minimization is the best result of this approach. While larger, more complex organizations must be tested, the early results show promise. The computation is performed locally and in parallel, which allows the transition process to be completed more rapidly. The intent is for each agent to work with complete information and to have the same organization image, without transferring the entire structure each transition cycle. The rate of message growth is small. Even with a large change, all computation is local. This allows a near linear growth rate during organization augmentation. This reduces temporal computation problems in transition processes.

There are a few negative side effects of this approach. Perfect information requires that information is transferred from agent to agent without interruption or error. If there is a transfer loss, the synchronization of the organization image maps suffer. Recovering from loss, during exchange, is key for the design. There must be synchronization allowing each agent to recompute simultaneously with all others. If there are lag times, it can create temporal problems in transition.

A.5 Evaluation

In this section, we extend the implemented organization to show the reaction to larger numbers of organization properties being sent to the organization and how the various arrangements affect the organization as it grows and shrinks, due to the changes. There are a number of ways an organization may change. The important factor is how long it takes to recompute, if it is indeed possible to recompute. As shown in the workshop example, there are organization properties that
force the organization to a premature exit, in which all goals may not be accomplished.

As time to recompute is important to consider, we are compelled to consider the time required to recompute organizations that get very large. It is important to consider the shape of data that results from large incorporation of transition properties into an existing organization. In this section, we simulate initial organization and the reorganization with large sets of properties. Then analysis is made, based upon the size and type of properties used.

A.5.1 Initial organization

While the process of initial organization only occurs once in the life of any organization, we must look at the computational cost of instantiating the organization. In simulating the initial organization, the minimum initial organization that is valid must have at least one goal, role, capability and agent. There also must exist an achieves, requires and possesses relationship between the objects. While there is a single initial organization transition, the number of predicates involved vastly changes the time to initially reorganize. If there is a minimal organization of four objects and three relationships, the time is quite small. If a large number of objects and relationships are included in the complex predicate as input to initial organization, the time can be quite large. The results form the baseline from a comparative result, on a similar structural model [86] and a previous minor result with this model [65].

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Object</th>
<th>Relationship</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>0.019182323</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0.019335698</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>4</td>
<td>0.020033945</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>4</td>
<td>0.019820116</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td>0.019809222</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>5</td>
<td>0.019670378</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>5</td>
<td>0.021213310</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>6</td>
<td>0.019777095</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
<td>0.019732396</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>7</td>
<td>0.020067076</td>
</tr>
</tbody>
</table>
Table 25 shows 10 configuration simulation states of an initial organization computation. The minimum number of objects and relationships is seven comprised of four objects and three relationship predicates. This is shown in simulation 1. Each of the following simulations adds either one object or one relationship prior to computing the initial organization state. Fig. 92 shows the general trend as the time generally increases from an initial organization with 7 predicates to a larger initial organization computation with 16 predicates, as shown in simulation 10 of Table 25.

![Figure 92: Initial Organization Time](image)

Fig. 93 compares the time to initial organizations on three ranges of differing complex predicate sizes. The simulations were executed in groups of 10. For this comparison, each simulation contains 4 object predicates and 4 relationship predicates. The lowest plot shows the time to compute an initial organization with a single group of 8 simple predicates, representing a single simulation. Each subsequent simulation, adds 8 predicates. For example, the second simulation contains 16 simple predicates, the third simulation contains 24 simple predicates all the way to the tenth simulation which contains 80 simple predicates. The middle plot shows the simulations with the initial complex predicate containing 10 sets of 8 simple object and relationship predicates. The simulations then step by 10 sets per simulation until the final simulation contains 100 groups of 8 simple predicates each. This shows the time to initially compute an organization with a maximum of 800 elements. The upper plot steps from 100 sets to 1000 sets, giving the time to compute large initial organizations. In this set, the time to initially compute an initial organization containing 4000 objects and 4000 specified relationships is 0.187157 seconds. This organization also computes additional relationships where possible. The results indicate the capability of the initial organization
algorithm to compute a very large initial organization, in minimal time.

![Initial Organization Comparison on Ranges of Predicates](image)

Figure 93: Initial Organization Comparison on Ranges of Predicates

### A.5.2 Reorganization

Reorganization requires that a single transition property φ be present as a catalyst to enable execution. In the model, transition properties are implemented as predicates. The predicates can be represented as either simple predicates or complex predicates. The reorganization process is simulated from a number of perspectives. The first view is that of adding objects and deleting objects.

### A.5.3 Simple Properties

The first evaluation uses simple properties for each transition. This is defined by a single simple predicate contained within each φ transition property. In this section, we look at adding simple object predicates followed by the use of complex predicates. Within using simple predicates, we look at adding objects and relationships individually, adding objects and relationships mixed together and deleting simple predicates. Then, we look at the organization transition using complex predicates of differing size up to transition properties that contain more objects and relationships which are larger than the original organization to which they are added. For each of these simulation experiments, the transitions are measured over the time it takes to compute the new state, from the existing state. Because the transition processes are computationally intensive, the time required
to reach a new state is critical in creating valid, realistic organization models and corresponding transitional algorithms.

### A.5.4 Adding Objects

While it is legal to add an object to an organization, if subsequent relationships do not bind the object to other objects within the structure, the object has no bearing on the outcome. It has an effect on the reorganization computation process.

Table 26 illustrates the data for simulation of adding strictly objects to an organization. The initial simulation 1, which represents the initial organization, is higher than the remainder of the simulations. There are 3 relationship predicates, required to form a minimal organization, along with 4 objects. In addition, each subsequent predicate is an object and is not related to any other object with a relationship predicate. Fig. 95 shows the relationship of the time for each simulation to execute. Even though the number of predicates grows, the trend is reasonably flat, even decreasing minimally.

![Figure 94: Reorganization: Objects Only](image)

Figure 94: Reorganization: Objects Only
Table 26: Reorganization: *Objects Only*

<table>
<thead>
<tr>
<th>State</th>
<th>Object</th>
<th>Relationship</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>0.019421184</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0.009218770</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>0.004433245</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>3</td>
<td>0.004319823</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>3</td>
<td>0.004259200</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>3</td>
<td>0.004261994</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>3</td>
<td>0.004357537</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>3</td>
<td>0.004076216</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>3</td>
<td>0.003963074</td>
</tr>
<tr>
<td>10</td>
<td>13</td>
<td>3</td>
<td>0.003939887</td>
</tr>
<tr>
<td>11</td>
<td>14</td>
<td>3</td>
<td>0.003874795</td>
</tr>
<tr>
<td>12</td>
<td>15</td>
<td>3</td>
<td>0.003565257</td>
</tr>
<tr>
<td>13</td>
<td>16</td>
<td>3</td>
<td>0.003514134</td>
</tr>
<tr>
<td>14</td>
<td>17</td>
<td>3</td>
<td>0.003386185</td>
</tr>
<tr>
<td>15</td>
<td>18</td>
<td>3</td>
<td>0.003680914</td>
</tr>
<tr>
<td>16</td>
<td>19</td>
<td>3</td>
<td>0.003629080</td>
</tr>
<tr>
<td>17</td>
<td>20</td>
<td>3</td>
<td>0.003534248</td>
</tr>
<tr>
<td>18</td>
<td>21</td>
<td>3</td>
<td>0.003352102</td>
</tr>
<tr>
<td>19</td>
<td>22</td>
<td>3</td>
<td>0.003302934</td>
</tr>
<tr>
<td>20</td>
<td>23</td>
<td>3</td>
<td>0.003168279</td>
</tr>
<tr>
<td>21</td>
<td>24</td>
<td>3</td>
<td>0.003366908</td>
</tr>
<tr>
<td>22</td>
<td>25</td>
<td>3</td>
<td>0.003882058</td>
</tr>
<tr>
<td>23</td>
<td>26</td>
<td>3</td>
<td>0.003139318</td>
</tr>
<tr>
<td>24</td>
<td>27</td>
<td>3</td>
<td>0.003138387</td>
</tr>
<tr>
<td>25</td>
<td>28</td>
<td>3</td>
<td>0.003203480</td>
</tr>
</tbody>
</table>
A.5.5 Adding relationships

We cannot, by definition, add relationships, without having objects that correspond to the relationships, so in this simulation, objects and relationships are added one at a time. If relationships are added without corresponding objects, they result in orphan predicates that cannot participate in the organizational structure.

Table 27 and Fig. 95 illustrates the data when adding objects and relationships for the objects. In this simulation, objects and relationships added relate to each other. No orphan objects or relationships are considered or added. The initial organization simulation begins with four objects and three relationships, the minimal organization. Then, each simulation adds 4 objects then three relationships to tie the objects together. The last simulation contains twenty objects and fifteen relationships. While the organization transition time decreases from the initial organization to the next set of reorganizations, the overall time to reorganize to the last simulation is reasonably flat. There is some variation as the organization grows in size. This is reasonable for a small organization.

![Figure 95: Reorganization: Objects and Relationships](image)

Fig. 96 shows the trend of time for a large number of reorganizations. In this set of simulations, 390 reorganizations were completed adding even numbers of objects and relationships. The trend is fairly flat until approximately 40 reorganizations. With the relations added, there are additional relationships generated with the existing properties. In this case, the end result is the initial
organization and 390 additional reorganizations. During the reorganizations, yielding 390 rules, 199 additional relationships were constructed, yielding a total of 509 total objects and relationships. The toggling of times is explained by additional new relationships being generated and computed as planned relationships are added. Time for the last reorganization is 0.109313614 seconds.

Figure 96: Reorganization: Adding Objects and Relationships

A.5.6 Deleting Objects

When objects are deleted from the organization, there is a compound effect on the relationships bound to the objects. This is shown in Figure 97. Since a relationship requires two objects, the removal of the two objects also requires removal of any relationships currently bound to that object. Adding 390 objects and relationships, then deleting 390 objects and relationships returns the organization state to the original minimal organization.

Figure 97: Reorganization: Adding and Deleting Objects and Relationships

190
A.5.7 Adding Complex Predicates

Complex predicates consist of a set of simple predicates, of any size. In Table 28, the number of objects and relationships versus time is described. In this simulation, an initial minimal organization of 4 objects and 3 relationship predicates is computed on the first organization. The initial organization also adds a global goal, for which all other goals are subgoals. For each additional transition, sets of 4 objects and 4 relationships are added. The size of the organization grows quickly.

Fig. 98 show the curve for adding complex predicates, of size 8, to an existing organization. The increase in transition time, each state, rises slightly upward. The initial organization, which includes objects and relationships, decreases to a lower time on the reorganization processes.

![Figure 98: Reorganization: Complex Predicates](image)

Fig. 99 shows the simulation of an initial organization with 4 objects and 4 relationships, then increasing by 4 objects and 4 relationships through 125 states until the organization has a size of 1000 total added objects and relationships. The total number of relationships is actually larger. The upward trend stair steps at certain points, when the addition of new objects forces a set of new relationships to be asserted, which were not included in the predicate. The nature of these new assertions are assignments and capable relationships.
A.5.8 Deleting Complex Predicates

Fig. 100 shows the simulation of an initial organization with 4 objects and 4 relationships, then increasing by 4 objects and 4 relationships until it has a size of 1000 total added objects and relationships, which is a continuation of the simulation shown in Fig. 18. Once the organization is at a size of 1000 complex predicates deleting objects and relationships, at the same rate of 4 objects and 4 relationships occur. The trend to recompute the organization progresses downward. At the end, the time is not symmetric to the starting times, as there are still relations which exist in organization as a by product of the complex add predicates.

Figure 100: Reorganization: Complex Predicates - Add and Delete Sets
A.5.9 Adding complex predicates equal to existing organization

Often, if an organization is forced to merge with another organization, then organization may grow by a non-trivial size. For example, if two equally sized organizations merged and became one, each organization doubles its original size. In this simulation, we studied the time effects of send predicates, to the organization, that double the size of the organization each transition. Table 14 shows the data used in this simulation. The predicate indicates the size of the predicate where each predicate has 4 new objects and 4 new relationships to add. After the initial organization, the predicate size doubles each time. The only caveat to this is a subgoal relationship was added in the initial organization, which slightly lessens the doubling factor for relationships. As the organization grows, the numbers of internally added relationships, capable and assigned, also grows. The added components totals 8193 objects and relationships.

The time curve, shown in Table 29, indicates the doubling of the organization takes an greatly upward trend as the organization transitions from a size of 1025 to the next size of 2049. Future evaluation will test the limits of organization size. While an organization size of 4096 objects is not trivial, larger organizations do exist. Recomputing an organization of size 8193 in 1.199 seconds is reasonable. Most organization transitions do not involve all objects and relationships during a transition.

Figure 101: Reorganization: Complex Predicates - Double Size
A.5.10 Deleting Complex Predicates larger than existing organization

As organizations may grow at levels doubling the current size of the organization, or more, they may also shrink at the same or greater rates. If a company splits off divisions, the company may be only half the size of its previous organization. In this simulation, the organization decreases by half with each transition property. Table 15 shows the size of the predicates, number of predicate object and relationships, and transition time. Table 30 and Fig. 102 shows the transition time from state to state. As the time increased rather quickly as the organization doubled, the time to recompute decreases quickly as the organization size halves each transition. The transition time does toggle after a steady decrease.

Figure 102: Reorganization: Complex Predicates - Reduce By Half

A.6 Conclusions

The ability to formalize the basic processes of organization transition is fundamental to the understanding, capture and application of evolved biological organization models to multiagent systems. The process of transition, organization or reorganization, is seemingly very simple, but the reality is complex.

In the paper, we have shown the theoretical premises of translating agent organization properties into formal predicates. These predicates can then be applied to specific transition processes for instances of agent organizations.
There are two categories of transition property predicates, primitive and complex, which can represent any formal property of agent transition. A simple transition predicate represents a change in a single transition property. A complex predicate is a logical set of simple predicates joined by conjunctive or disjunctive relationships.

The properties and predicates have been created to be flexible in approach. The reason for this is the applicability to any organization model. Any organization model based on logic can use this basic set of predicates to model the transition processes associated with initial organization or reorganization processes.
Table 27: Reorganization: Objects and Relationships

<table>
<thead>
<tr>
<th>State</th>
<th>Object</th>
<th>Relationship</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>0.020193069</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0.011930008</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>0.004773232</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>3</td>
<td>0.004273728</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>3</td>
<td>0.004492470</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>4</td>
<td>0.004786642</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>5</td>
<td>0.004592204</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>6</td>
<td>0.004573765</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>6</td>
<td>0.005894045</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>6</td>
<td>0.004533816</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>6</td>
<td>0.004406147</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>6</td>
<td>0.003848254</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>7</td>
<td>0.004614833</td>
</tr>
<tr>
<td>14</td>
<td>12</td>
<td>8</td>
<td>0.004278197</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>9</td>
<td>0.005519975</td>
</tr>
<tr>
<td>16</td>
<td>13</td>
<td>9</td>
<td>0.004215340</td>
</tr>
<tr>
<td>17</td>
<td>14</td>
<td>9</td>
<td>0.004423467</td>
</tr>
<tr>
<td>18</td>
<td>15</td>
<td>9</td>
<td>0.004098845</td>
</tr>
<tr>
<td>19</td>
<td>16</td>
<td>9</td>
<td>0.003946032</td>
</tr>
<tr>
<td>20</td>
<td>16</td>
<td>10</td>
<td>0.004273448</td>
</tr>
<tr>
<td>21</td>
<td>16</td>
<td>11</td>
<td>0.005125791</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>12</td>
<td>0.004109460</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>12</td>
<td>0.004183772</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>12</td>
<td>0.004255848</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>12</td>
<td>0.004841118</td>
</tr>
<tr>
<td>26</td>
<td>20</td>
<td>12</td>
<td>0.003981233</td>
</tr>
<tr>
<td>27</td>
<td>20</td>
<td>13</td>
<td>0.004317588</td>
</tr>
<tr>
<td>28</td>
<td>20</td>
<td>14</td>
<td>0.004254172</td>
</tr>
<tr>
<td>29</td>
<td>20</td>
<td>15</td>
<td>0.004432407</td>
</tr>
</tbody>
</table>
Table 28: Reorganization - Complex Predicates

<table>
<thead>
<tr>
<th>State</th>
<th>Object</th>
<th>Relationship</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>0.019922364</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>8</td>
<td>0.005504051</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>12</td>
<td>0.006399417</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>16</td>
<td>0.007452902</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>20</td>
<td>0.006987201</td>
</tr>
<tr>
<td>6</td>
<td>24</td>
<td>24</td>
<td>0.007305118</td>
</tr>
<tr>
<td>7</td>
<td>28</td>
<td>28</td>
<td>0.007605436</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>32</td>
<td>0.007381943</td>
</tr>
<tr>
<td>9</td>
<td>36</td>
<td>36</td>
<td>0.008508065</td>
</tr>
<tr>
<td>10</td>
<td>40</td>
<td>40</td>
<td>0.007355124</td>
</tr>
<tr>
<td>11</td>
<td>44</td>
<td>44</td>
<td>0.008310833</td>
</tr>
<tr>
<td>12</td>
<td>48</td>
<td>48</td>
<td>0.007521347</td>
</tr>
<tr>
<td>13</td>
<td>52</td>
<td>52</td>
<td>0.008910071</td>
</tr>
<tr>
<td>14</td>
<td>56</td>
<td>56</td>
<td>0.007806020</td>
</tr>
<tr>
<td>15</td>
<td>60</td>
<td>60</td>
<td>0.011343062</td>
</tr>
<tr>
<td>16</td>
<td>64</td>
<td>64</td>
<td>0.008287087</td>
</tr>
</tbody>
</table>

Table 29: Reorganization - Doubling Predicates

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Pred</th>
<th>Object</th>
<th>Relationship</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>0.030100753</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>0.01913232</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>16</td>
<td>17</td>
<td>0.012051531</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>32</td>
<td>33</td>
<td>0.012455214</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>64</td>
<td>65</td>
<td>0.019561704</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>128</td>
<td>129</td>
<td>0.021259406</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>256</td>
<td>257</td>
<td>0.034361909</td>
</tr>
<tr>
<td>7</td>
<td>64</td>
<td>512</td>
<td>513</td>
<td>0.053398966</td>
</tr>
<tr>
<td>8</td>
<td>128</td>
<td>1024</td>
<td>1025</td>
<td>0.254627893</td>
</tr>
<tr>
<td>9</td>
<td>256</td>
<td>2048</td>
<td>2049</td>
<td>0.587807947</td>
</tr>
<tr>
<td>10</td>
<td>512</td>
<td>4096</td>
<td>4097</td>
<td>1.199390984</td>
</tr>
</tbody>
</table>
Table 30: Reorganization - Halving Predicates

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Pred</th>
<th>Object</th>
<th>Relationship</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>512</td>
<td>4096</td>
<td>4097</td>
<td>1.199390984</td>
</tr>
<tr>
<td>2</td>
<td>256</td>
<td>2048</td>
<td>2049</td>
<td>0.694899212</td>
</tr>
<tr>
<td>3</td>
<td>128</td>
<td>1024</td>
<td>1025</td>
<td>0.418405691</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>512</td>
<td>513</td>
<td>0.308071557</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>256</td>
<td>257</td>
<td>0.256492654</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>128</td>
<td>129</td>
<td>0.225215216</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>64</td>
<td>65</td>
<td>0.30098574</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>32</td>
<td>33</td>
<td>0.300371695</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>16</td>
<td>17</td>
<td>0.362175081</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>0.322663076</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>0.322274759</td>
</tr>
</tbody>
</table>
References


[33] Filipe, Joaquim. A Normative and Intentional Agent Model for Organizational Modeling. Third international Workshop on Engineering Societies in the Agent’s World (ESAW ’02), Universidad Rey, Juan Carlos, Madrid, Spain (EU), Sept. 16-17, 2002.


Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAA-
MAW '97), Magnus Boman and Walter Van de Velde (Eds.), Springer-Verlag,
Berlin, Germany, 1997.

[38] Grosz, B. and Kraus, S. Collaborative Plans for Complex Group Action, Artificial
Intelligence, vol. 86, no. 2, pp. 269-357, 1996.

Organization of the Co-operative Movement in Great Britain and Ireland. Co-
operative Union Ltd., Manchester, UK, 1934.

[40] Mahdi Hannoun, Olivier Boissier, Jaime Simao Sichman and Claudette Sayettat.
MOISE: An Organizational Model for Multi-agent Systems. IBERAMIA-SBIA,

[41] Hoogendoorn, M., Jonker, C.M., Schut, M.C., and Treur, J., Modeling Central-
ized Organization of Organizational Change, Computational and Mathematical

[42] Hoogendoorn, M., Jonker, C.M., and Treur, J., Redesign of Organizations as a
Basis for Organizational Change. In: P.Noriega et al. (eds.), COIN 2006 Work-

[43] Hoogendoorn, M., Jonker, C.M., Schut, M.C., and Treur, J., Modeling Organi-
zational Change for Naval Missions, WSEAS Transactions on Systems, vol. 5,
2006, pp. 532-543.

[44] Rodney R. Howell, On Asymptotic Notation with Multiple Variables, Kansas

[45] Jomi Fred Hbner, Jaime Simo Sichman, Olivier Boissier. MOISE+: Towards a
Structural, Functional, and Deontic Model for {MAS} Organization, Proceedings
of the First International Joint Conference on Autonomous Agents and Multi-

[46] Hayes-Roth, Barbara, Pfleger, Karl, Lalanda, Philippe, Morignot, Philippe,
Balabanovic, Marko. A Domain-Specific Software Architecture for Adaptive Intelli-
gent Systems. IEEE Transactions on Software Engineering, Vol 21, No. 4, April
1995.

[47] Linli He, Thomas Ioerger, A Quantitative Model of Capabilities in Multi-Agent
Systems. International Conference on Artificial Intelligence (IC-AI 2003), Las

[48] Hopcroft, John E., Motwani, Rajeev, Ullman, Jeffrey D. Introduction to Au-
tomata Theory, Languages and Computation, 2nd Edition. Pearson Education,

202


