Self-Organising Resource Allocation in Open Systems

Theory and Experiments

Julia Gertrud Schaumeier

Submitted in part fulfilment of the requirements of the degree of Doctor of Philosophy of Imperial College London
Abstract

Open, embedded multi-agent systems have applications in sensor and opportunistic networks, and in cloud and grid computing. Features of such systems include: no centralised control, competition for resources between autonomous agents, (un)intentional errors, and a speed and complexity of decisions beyond human capabilities. Moreover, there is a requirement to optimise performance with respect to multiple, possibly conflicting, criteria, for example longevity, occupancy and fairness.

This thesis addresses the problem of engineering self-organised resource-allocation management schemes for open, embedded systems. Based on the theory of institutions for collective action as defined by political economist Elinor Ostrom, we define a formal model for self-organised resource-allocation, using the computational framework of dynamic norm-governed multi-agent systems.

Our model of an electronic institution encapsulates a first-order logic axiomatisation of the principles for enduring institutions. An experimental platform for an abstract common-pool resource management situation has been developed, and the experiments show the importance of all principles in order to achieve longevity, appropriate behaviour and the right balance of membership. The results furthermore suggest that the mechanisms to design institutional rules should be made available to the system components themselves. In order to successfully self-organise, the system has to be aware of its internal state and externalities.

To represent and reason about awareness, some aspects from the field of organisational justice have been formalised in the same framework. Agents will not only follow a collectively decided allocation procedure but will execute the allocation according to their own notion of fairness, and also use this notion to judge the perceived behaviour of others. Further experiments show that the ability for introspection and reflection on the perceived environment leads to an improved management profile and further enhances the system’s performance.

Adaptive institutions are a key factor in dealing with resource distribution. Self-aware agents using electronic institutions in socio-technical systems could be a significant innovation in reducing the current lag between institutional and environmental change, and make an important contribution to the sustainability agenda.
Acknowledgements

"Thanks. Thanks very much." [90]

This thesis could not have been written without the help and support of many people. Foremost I want to thank my two supervisors who guided me in shift operation through various stages of my research.

Dr Moez Draief greatly assisted me during my initial period of problem formulation by presenting a range of problems to me and encouraging me to look at different disciplines for solving them. The various techniques he suggested provided me with a good foundation for further research.

Dr Jeremy Pitt covered the next shift and did a marvellous job. He sparked my interest for the type of allocation problems considered in this work and gave me the opportunity for collaboration and exchange with many interesting researchers from social and computer science. The highlights were a trip to the Workshop in Political Theory and Policy Analysis in Bloomington (IN) to meet Elinor Ostrom, the award of the best paper prize jointly with Alexander Artikis at the Self-Adaptive and Self-Organizing Systems conference just a few days later, and giving a talk in front of an audience of 150 people with bare feet. I greatly benefited from travelling with Jeremy, the time spent on planes, trains, or in Indian guest houses was very well suited to focus on particular problems in detail or to concoct possible lines of research. Without his ongoing intellectual (and sometimes alimentary) support I could not have reached the point of submission.

I am very grateful to two projects, UKIERI (SA08-037) and QNRF (09-1150-2-448), for their financial support and for the intellectual gains I made from involved collaborations. Many people I met through these projects inspired my work in one or the other way.

Further thanks go to the colleagues from our research group with whom I had many interesting and motivating discussions. Special thanks go to Sam Macbeth who provided much needed 24/7 support for the simulation platform Presage2.

There were many people more who made working on my thesis a great deal easier: Family, friends and flatmates. Although I could list them individually, I think they would appreciate it more if I took them down to the pub after my viva.
Declarations

I, Julia Schaumeier, hereby declare that this work is my own, and where it is based on or derived from the work of others, I have acknowledged this and included a reference in the bibliography.

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution-Non Commercial-No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or distribution, researchers must make clear to others the licence terms of this work.

London, September 2013
1. Introduction 19
   1.1. Motivation .............................................................. 19
   1.2. Methodological background ........................................ 20
   1.3. Thesis outline .......................................................... 21
   1.4. Contributions .......................................................... 26

2. Resource Allocation in Open Systems 27
   2.1. Introduction .......................................................... 27
   2.2. Features of open systems .......................................... 27
   2.3. Examples of open systems .......................................... 28
   2.4. Different types of resource replenishment ..................... 30
       2.4.1. Exogenous replenishment .................................... 30
       2.4.2. Endogenous replenishment .................................. 32
       2.4.3. Hybrid replenishment ....................................... 33
   2.5. Common-pool resource management ................................ 33
       2.5.1. Top–down (closed) ............................................. 34
       2.5.2. Bottom–up (open) ............................................ 34
       2.5.3. Sustainability issue ......................................... 35
   2.6. Institutions for CPR management .................................. 35
       2.6.1. Institutions ...................................................... 35
       2.6.2. Rules ............................................................. 36
       2.6.3. Principles ...................................................... 37
   2.7. Related approaches .................................................. 39
       2.7.1. Computational social choice ................................ 39
       2.7.2. Voting mechanisms ........................................... 39
       2.7.3. Fair division .................................................... 41
       2.7.4. Game-theoretic example .................................... 45
   2.8. Summary and Problem Specification .............................. 46

3. Self-Organisation 47
   3.1. Introduction .......................................................... 47
   3.2. Sociologically-inspired computing ............................... 47
3.3. Dynamic specification of open systems ........................................ 49
  3.3.1. Agents for open systems ................................................. 50
  3.3.2. Institutions for open systems ......................................... 50
  3.3.3. Institutionalised power ................................................ 51
  3.3.4. Rule-based specification change ...................................... 51
3.4. Relating the work of Ostrom and Artikis ................................. 53
3.5. Formal characterisation ..................................................... 54
  3.5.1. Roles ............................................................................. 55
  3.5.2. Rules and methods ........................................................ 56
  3.5.3. Nesting of rules, roles, methods ...................................... 57
3.6. Open systems model ............................................................. 59
3.7. Simulation Platform .............................................................. 60
3.8. Formalism: Drools ................................................................. 61
  3.8.1. Rule-based systems ........................................................ 62
  3.8.2. Drools Expert ............................................................... 64
3.9. Formal axiomatisation of six principles ...................................... 67
  3.9.1. \( P1 \): Clearly defined boundaries .................................... 68
  3.9.2. \( P2 \): Appropriation and provision rules, local conditions ...... 70
  3.9.3. \( P3 \): Collective-choice arrangements ............................. 71
  3.9.4. \( P4 \): Monitoring .......................................................... 74
  3.9.5. \( P5 \): Graduated sanctions .............................................. 75
  3.9.6. \( P6 \): Conflict-resolution mechanisms ............................ 77
3.10. Summary ............................................................................. 79

4. Self-Organising Resource Allocation ........................................... 81
  4.1. Introduction .......................................................................... 81
  4.2. Principled operationalisation .................................................. 81
    4.2.1. Testbed specification in Presage2 ..................................... 82
    4.2.2. Classes .......................................................................... 83
    4.2.3. States ............................................................................ 86
    4.2.4. Testbed control loop ....................................................... 87
  4.3. Controlled experimentation ..................................................... 95
    4.3.1. Experimentation with brute facts ..................................... 95
    4.3.2. Experimenting with institutional facts .............................. 96
    4.3.3. Goal .............................................................................. 97
  4.4. Results ................................................................................ 97
    4.4.1. Existence and management principles ............................... 98
    4.4.2. Protection principles ....................................................... 99
    4.4.3. Alternatives from the parameter set .................................. 101
    4.4.4. Comparing the refill schemes ....................................... 103
  4.5. Evaluation ........................................................................... 109
5. **Awareness and Fairness**

5.1. Introduction .................................................. 113

5.2. Self-awareness for self-organisation .............................. 114

5.2.1. Self-awareness in computer science .......................... 114

5.2.2. Levels of awareness in social science ........................ 115

5.2.3. Awareness in open systems .................................. 116

5.3. Organisational justice in open systems .......................... 118

5.3.1. Organisational justice ........................................ 118

5.3.2. Procedural justice perceptions ............................... 119

5.3.3. Three norms of justice ..................................... 120

5.4. Fairness in open systems ....................................... 122

5.4.1. P7: Minimal recognition of rights to organise ............... 123

5.4.2. P8: Nested enterprises ...................................... 125

5.5. Summary ..................................................... 126

6. **Fair Self-Organising Resource Allocation** ..................... 127

6.1. Introduction .................................................. 127

6.2. Fairness judgement ............................................ 127

6.3. Principled operationalisation .................................. 128

6.3.1. Ruleflow .................................................. 128

6.3.2. Agent behaviour ........................................... 130

6.4. Controlled experimentation ...................................... 134

6.4.1. Parameters .................................................. 134

6.4.2. Experimentation with brute facts ............................ 136

6.4.3. Experimentation with institutional facts ...................... 136

6.5. Results ....................................................... 137

6.5.1. Fairness evaluation for the CUSTOM refill scheme .......... 138

6.5.2. Fairness evaluation for HIGH, MODERATE and LOW refill schemes .... 145

6.5.3. Evaluation ................................................. 151

6.6. Summary ..................................................... 155

7. **Summary, Conclusion and Future Work** ........................ 157

7.1. Final summary ................................................ 157

7.1.1. Conclusions from the testbed and its extension ............ 158

7.1.2. Answering the five questions ................................ 161

7.2. Limitations ................................................... 162

7.3. Future Work .................................................. 166

7.3.1. Principles and methods: Institutional facts ............... 166

7.3.2. Environment: Brute facts ................................. 167
# Contents

D.9. Head :: allocate .................................................. 216  
D.10. Gatekeeper :: split ........................................... 217  
D.11. Agency :: appoint ........................................... 218  
D.12. Additional or changed field values ......................... 218  
D.13. Control loop for CPR testbed extension .................... 220  

E. Supporting material (extension) .................................. 223  
E.1. End distribution of single trials ............................... 223  
E.2. Average end distribution ..................................... 224  
E.3. Purity on average and for individual trial .................. 225
List of Figures

1.1. Methodology for sociologically-inspired computing (SIC) 20
1.2. Chapters following the SIC methodology 22
2.1. Dependencies of rules and levels of analysis 37
3.1. Sociologically-inspired computing 49
3.2. K-level protocol stack for dynamic specifications 52
3.3. Institutional rules versus dynamic specification 53
3.4. Nesting of rules for resource allocation 57
3.5. Nesting of rules for role assignment 58
3.6. Basic operation of a production system 62
4.1. Creating a simulation with Presage2 82
4.2. Flow of ruleflow groups 83
4.3. Class diagram of the testbed 84
4.4. Agent activity state chart 86
4.5. Rule flow in the resource allocation process 89
4.6. P1, P2 and P3, refill scheme CUSTOM 98
4.7. P4, P5 and P6, refill scheme CUSTOM 100
4.8. P4 ‘good’ population, refill scheme CUSTOM 101
4.9. P4 ‘bad’ population, refill scheme CUSTOM 102
4.10. P1, P2 and P3, refill scheme comparison 104
4.11. P4, P5 and P6, refill scheme comparison 105
4.12. P4 ‘good’ population, refill scheme comparison 107
4.13. P4 ‘bad’ population, refill scheme comparison 108
4.14. P4 ‘good’ and ‘bad’ population, efficiency comparison 111
5.1. Different levels of awareness 115
6.1. Rule flow in the resource allocation process with fairness judgement 129
6.2. Satisfaction, head change and remaining agents for custom refill scheme 139
6.3. Remaining agents by distribution norm for custom refill scheme 140
6.4. Distribution of remaining agents by norm for custom refill scheme 141
List of Figures

6.5. Purity by distribution norm for custom refill scheme . . . . . . . . . . . . 142
6.6. Subcluster change of socialist population \((Pr7, Pr8)\) . . . . . . . . . . . . 143
6.7. Subcluster change of socialist population \((rate = 15, Pr8)\) . . . . . . . . . . 144
6.8. Satisfaction, head change and remaining agents for high refill scheme . . 146
6.9. Satisfaction, head change and remaining agents for moderate refill scheme 147
6.10. Satisfaction, head change and remaining agents for low refill scheme . . 148
6.11. Distribution of remaining agents by norm for low refill scheme . . . . . . 150
6.12. Purity by distribution norm for low refill scheme . . . . . . . . . . . . . . 151
6.13. Subcluster change of socialist population \((Pr7, Pr8), high/moderate\) . . 152
6.14. Subcluster change of socialist population \((Pr7, Pr8), low\) . . . . . . . . 153
6.15. Subcluster change of socialist population \((rate = 15, Pr8), low\) . . . . 154

C.1. \(P1, P2\) and \(P3\), refill scheme HIGH . . . . . . . . . . . . . . . . . . . . . . . . . . . 203
C.2. \(P4, P5\) and \(P6\), refill scheme HIGH . . . . . . . . . . . . . . . . . . . . . . . . . . . 203
C.3. \(P4\) ‘good’ population, refill scheme HIGH . . . . . . . . . . . . . . . . . . . . . . . . . 204
C.4. \(P4\) ‘bad’ population, refill scheme HIGH . . . . . . . . . . . . . . . . . . . . . . . . . 204
C.5. \(P1, P2\) and \(P3\), refill scheme MODERATE . . . . . . . . . . . . . . . . . . . . . . . 204
C.6. \(P4, P5\) and \(P6\), refill scheme MODERATE . . . . . . . . . . . . . . . . . . . . . . . . 205
C.7. \(P4\) ‘good’ population, refill scheme MODERATE . . . . . . . . . . . . . . . . . . . . 205
C.8. \(P4\) ‘bad’ population, refill scheme MODERATE . . . . . . . . . . . . . . . . . . . . . 206
C.9. \(P1, P2\) and \(P3\), refill scheme LOW . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 206
C.10. \(P4, P5\) and \(P6\), refill scheme LOW . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 206
C.11. \(P4\) ‘good’ population, refill scheme LOW . . . . . . . . . . . . . . . . . . . . . . . . . 207
C.12. \(P4\) ‘bad’ population, refill scheme LOW . . . . . . . . . . . . . . . . . . . . . . . . . 207

E.1. Subcluster change of capitalist population \((Pr7, Pr8)\) . . . . . . . . . . . . 223
E.2. Subcluster change of capitalist population \((rate = 15, Pr8)\) . . . . . . . . . 224
E.3. Distribution of remaining agents by norm for high refill scheme . . . . . 224
E.4. Distribution of remaining agents by norm for high refill scheme . . . . . 224
E.5. Purity by distribution norm for moderate refill scheme . . . . . . . . . . . . 225
E.6. Purity by distribution norm for moderate refill scheme . . . . . . . . . . . . 226
# List of Tables

2.1. Design principles for managing CPR institutions ........................................ 38

3.1. Example processes in institution and dynamic specification ......................... 55
3.2. Selected keywords of the Drools Expert system (RHS) ............................... 66

5.1. Agent preferences for distribution norms .................................................. 122

6.1. Probability of allocation by distribution norm ........................................... 135
6.2. End distribution of socialist population ($Pr_7, Pr_8$) ................................ 143
6.3. End distribution of socialist population ($rate = 15, Pr_8$) .......................... 144
6.4. End distribution of socialist population ($Pr_7, Pr_8$), low .......................... 152

7.1. Benefits of Ostrom's principles for resource allocation in open systems .......... 158

B.1. Values for testbed initialisation ............................................................... 199
B.2. Field values for Common Pool ............................................................... 200
B.3. Field values for Institution ................................................................. 200
B.4. Field values for Member ................................................................. 201
B.5. Field values for NonMember .............................................................. 201
B.6. Experimental parameters of individual runs ............................................ 202
B.7. Figures for sets of runs according to refill scheme .................................... 202

D.1. Extension field values .............................................................................. 219

E.1. End distribution of capitalist population ($Pr_7, Pr_8$) ............................... 223
E.2. End distribution of capitalist population ($rate = 15, Pr_8$) ......................... 224
E.3. End distribution of socialist population ($Pr_7, Pr_8$), high/moderate .............. 226
E.4. End distribution of socialist population ($rate = 15, Pr_8$), low ................... 226
List of Rules and Functions

3.1. Drools class declaration ............................................ 68
3.2. “Grant access to non-member” ..................................... 68
3.3. “Sanction monitored non-members” ................................. 69
3.4. “Member demands” .................................................. 70
3.5. “Perform resource allocation” ....................................... 70
3.6. “Call for votes” ...................................................... 71
3.7. “Vote for raMethod” ............................................... 72
3.8. “Count votes” ........................................................ 72
3.9. “Declare winner and update raMethod” ........................... 73
3.10. “Member monitoring list” .......................................... 74
3.11. “Deduct member monitoring cost” ................................. 74
3.12. “Sanction members” ............................................... 75
3.13. “Sanction members (repeated offences)” ......................... 76
3.14. “Member back after sanction” ..................................... 77
3.15. “Appeal against sanction” ......................................... 78
3.16. “Uphold sanction” ................................................. 78

5.1. “Recognise winner head” ........................................... 124
5.2. “External appointment of head” .................................... 124
5.3. “Split resources for subclusters” .................................. 125

A.1. double demand(); .................................................... 183
A.2. Vote vote(); .......................................................... 184
A.3. double appropriate(); .............................................. 184
A.4. void changeBehaviour(); .......................................... 185
A.5. “Members appeal against sanction” ............................... 186
A.6. double appropriate(); .............................................. 186
A.7. “Assign new head” .................................................. 186
A.8. “Include member after sanction” .................................. 187
A.9. “Member exclusion”, “Member exclusion with graduated sanctions” ..... 187
A.10. Set<String> monitor(); “Agent monitoring list” .................. 188
A.11. CallForVote callForVotes(); “Call for votes” ...................... 189
A.13. Set<Allocation> allocate(); ....................................... 191
A.14. “Sanctioning monitored agents” (3 rules) .......................... 192
A.15. “Refill pool”, “Refill pool - HIGH/MODERATE/LOW scheme” .... 194
A.16.“Common pool depleted”, “End of the institution” ................................. 195

D.1. Vote vote(); ........................................................................................................ 209
D.2. “Update satisfaction if head appointed externally or head elected” ........ 209
D.3. “Collect members for sample”, sample(); ......................................................... 210
D.4. “Judge head allocation”, void judgeHead(); ..................................................... 211
D.5. “Members leave due to dissatisfaction” .............................................................. 213
D.6. “Members change subcluster” .......................................................................... 214
D.7. “Declare winner head” ...................................................................................... 214
D.8. “Update winner head” ....................................................................................... 215
D.9. Set<Allocation> allocate(); ............................................................................... 216
D.10.”Adjust maximum pool level” ......................................................................... 217
D.11.“External appointment of head” ....................................................................... 218
1. Introduction

1.1. Motivation

Open systems, from sensor networks and virtual organisations to urban transportation and smart grids, face a common challenge. They consist of heterogeneous and competing components, are resource constrained and decentralised. The system components, often referred to as agents, form opportunistic alliances and need to collectivise and distribute resources without a centralised controller. They depend on mechanisms to self-organise despite malfunction or intentional disruption, without the intervention of the designer at runtime.

In this work, we address the problem of adaptation and self-organisation in open systems with respect to resource allocation. If such systems involve human actors, they are often privatised or placed under governmental administration, as a game-theoretic analysis of strictly rational actors predicts the depletion of a resource that is governed by private actors, such as in a commons. Elinor Ostrom however, showed that depletion is not the inevitable outcome when leaving people to self-organise the process of resource allocation [85]. In most cases where the resource is sustainable, the actors have formed an institution, i.e. a set of rules, that governs their behaviour with respect to appropriating from the common pool. These rules are conventionally agreed by the concerned actors, are mutable, mutually understood, monitored and enforced, and are nested within each other. They also define what role an actor has to fulfil to be permitted (or obliged) to perform specific actions. Furthermore, the rules are organised in different levels to address the different nature of issues arising when taking collective action. These are the operational, collective-choice and constitutional level. She observed that despite forming an institution, there were still cases when the resource did not sustain continuous appropriation. Comparing these cases to examples of successful self-governance, she formalised a catalogue of eight principles for long-enduring common-pool resource (CPR) management. We use Ostrom’s theory on institutions for collective action as a starting point for designing electronic institutions for self-organising open systems.
1. Introduction

1.2. Methodological background

In order to implement a theory from social sciences within a computing context, we apply the methodology of sociologically-inspired, based on work by Steels [111], that we previously presented in [93]. The basic structure of that methodology is shown in Figure 1.1. Given some phenomena and appropriate (research) tools, a preformal theory can be constructed, typically in natural language. That language is then represented in a calculus of choice, in order to attach computational meaning to the theory. Then, the represented theory is embedded into a computational environment and can include the implementation of individual agent behaviour and the environment. Finally, the computer model is used for experimentation and the obtained results allow us to draw conclusions about the accuracy of either preformal theory, calculus or computer model.

Given that we use preformal theories created by sociology-related sciences, there remain three steps for us to apply from this methodology. They are the formal characterisation, the principled operationalisation and the controlled experimentation.

In the first step, the formal characterisation, we characterise the theory of institutions for collective action in the open computing systems context. To this end, we use a computational framework for specifying electronic institutions [8], and part of this framework is the specification of a norm-governed system\(^1\) including the permissions, obligations and institutionalised powers [59], and a protocol stack for defining how to change the specification. This allows the agents to modify the rules (or protocol) of the institution at runtime. To formalise the norm-governed system, we will use a rule-based reasoning engine that efficiently matches rules and facts.

\(^{1}\)The norms in the norm-governed context are stated in explicit form, which we typically refer to as rules.

Figure 1.1.: Methodology for sociologically-inspired computing (SIC)
The formalisation of principles represents what is explicitly expressed with the institution (the rules), but we also have to consider their implicit meaning (the norms). The implicit meaning makes assumptions about the actors’ internals, that is that people are able to reason about the social or organisational context they are in. For agents in open systems this entails that the principles should not just be implemented by the system designer but their specification has to be made available to the system components themselves. We therefore endow the agents with some notion of awareness and self-awareness, which enables them to reason about the rules with respect to the state of the environment, and in particular about their own and other agents’ behaviour.

In the second step, the principled operationalisation, we integrate the formal characterisation into a testbed, using a time-driven simulation platform. We define the functions for agent behaviour, the institution and the environment for different types of CPRs and bring everything together with a control loop.

In the third step, the controlled experimentation, we test the performance of principles on a resource-allocation scenario where the resource is supplied by the environment. We assume an economy of scarcity, meaning that there is neither an overabundance of resources available, nor is there so little resource that the system is in constant crisis. Under these conditions, the agents have to apply the rules of the institution. They decide on a policy that defines how the resource is distributed and by whom, and they monitor the appropriation phase and sanction any noncompliant behaviour.

We use objective and subjective evaluation methods for judging the longevity, occupancy and fairness of the system. In a first set of experiments, we optimise the choice of principles from the designer’s perspective with respect to enduring system operation, high membership and a well-known fairness measure. In a second set of experiments, the components of the open system optimise occupancy and fairness from an agent’s view, the subjective system perspective. To evaluate the fairness of procedures, the agents use self-awareness. This constitutes one aspect that human actors regularly consider when designing rules for resource allocation. Here, the agents compare the outcome of an allocation procedure to their internal fairness norm and draw consequences, such as reassigning the role of the distributor.

1.3. Thesis outline

The body of work is divided into five chapters and follows the sociologically-inspired computing methodology. Figure 1.2 shows in what order these chapters correspond to the individual steps of the methodology.
Chapter 2 firstly presents different types of open systems and reviews several approaches to the problem of resource allocation within these systems from a computer science perspective. Then, we present common approaches to CPR management in the physical world, including institutions for collective action and self-governance of CPRs, and the following question arises:

**Q1** Can the problem of resource allocation in open systems be addressed by modelling an institution for self-governance?

The theory of institutions formalised by Elinor Ostrom represents the first step of theory construction from the SIC methodology. We describe the problem of defining rules of an institution and conclude with eight principles [85, p. 90] that serve as guidelines for designing those rules.

Lastly, we include relevant background from computational social choice, fair divisions and an example from game theory, that deals with the problem of resource allocation.

Chapter 3 is a first step towards integrating Ostrom’s principles into the decision-making process of open systems. We start by revisiting the methodology of sociologically-inspired computing and present possible frameworks for that integration. In order to express required, permitted or prohibited actions that can be performed within an institutional context, we chose to specify the open system using a framework of norm-governed systems [8] and answer the question:

**Q2** Can Ostrom’s design principles be encoded in norm-governed systems?
1.3. Thesis outline

In the remainder of the chapter, we address the second step of the methodology, i.e. the formal characterisation, and the calculus (or rather formalism) used. We describe how the nesting of rules is realised in both frameworks, create roles and specify the methods\(^2\) that are used to govern the resource-allocation process in an open system. We then instantiate a formal model of the open system as a basis for the computer model and present the formalism we use for expressing the institution. Finally, we formally axiomatise the first six principles using the chosen formalism.

After positively answering the second question, the following question remains:

\[ Q3 \]
Is it possible to use the formal axiomatisation to specify and implement a testbed that ascertains the sufficiency of these principles for enduring open systems?

Chapter 4 addresses the steps of principled operationalisation and controlled experimentation of the SIC methodology. For the principled operationalisation, we first specify the testbed using a time-driven simulation platform. We then introduce the classes that define the different roles, the institution and the environment, and the control loop that defines what rules are applied when and in what order. We explain in detail what actions the agents perform occupying a specific role, and the individual agent behaviour, including components that allow each agent to deviate from the prescribed behaviour. For example, the agents can decide not to comply with the rules regulating the appropriation of resources and also change this behaviour in subsequent steps.

For the step of controlled experimentation, we describe the parameters that we manipulate. These include parameters that influence the environment, e.g. the resource replenishment or factors that lead to unforeseen errors upon resource appropriation, and the agent population. Further parameters specify what principles are used during experimentation to find out what set of principles is sufficient in what environmental circumstances. To this end, we analyse three categories of experiments. The first one uses the first three principles on a compliant agent population and shows the manageability of resource allocation in open systems. The second one uses the next three principles on a noncompliant agent population and where unintentional appropriation errors occur, in order to show the protectability of the resource. For the third category, we use two sets of experiments to test the effect of changing the parameters of a single principle on a compliant and a noncompliant agent population. These experiments show from an objective perspective that it is important to implement the principles with respect to the prevailing environment.

\(^2\)A method is a group of rules that address a specific issue, such as conflict resolution or access control.
After evaluating the results we can positively answer the third question, but the results also raise another one:

Q4 Can we equip the agents with mechanisms to evaluate the self-organisation?

Chapter 5 tries to answer this fourth question. To do this, we re-evaluate and extend the preformal theory, formalism and computer model by performing a second iteration of the SIC methodology where we complement Ostrom’s theory with subjective mechanisms of evaluating the institution, system and agent behaviour.

The preformal theory we used for self-organising resource allocation in open systems are Ostrom’s design principles for sustainable resource management. These principles express the explicit requirements on an institution for self-governance, i.e. the rules. However, there is an implicit expectation on the behaviour of human actors that is not mentioned explicitly.

This expectation is that the actors reason about the individual, sociological or organisational context they are in and use this perception in their internal decision-making process. The degree to what such a perception is accurate influences the behaviour of the actor itself and its judgement of other actors’ behaviour, which in turn depends on the norms that an actor perceives to be the standard. In short, the explicit requirements lead to long-enduring CPR management, based on the assumption that human actors create rules of the institution based on (to them) satisfying and fair procedures.

For agents in open systems however, the implicit assumptions have to be made explicit. As there is no human intervention at runtime, it is necessary that the agents are able to reason about the environment and consequences of their own and other agents’ actions. We therefore endow them with the capability of exhibiting (self-)awareness. We will explain several levels of awareness that are used by humans and relate to equivalent mechanisms used in artificial societies.

On the example of organisational fairness, we test the impact of self-awareness by using an agent that judges other agents’ behaviour in relation to its internalised norm. This category of fairness influences the motivation for compliance with the rules, and the efficiency and satisfaction within an organisation [32]. In open systems, the functionality of measuring fairness from within the system (using the agents’ perspective) is particularly important, as it is meaningless for a system’s operation to objectively measure fairness from an external perspective.

We complement the formal characterisation from Chapter 3 with the two remaining design principles. These principles give the agents an opportunity to apply their self-
1.3. Thesis outline

aware capabilities by seeking alternatives to current allocation procedures, should they perceive them as unfair. The following question arises:

Q5 Does a qualitative evaluation of processes enable the agents to make more informed choices with respect to self-organisation?

Chapter 6 contains the second round of principled operationalisation and controlled experimentation. The agent model is extended with the capability of self-aware fairness evaluation, and the functionality for this method is integrated into the corresponding agent class. The control loop of the resource allocation process is extended with rules that initiate the fairness evaluation process and rules for actions that the agents are permitted to take in response to that evaluation.

Again, we define several parameters for experimentation that specify the principles present in the institution, particular agent behaviour and the resource replenishment of the environment. Furthermore, we introduce five measurements for evaluating the data obtained from experimentation. We experiment with different parameter modulations on the last two principles, and identify several factors that positively or negatively influence the satisfaction of the agent population. The results suggest that the subjective evaluation mechanism works well and we can positively answer the fifth question.

Chapter 7 summarises this work by drawing conclusions from the testbed and its extension. With these findings, we argue that the choice of parameters has to be made available to the agents themselves. We then recapitulate the answers to the five questions asked in previous chapters. Afterwards, we present several limitations of the testbed and the design decisions that were made when applying the methodology of sociologically-inspired computing. We conclude this work with suggesting several lines of research, such as assisted resource allocation or alternative methods for fairness evaluation.

We have presented several parts of this work in previous publications. In a journal paper called “Axiomatization of Socio-Economic Principles for Self-Organizing Institutions: Concepts, Experiments and Challenges”, we address the problem of engineering electronic institutions that can self-organise the resource allocation process in open systems. The concepts we mention in this paper are revisited and elaborated in Chapters 2, 3 and 4. From a further publication, called “A tripartite analytic framework for characterising awareness and self-awareness in autonomic systems research”, we adopt parts of Chapter 5 on self-awareness in open systems.
1. Introduction

1.4. Contributions

The contributions of this work are threefold. It contains the first formal characterisation of Ostrom’s design principles for creating institutions for resource management in a rule management system. The second contribution is the design and implementation of a large-scale, reusable experimental testbed, where we transform design principles into policies or procedures that can be directly executed. And thirdly, two sets of experiments have been made with this testbed. The first set of experiments shows that Ostrom’s principles are sufficient conditions for creating electronic institutions for self-organising and enduring resource management. The second set shows that these principles can be complemented by heuristic fairness theory for optimising individual and collective performance, which provides the means for the system to operate in the best interest of its components given only their subjective opinions and interactions with which to work.

In total, this thesis shows how theories of collective action can be formalised and leveraged for engineering solutions to resource-allocation problems that are frequently encountered in the deployment of open systems and networks.
2. Resource Allocation in Open Systems

2.1. Introduction

In this chapter we discuss types of open systems that are subject to the problem of resource allocation, such as wireless sensor networks, vehicular ad hoc networks, virtual organisations, or systems for demand-side infrastructure management. The nature of the resource in each of these systems is different, and there are different ways that a resource can be replenished and different ways to manage its distribution. We will discuss exogenous and endogenous resource replenishment as well as a hybrid approach, and how resources are commonly managed in human societies, either top–down or bottom–up. We elaborate on a specific example of bottom-up approaches, the evolution of institutions for common-pool resource management. This example is chosen to manage resources in the types of systems considered in this work. Lastly, we address related concepts from the literature for dealing with resource-allocation problems, such as computational social choice, including examples on voting and fair division, and a game-theoretic approach. Sections 2.4.1 and 2.6 are based on our work presented in [93].

2.2. Features of open systems

Open systems are a conglomeration of autonomous components that interact with themselves and the environment in order to achieve individual and common goals. These systems offer substantive advantages over centralised or closed solutions with respect to scale, opportunity and generativity. The success of open systems is based on the assumption that the components cooperate and coordinate their behaviour. As we stated previously in [93], this makes the system tolerant to heterogeneity, resource conflicts and unforeseen events.

Applications of open systems include swarm robotics, cloud computing, infrastructure management, and so on. In all these applications, the components have to share information and resources to achieve their goals, though complete information is not usually available. Reasons for that are manifold: the heterogeneity of the components and their
unknown provenances; the mutable and unpredictable environment that the system is embedded in; and the involved numbers of components that can lead to reasoning with only partial information, depending on the components’ configuration.

Typically, the decision-making processes in those applications are far too fast and too frequent for manual operator intervention. This means the system has to operate autonomously, i.e. it needs a mechanism to coordinate individual behaviour without an external decision-making authority.

Without uniform goals, no common knowledge and no central controller, the decisions that the components are taking individually are subject to uncertainty, and conflicting opinions and requirements. This is complicated by the fact that the systems have to operate even in conditions when resources get scarce and the components have to prioritise individual and common goals. In some cases, a complete depletion of the resource, such as battery power or server space, is the most undesirable outcome and can lead to a collapse of the open system as a whole.

Further problems arise when the assumption of cooperation is void, and the components exhibit selfish behaviour and deliberately disrupt the system to achieve conflicting goals. Therefore, the system must be able to operate under conditions of intentional and unintentional error, in order to recover from suboptimal states.

The components of an open system are also referred to as agents. Artikis et al. [12] name three characteristics that allow us to consider a (multi-agent) system as open, if present. Firstly, the internal architecture of the agents in the system is not publicly known; secondly, there is not necessarily a global utility function that is shared by the agents; and thirdly, the agents’ behaviour and interactions cannot be predicted in advance. Or as Hewitt [55] states it, open systems are “always subject to unanticipated outcomes in their operation”.

2.3. Examples of open systems

Our main concern with open systems is how open systems handle sharing resources. In the following we name a few applications where components (possibly developed by different parties) or agents form opportunistic alliances and depend on pooling and sharing resources.

**Wireless sensor networks** A wireless sensor network (WSN) consists of a collection of typically resource constrained sensor nodes that are distributed in the environment. Their purpose is to sense events like temperature, speed or particle density. They process this information via the (ad hoc) network [33], for example for data collection purposes or...
2.3. Examples of open systems

to autonomously perform actions, such as intervention in a production line. The specifications of how to process the sensed information depends on the use case. Sensors can be mobile or stationary, alert or sleeping, and the node hierarchy can be pre-determined or revised constantly. An example of resources in sensor networks is the electricity needed for (multi-hop) communication or data processing.

**Vehicular ad hoc networks**  Vehicular ad hoc networks (VANets) support traffic management, passenger safety and driver assistance. Therefore, sensors are installed into vehicles, form ad hoc networks with vehicles in the vicinity, and then communicate with either road side units or other vehicle clouds. The sensors process and exchange information about the environment, and alert the driver about any unexpected changes. This could be a traffic jam, an accident behind a curve or a dangerously overtaking vehicle. VANets are different from most wireless sensor networks in that their connectivity is sporadic and the contact time between vehicles is relatively short [96]. Energy in this case is not a resource that is likely to be constrained, rather the volume of shared data and its correctness.

**Virtual organisations**  Virtual organisations (VO) allow companies to decentralise the management of projects and companies and to pursue “problem solving based on collaboration in computation- and data-rich environments” [46]. Multiple disciplines can collaborate and share their knowledge and facilities, such as databases, software and computing power. VOs can be set up very quickly and tailored to specific tasks. The decentralisation allows employees to work from any location at any time and across boundaries of various physical or virtual institutions. As projects involve more and more people, a decentralised management maintains scalability and mobility. VOs can be formed to support cloud computing for enterprises where “dynamically scalable and often virtualized resources are provided as a service over the Internet” [18]. These resources can be combined with ‘real’ resources such as time (especially for real-time services), computing power, electricity, and so on.

**Demand-side infrastructure management**  As the world’s population is growing fast and gravitating towards cities, maintaining the reliability of infrastructure such as transport, water and electricity, is a key concern. In [113], the benefits and challenges of demand-side management of an electricity system are evaluated in order to balance generation and need. Suggestions for distributed and reactive mechanisms to protect critical infrastructure are presented in [56].

Several examples in the literature investigate these types of open systems from a multi-agent perspective. Vinyals et al. [118] survey multiple contributions and identify several
challenges that agent technologies bring to the sensor network domain. A framework for Virtual Organisations that are managed by intelligent agents can be found in [77], an agents approach to managing building energy systems is described in [126].

2.4. Different types of resource replenishment

In the above examples, the resources are of different nature and their availability is coupled with the type of replenishment. We will firstly discuss resource allocation methods used for exogenous replenishment, where the resource is provided by the environment and the agents have no control over this process. Then, we present problems that occur with endogenous replenishment, where the agents themselves have to provide the resource by pooling individual contributions. The third part is a hybrid approach where both exogenous and endogenous replenishments are used. All resources are implied to be divisible goods.

2.4.1. Exogenous replenishment

An example where a system relies on external events for resource replenishment is an irrigation system. Individuals have no control over rainfall or melt water to refill reservoirs, but have to deal with the state of the environment at hand.

To formulate the resource that an agent can be allocated at any time, we define a resource allocation system (following our previous work [93]) as

\[ \langle A_t, P_t, m_t \rangle \]

where at time \( t \in \mathbb{N} \) (and we regard time as rounds or slices\(^1\)) \( A_t \) is the set of agents that would like to appropriate the resource and \( P_t \in \mathbb{R}^+ \) is the pooled resource itself, e.g. for water this would be the reservoir containing a possible refill and any remaining resource from \( t-1 \). With \( m_t \) we denote the operation of performing the resource allocation, that is \( m_t : A_t \rightarrow P_t \). Each agent \( a \in A_t \) is allocated a certain fraction of the resource \( P_t \), with the constraint that all agents together cannot be allocated more than the pooled resource itself minus some critical threshold \( D \in \mathbb{R}^+ \):

\[ \sum_{a \in A_t} m_t(a) \leq P_t - D \]

The function \( m_t \) can be defined in various ways, for example the agents \( a_i \) (with \( i \in \)\(^1\)These might accidentally be called ‘steps’ from time to slice.)
2.4. Different types of resource replenishment

\{1, \ldots, |A_t|\} can form a queue \(a_1 - a_2 - \ldots - a_{|A_t|}\) and allocate the resource in a first-come-first-serve manner. This happens until there is no more resource left and the agents remaining in the queue get nothing. The utility \(u_{a_t}\) that any one agent gains from its allocation at time \(t\) depends on the actual appropriation \(r_{a_t}(t)\) it makes (in an optimal situation\(^2\) we have \(m_t(a_i) = r_{a_t}(t)\):

\[
u_{a_t}(t) = \begin{cases} 
  r_{a_t}(t), & \text{if } \sum_{j=1}^t r_{a_j}(t) \leq P_t \\
  0, & \text{otherwise.} 
\end{cases}
\]

However, if \(P_t - \sum_{a \in A_t} r_a(t) < D\), we consider the resource as depleted and no further appropriations are possible at a later time. This happens for example in water reservoirs near the coast, where salt water flows in and ruins the fresh water supply, should the resource level fall below the critical threshold \(D\).

Typically, an agent wants to maximise the sum of individual utilities\(^3\) over a specific time frame \([S, T]\), i.e. \(U_{a_t}(\{S, T\}) = \max \sum_{t=S}^T u_{a_t}(t)\). Let \(T = S\), then the time frame is one time slice and the maximum that the first agent in the queue \(a_1\) can appropriate is \(P_S\). For \(T = S + 1\), the maximum amounts to \(U_{a_1}(\{S, S + 1\}) = \max (P_S, P_{S+1} + P_S - D)\). Given that at time \(S\) the replenishment for \(S + 1\) is unknown as well as what other agents appropriate further down the queue, a rational agent will choose to appropriate the full amount \(P_S\) straight away (unless there are no other agents).

This example is maximising the utility for only one agent and two time slices. Given that all agents try to do the same and not all agents’ utilities can be maximised simultaneously, this is a classic “tragedy of the commons” as described by Hardin in [52].

We can see, with the time frame \([S, T]\) becoming longer, the agents’ rational behaviour inevitably leads to the depletion of the pooled resource, as the constraints on \(P\) have to hold throughout:

\[
U_{a_t}(\{S, T\}) = \max \sum_{t=S}^T u_{a_t}(t),
\]

where \(\forall t \in [S, T - 1] : \sum_{a \in A_t} r_a(t) \leq P_t - D\) and \(\sum_{a \in A_T} r_a(T) \leq P_T\).

For a sustainable resource and enduring system operation, a resource-allocation method that respects these constraints has to be defined (and enforced) so that allocations and appropriations coincide, i.e. \(m_t(a_i) = r_{a_t}(t)\) for all \(i\) at any time \(t\). This means that the

\(^2\)In this situations the agents follow the allocation when appropriating the resource.

\(^3\)Here, maximising the utility is equal to maximising the appropriated resource, but this only holds as long as too much resource is not a disadvantage.
agents gain less utility than possible in each time slice, which “may be suboptimal in the short run but prove wiser in the long run” [87].

There are various other methods to determine an appropriate allocation $m_t$. The effect of different allocation methods in divisible-good auctions is examined in [65], and a cake-cutting algorithm to allocate each agent to equal satisfaction is described in [23].

2.4.2. Endogenous replenishment

Endogenous resource replenishment requires individual agents to contribute their own resources to the system, as is the case in ad hoc networks, for example. The contributed resource serves all agents to achieve their goals, for example multi-hop messaging.

Again, we consider the resource allocation system $\langle A, P, m_t \rangle$.

This time, the allocation function $m_t$ has to take into account the individual agent contributions $c_a(t)$ (assuming there is no remaining resource from the last time slice):

$$\sum_{a \in A} m_t(a) \leq \sum_{a \in A} c_a(t)$$

In this type of scenario, the pooled resource is usually distributed equally amongst the members and a resource coming from the pool is more valuable than the same amount of resource held individually. Imagine an ad hoc network where no agent agrees to transmit any data and keeps all energy to itself, then the whole system collapses.

We set the resource that each agent $a$ is able to contribute to a maximum of 1, i.e. it will decide to contribute any amount $c_a(t) \in [0, 1]$. For agent $a_i$ (where $i \in \{1, \ldots, |A_t|\}$) this results in a utility following the classic linear public good game (further properties of this game can be found in [16], for example):

$$u_{a_i}(t) = \frac{\alpha}{|A_t|} \sum_{a \in A_t} c_a(t) + \beta \left(1 - c_{a_i}(t)\right)$$

where $\alpha > \beta$ and $\frac{\alpha}{|A_t|} < \beta$.

The utility is a fraction of the pooled resource times its value giving variable $\alpha$, and the value that is gained from withholding the resource $(1 - c_{a_i}(t))$ is denoted by $\beta$.

Let $|A_t| = 2$, then the utility agent $a_1$ can achieve by contributing fully if agent $a_2$ does so as well is $u_{a_1} = \frac{\alpha}{2} \cdot 2$, but only $u_{a_1} = \frac{\alpha}{2}$ if agent $a_2$ contributes nothing. Given that the contribution of agent $a_2$ is unknown to agent $a_1$ at time $t$, $a_1$ will rationally choose to not contribute anything, resulting in $u_{a_1} = \beta$, which is more than $\frac{\alpha}{2}$, but less than $\alpha$. 

32
This is yet another tragedy caused by rationally behaving agents. The linear public good game can be summarised in two conflicting strategies:

- The strategy according to the Nash equilibrium is for all agents to fully free ride, i.e. not to provide any resource to the common pool. This leads to the highest individual utility which is independent of other agents.

- The Pareto efficient strategy is for all agents to contribute their entire available resources to the pool, i.e. not to withhold anything. This leads to a utility that is higher than for the Nash equilibrium but depends on the cooperation of all other agents to achieve its maximum.

Though these strategies apply to rational agents, several studies have shown that humans deviate from that behaviour and tend to contribute some percentage of their resource instead of either all or nothing. One reason for that are the variables $\alpha$ and $\beta$ that are usually not set globally, but are specific to the individual. This effect has been studied extensively in economic sciences (as presented in [125], for example) and policies for $m_t$ have been designed to enhance contributions (refer to [47] or [114], for example).

### 2.4.3. Hybrid replenishment

Hybrid replenishment occurs when there exist both exogenous and endogenous replenishment mechanisms. Most demand-side infrastructure management falls into this category such as smart grids. These are future energy grids where not only power plants provide energy (externally to the demand-side) but also individuals at the demand-side, such as households, can feed energy that they produce locally into the grid.

These systems rely on both sources of replenishment to balance the fact that one of the replenishment methods can be unreliable, such as (endogenously produced) solar power, and resource has to be taken on from elsewhere. The fact that external resources are not needed at all times leads to utility fluctuations on both sides that have to be taken into account.

### 2.5. Common-pool resource management

The types of open systems considered in this work operate roughly as follows. Firstly, agents form an opportunistic alliance; secondly, they decide on a policy that defines how to distribute the resources from the common pool; and thirdly, they execute the resource allocation for as often as possible or needed. The agents’ focus is to maximise their individual utility over the entire time frame, which means that a suitable policy
has to be found that enforces compliance with the rules so that the agents do not deplete the resource at an early stage.

2.5.1. Top–down (closed)

Traditional approaches to manage a common-pool resource (CPR) in human societies include privatisation and state control, as they can create incentives for maintaining a resource which is otherwise used inefficiently [109].

There are arguments to support the choice of state control over commons, such as Hardin’s rather gloomy paper about the tragedy of the commons. According to him, “Individuals locked into the logic of the commons are free only to bring on universal ruin;” and he argues further that “As the human population has increased, the commons has had to be abandoned in one aspect after another.” [52, p. 1248]. Sinn, on the other hand, argues for privatisation and states that “under well-defined property rights and correct price expectations, a competitive extraction industry brings about a Pareto-optimal depletion path, and it has been shown that oligopolistic and monopolistic market structures may even produce a bias towards conservation.” [109, p. 235].

2.5.2. Bottom–up (open)

Although often claimed, state control and privatisation are not the only way to successfully manage a CPR. Weitzman shows that there is a limitation to the mentioned inefficiency and that “The variable factor will always be better off with (inefficient) free access rights than under (efficient) private ownership of property.” [119, p. 225].

In order to avoid the tragedy of the commons and ensure a sustainable resource, the agents that appropriate the resource need to get organised, especially when the resource gets scarce. They need to define policies for drawing resources and enforce compliance with these regulations. Typically the appropriators will set up an institution that encapsulates all these rules and policies, and promotes cooperation. It is then possible that “the private objective of those with bargaining strength to alter institutions produce institutional solutions that turn out to be or evolve into socially efficient [institutions]” [83, p. 16].

There are a number of examples where the self-organisation of managing a common-pool resource with institutions is very successful as shown by a large variety of examples in Ostrom [85].

---

4 The variable factor can be, for example, the labour needed for the appropriation, so it directly feeds into the utility.
2.5.3. Sustainability issue

One of the main concerns in open systems is that of sustainability. It appears to be very hard to create enough incentive to sustain a resource over a long period if no entity can be held accountable for it. This is especially the case if the short-term benefit is negative, as it is for measures reducing our carbon footprint, for example. Institutions can create incentives to meet such a sustainability agenda through premiums or sanctioning mechanisms in the short run, and through the compliance with rules and the evolution of appropriate norms in the long run.

2.6. Institutions for CPR management

“Wealth that is free for all is valued by none because he who is foolhardy enough to wait for its proper time of use will only find that it has been taken by another.” [50]

This quote by Gordon summarises the dilemma outlined in Section 2.4.1. It reflects a short-sighted behaviour which has to be avoided for the benefit of the resource and future appropriations. As mentioned in the last section, people introduce institutions in order to regulate appropriation procedures and to reduce the uncertainty within these interactions [83].

2.6.1. Institutions

Elinor Ostrom conducted a substantial amount of research in self-governance of common pool resources [85]. She investigated different types of CPR management all over the world, from communal tenure in Swiss high mountain meadows over irrigation communities in the Philippines to Turkish inshore fisheries. For each case she analysed the strengths and weaknesses of the institutions that had evolved and what policies in particular caused either failure or success in managing the resource. She then published a catalogue of eight principles that, considered when formulating allocation policies, ensure the longevity of the common pool.

Due to multiple definitions and meanings of the term ‘institution’, we will follow the line of Elinor Ostrom who uses the concept of rules as a referent for the term. By rules she refers to “prescriptions commonly known and used by a set of participants to order repetitive, interdependent relationships” [84, p. 5]. Prescriptions specify what actions are required, prohibited or permitted. According to Ostrom, there are four steps to achieve predictability and order in a defined situation using rules. Firstly, by creating roles; secondly, by defining how agents take on or step down from these roles; thirdly,
2. Resource Allocation in Open Systems

by defining prescriptions for each particular role; and fourthly, by stating what outcome an agent is required, permitted or forbidden to affect. Another important aspect of prescriptions is that all participants are assumed to know the rules and can be held accountable when violating them.

In general linguistic use, the term institution often also refers to the organisation (e.g. of academic or social nature) that uses a specific rule set defining interactions within the organisational context. We will make this distinction apparent when it is important for the context, i.e. refer to an organisation (or alike) whenever we mean the group of people, agents, etc., and not the rules.

2.6.2. Rules

Institutions must be designed to allow for adaptation because some current understanding is likely to be wrong, the required scale of organization can shift, and biophysical and social systems change. Fixed rules are likely to fail because they place too much confidence in the current state of knowledge, while systems that guard against the low probability, high consequence possibilities and allow for change may be suboptimal in the short run but prove wiser in the long run.

Ostrom and Hess [87, p. 68]

Rules specify sets of actions an agent can choose from, which means they jointly affect the structure of a particular situation rather than the behaviour of agents directly [84]. As the quote implies, there need to be multiple levels to analyse and change how rules affect different behavioural aspects. In [85], Ostrom suggests three levels of analysis, according to three levels of rules, that are nested within each other, as shown in Figure 2.1:

- The first level concerns the operation. Actions based on operational choice have a direct impact on the physical world and include actions such as appropriation or provision of resources that are chosen from the operational rules.

- The second level of analysis concerns collective choice. Participants make policies and management decisions, and they adapt operational rules using collective choice. Examples for operational rules they change are including/excluding participants, strategies for appropriation, monitoring and sanctioning. How the participants' preferences are aggregated, is defined by collective-choice rules, such as majority or unanimity [86].

- The third level concerns the constitution. On this level is decided who is eligible and what rules are to be used for making collective choices. The constitutional rules define how and what constitutional choices can be made.
2.6. Institutions for CPR management

At any of these three levels of analysis, the rules specified by a higher level are assumed to be fixed during the time of decision making. Typically, operational choices are made routinely, whereas choices at higher levels take more effort and can be costly, hence rules are changed less frequently. The nesting of the rules leads to more stable strategies and more reliable expectations on the behaviour of participants.

2.6.3. Principles

For the successful management of a common-pool resource, appropriate rules have to be chosen for each of the three above mentioned levels. Ostrom mentions eight socio-economic principles that represent design guidelines for setting up an institution for CPR management [85, p. 90], as summarised in Table 2.1.

Principle **P1** defines the boundaries of the CPR, meaning how many households or individuals are authorised to withdraw from the resource, including closing the resource from unauthorised access. This solves the issue of ‘overcrowding’, one of the tragedies that commons involve according to Hardin [52].

Principle **P2** defines rules on the appropriation that put restrictions on time, place, technology and quantity. The rules have to be in line with the prevailing local environment and include additional aspects on provisions such as labour, material, etc.

Principle **P3** concerns the level of collective choice and states that the individuals which are affected by the operational rules should also have the rights to participate in the modification of these rules. This is a very important principle, as the local appropriators
## 2. Resource Allocation in Open Systems

Table 2.1.: Design principles for managing CPR institutions

<table>
<thead>
<tr>
<th>Principle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Clearly defined boundaries</td>
</tr>
<tr>
<td>P2</td>
<td>Congruence between appropriation and provision rules and local conditions</td>
</tr>
<tr>
<td>P3</td>
<td>Collective-choice arrangements</td>
</tr>
<tr>
<td>P4</td>
<td>Monitoring</td>
</tr>
<tr>
<td>P5</td>
<td>Graduated sanctions</td>
</tr>
<tr>
<td>P6</td>
<td>Conflict-resolution mechanisms</td>
</tr>
<tr>
<td>P7</td>
<td>Minimal recognition of rights to organise</td>
</tr>
<tr>
<td>P8</td>
<td>Nested enterprises</td>
</tr>
</tbody>
</table>

have usually much better knowledge about the condition of the resource and prevailing environment than external entities.

Principle \( P4 \) is about monitoring the conditions of the CPR and appropriator behaviour. This can be done by the appropriators themselves or by appointed agencies. The costs for monitoring can be paid by provisions mentioned in \( P2 \) and depends on the set of rules that is adopted in the institution, but is typically low for long-enduring CPRs.

Principle \( P5 \) concerns the sanctioning methods of the institution. Upon violation of a rule, a participant will become subject to graduated sanctions, depending on the severity and frequency of the violation. The presence of this principle leads to ‘quasi-voluntary compliance’ for the participants.

Principle \( P6 \) states that participants (and appointed agencies) need a facility for fast conflict resolution at a low cost. A point of contention can be a wrongful sanction initiated by the monitoring agency, for example.

Principle \( P7 \) protects the stability of the institution and stipulates that the rights to formulate their own rules has to lie within the appropriators and should not be challenged by external authorities, such as the government.

Principle \( P8 \) applies to CPRs that are part of some larger system and states that appropriate rules have to be established that solve potential conflicts between the different layers of nested CPRs.

In [84, p. 6], Ostrom writes that “Rules are the means by which we intervene to change the structure of incentives in situations.” That is an important basis for creating high compliance with the rules, predictability on the appropriable resource and ultimately satisfaction for all participants.
Defining what we mean by ‘institutions’ for CPR management and how to design a
good set of rules that enables the appropriators to successful self-governance is a first
step towards answering \( Q1 \) (see page 46).

2.7. Related approaches

In this section, we present related approaches for dealing with the problem of resource
allocation. These are computational social choice, including a selection of research work
on voting and fair division, and game-theoretic problems.

2.7.1. Computational social choice

Computational social choice builds upon two disciplines, computer science and social
choice theory. This leads to interesting interdisciplinary work that can solve problems
arising in both separate fields [28]. Firstly, techniques from artificial intelligence can be
used for solving questions that arise in the domain of social choice; secondly, techniques
from social choice theory can be used for solving questions that arise in the domain of
computer science and AI. Examples include the management of autonomous (software)
agents using voting protocols, or fair division for resource allocation [61]. These typically
consider topics from further areas such as welfare economics, game theory, multi-agent
systems, computational logic and operations research.

A typical problem is to define techniques and models with desirable properties, such
as fairness or optimality, according to some specific definition [60]. These definitions
include the optimisation of utilitarian social welfare (a fairness measure that is most
commonly used for multi-agent systems), envy-freeness or Pareto optimality. Topics of
interest to computational social choice include complex aggregation rules, voting rules
that are robust to manipulation, or resource allocation. Much work has been carried out
to analyse the computational complexity of these techniques [27]. When adapting them
for multi-agent systems there are important aspects to consider. For example, many of
the defined techniques assume that all agents are able and willing to follow the rules, an
assumption that cannot be guaranteed in open systems considered in this work.

2.7.2. Voting mechanisms

Voting mechanisms are one way of making social choices. In general, voting does not
involve any payment, and depending on the mechanism, agents not only express their
preference on a single choice but can also rank their choices in order of preference [31].
Let there be three alternatives $X$, $Y$, $Z$ to choose from, then an agent preferring $X$ over $Y$ and $Y$ to $Z$, would vote $X \succ Y \succ Z^5$. Aggregating the preferences of several agents to determine a single winner (e.g. a restaurant to go to, or new head of group) depends on the chosen voting mechanism.

Five examples of voting mechanisms, instances are presented in [20] or [92], are:

- **Plurality**—Each agent votes for one alternative and the alternative with the most votes wins.

- **Runoff**—Each agent votes for one alternative in the first round, but the two most voted alternatives enter a second round (unless one of them has the majority of votes already). Each agent votes for one of the two alternatives, and the alternative with the most votes wins.

- **Borda count**—This time, each agent has to rank $m$ alternatives in order of preference. The alternatives are then assigned **Borda points** as follows: $m$ for the alternative ranked first, $m - 1$ points for the alternative ranked second, and so on. The Borda points are then aggregated over all agents for each alternative, and that sum is called **Borda score** of that alternative. The alternative with the highest Borda score wins.

- **Instant runoff**—Each agent ranks the alternatives in order of preference. The weakest alternative (that got top-ranked by the least agents) is eliminated from the set of alternatives. This mechanism is repeated until one alternative has the majority of votes.

- **Approval**—Each agent selects a subset of alternatives, and the alternative that is in these subsets most often wins.

Any voting mechanism for three or more alternatives is either manipulatable or dictatorial\(^6\) as argued by [49] and [103]. This means that agents might not reveal their true preferences, but rank alternatives in such a way that it manipulates the outcome. Consider for example a plurality voting mechanism (as presented in [20, p. 72]) with the additional condition that in a tie, the alternative with lexicographic lower order wins. Assume we have four agents. The first two agents vote $Y \succ Z \succ X$, the third agent votes $X \succ Y \succ Z$, and the fourth agent’s real preference is $Z \succ X \succ Y$, meaning that it prefers $X$ over $Y$. That last agent’s top ranked alternative ($Z$) is not going to be the winner. In order to manipulate the outcome, the last agent could state $X \succ Z \succ Y$ as its preference and so make $X$ the winner (which is ranked better than $Y$) due to the existing tie rule.

---

\(^5\)We assume a transitive and strict ordering in the preference relation.

\(^6\)A voting mechanism is called ‘dictatorship for agent $a$’, if the top ranked alternative of $a$ always wins and other agents preferences are not taken into account [20].
Voting mechanisms are examples of social choices without payments. If payments are involved in the decision-making process, auctions can be used. They typically require an agent in the role of the auctioneer, a central entity that collects the preferences of agents in form of bids. A number of protocols that can be used are presented in [27].

For auctions and voting mechanisms in multi-agent systems (as in a social context), it is important to clearly define the rules of the mechanism, what agents are eligible to vote, what agents are eligible to count the votes, and what agents are eligible to declare the winner. To this end, Jones and Sergot [59] logically formalised the concept of institutionalised power, stating what agent is empowered, permitted or obligated to perform what action in what role within a defined context.

2.7.3. Fair division

Fair division is another example from the field of social choice and has been of great interest to economists and mathematicians, as presented in [20] or [95], for example. The typical problem is to “endow individual agents with a suitable set of rules determining their willingness to accept certain deals such that resulting allocations will satisfy our fairness or efficiency criterion of choice” [39]. There are divisible and indivisible goods, in this work we only consider the divisible type. The allocation of indivisible goods can be solved with techniques from combinatorial optimisation, for example. A divisible good can be homogenous, when each part is valued equally (e.g. money as in [24]), or heterogeneous, when different parts can be valued differently (e.g. a cake with two different flavours or a piece of land with varying vegetation as in [23]).

When dividing goods, we can consider various fair or efficient criteria that are quantifiable with a utility function. One such example is the collective utility or social welfare. Let $n$ be the total number of agents, the utility by agent $a$ on its share $r_a$ be $u_a(r_a)$, and $P$ be the entirety of goods allocated as $r = (r_1,...,r_n)$. Then, possible aggregation functions include (for further functions refer to [27]):

- **utilitarian social welfare**, the sum of individual utilities:
  $$\sum_{a=1}^{n} u_a(r_a)$$

- **egalitarian social welfare**, which is equal to the lowest utility of any one agent:
  $$\min \{ u_a(r_a) : a = 1,...,n \}$$

---

41

---

7Here we will not consider the ordering of goods, i.e. what good is preferred over another.
2. Resource Allocation in Open Systems

- **elitist social welfare**, which is equal to the highest utility of any one agent:
  \[
  \max \{ u_a(r_a) : a = 1, \ldots, n \}
  \]

- **Nash product**, which is the product of the individual agent utilities:\
  \[
  \prod_{a=1}^{n} u_a(r_a)
  \]

These functions give information on the distribution of goods (or utility rather), but they do not give information on whether the distribution is accepted as fair by the individuals.

There are further fairness or efficiency criteria that can be considered for division, for instance as presented in [20, 23, 39]. Let the notation be the same as above and \( b \) an agent different from \( a \). Then a division is

- **Pareto optimal** (or **Pareto efficient**), if there is no other division \( r' \) possible so that at least one agent is more satisfied without dissatisfying any other agents (i.e. decrease their utility):
  \[
  \not\exists a, r' : u_a(r'_a) > u_a(r_a) \land u_b(r'_b) \geq u_b(r_b) \quad \text{for all } b
  \]

- **envy free**, if no agent feels that it gets less than any other agent:
  \[
  u_a(r_a) \geq u_a(r_b) \quad \text{for all } a, b
  \]

- **equitable**, if an agent’s individual utility is the same as for all other agents:
  \[
  u_a(r_a) = u_b(r_b) \quad \text{for all } a, b
  \]

- **proportional**, if the utility an agent receives is at least the utility of the whole good divided by the number of agents:
  \[
  u_a(r_a) \geq \frac{1}{n} \cdot u_a(P) \quad \text{for all } a
  \]

The above criteria are of special interest in multi-agent systems and can be used to define goals of an allocation. Which criterion is used depends on the application itself and additional criteria might be required.

---

*For a meaningful interpretation the utilities should be positive.
*Here the utility function has to be monotonic, i.e. for \( r_a \leq r'_a \Rightarrow u(r_a) \leq u(r'_a) \).
Much research on fair division uses the example of cake cutting [23], where the cake represents a divisible, heterogeneous good to be shared between $n$ agents. A procedure for two agents cutting a cake works as follows. The first agent cuts the cake into (for this agent) two parts of equal utility and the second agent chooses its preferred piece. This procedure is called ‘cut-and-choose’ and results in a Pareto-optimal and envy-free outcome, equitability cannot be ensured.

For more than two agents, several different procedures have been developed to ensure either envy-freeness or proportionality. There are further properties of such procedures that are important for deciding which procedure to use in a specific situation. Examples are the complexity of the procedure, or whether a procedure can be expressed as an algorithm with discrete steps. Furthermore, the cake can be divided in contiguous or discontiguous pieces, the number of cuts can be minimal or at least have an upper bound, and the procedure may require a ‘referee’. For $n = 2$ the cut-and-choose procedure is simple and ideal with respect to these properties [39].

We now present a few procedures of fair division for more than two agents ($n > 2$), see also [25]. Proportional procedures include:

- **Banach-Knaster last-diminisher procedure** — The first agent $a$ cuts off a piece of cake (representing $\frac{1}{n} \cdot u_a(P)$), then that piece is offered to each agent in turn. If the piece is considered too small by an agent $b$, this agent passes the piece on. Otherwise $b$ trims the piece down to $\frac{1}{n} \cdot u_b(P)$. In case the piece gets back to agent $a$, this agent has to take the piece. The remaining $n-1$ agents then divide the leftover cake (including trimming) following the same steps.

  This procedure guarantees a proportional but not an envy-free outcome. There is no external referee needed and the number of cuts required for full division is bounded by $2(n-1)$. If the agents merely indicate the cut they are going to take, contiguous shares are possible.

- **Dubins-Spanier procedure** — This procedure requires a referee that moves a knife from left to right over the cake. At any time, an agent can indicate that it wants a cut to be made, and this agent takes the piece of cake to the left of the knife. The remaining $n-1$ agents continue with that procedure until there is only one agent left which takes the leftovers.

  This procedure is also called ‘moving-knife procedure’ and guarantees a proportional but not envy-free outcome, using the minimal number of cuts ($n - 1$). It is not possible to construct a realistic algorithm for this procedure, but with a smooth utility function, good approximations on a discretely moving knife are possible.

Envy-free procedures are more difficult to achieve than the above mentioned proportional procedures. As agent $a$ cannot influence the share of other agents once $a$ was allocated,
and agent $a$ is uncertain about other agent’s utility functions, it is possible that a subsequent agent receives a piece of cake that $a$ would prefer to its own.

Here are two examples of envy-free procedures with $n > 2$:

- **Selfridge-Conway procedure**—Consider three agents $a$, $b$, and $c$. Agent $a$ divides the cake into three (in $a$’s measure) equal pieces. Two cases follow from this. Firstly, if $b$ thinks that two pieces are tied for largest, the agents pick a piece in order $c$, $b$, $a$, and the procedure finishes. Secondly, if $b$ thinks one piece is larger the two others, $b$ can trim that piece to obtain two pieces that tie for largest. Again, the agents pick a piece (not the trimmings) in order $c$, $b$, $a$, where $b$ is required to take the trimmed piece if $c$ did not take it. Then, the agent that took the untrimmed piece ($b$ or $c$) cuts the trimmings into three (in that agent’s measure) equal pieces, and the agents choose in order non-cutting agent, agent $a$, cutting agent.

This procedure guarantees an envy-free and also proportional outcome. The number of cuts needed is at most 5, but the pieces can be discontiguous.

- **’Brams-Taylor procedure’**—Consider four agents $a$, $b$, $c$, $d$. Agent $a$ cuts the cake into four pieces, cuts are always assumed to be equal in the cutting agent’s measure. This agent then distributes the pieces unless an agent objects. Assume agent $b$ objects, then $a$ and $b$ follow a well-defined process to create six sets of cake that are used for a 4-person cut-and-choose sequence. This sequence is repeated until agent $b$ has advantage over agent $a$, so $b$ does not envy $a$. This whole procedure is then repeated at most once for each pair of agents, which can be done in a finite amount of steps. “The extension to arbitrary $n$ is fairly straightforward and left to the reader” [25, p. 15].

The above description is merely an outline of the algorithm that is required for achieving an envy-free division, but we can see that it quickly becomes very complex for higher values of $n$.

There are several procedures of fair division that can minimise envy, either with respect to envy between pairs of agents, with respect to the accumulated envy of an agent, and so on, as argued in [39].

In this work, the allocation procedure of the ‘cake’ is slightly different from the assumptions made above, and the ‘game’ is played over multiple rounds. There can be unwanted resources in a round (therefore no envy), and a fair division in every single round is not necessary, a fair allocation over a certain period of time will be sufficient. Still, for the case when the resource is sparse (i.e. not all agents can be allocated the amount of resource they request), such game-theoretic procedures can be taken into account.

---

10 In their paper, Brams and Taylor call this procedure ‘Envy-Free Protocol for Arbitrary $n’."
However, the computational complexity for a 100 agent\textsuperscript{11} envy-free division would be impractical for the amount of intended repeats.

2.7.4. Game-theoretic example

The closest way we found to address the problem of resource allocation in this work from a game-theoretical perspective is to use a proportional allocation mechanism. This mechanism assumes complete knowledge and centralised decision making, and can be formalised as follows (refer to \cite{58} for details):

\[
\begin{align*}
\text{maximise} & \quad \sum_a u_a(r_a) \\
\text{subject to} & \quad \sum_a r_a \leq P \\
& \quad r_a \geq 0, \ a \in S \subseteq A
\end{align*}
\]

where \(u_a(r_a)\) is the utility\textsuperscript{12} of agent \(a\) from the subset of agents \(S\) that were allocated an amount \(r_a\). The sum of utilities is to be maximised, under the condition that the sum of allocations has to be below the total resource level \(P\), similarly to the formulation of utility in Section 2.4.1 for exogenous resource allocation.

Depending on the individual utility, an agent makes a bid for an amount of resource (resource units are equally priced). We consider the case when an agent anticipates the influence of their bids and adjusts the bid accordingly. There exists a Nash equilibrium where the distributor allocates as much resource as possible whilst maximising all agents utility, more details can be found in \cite{58}. We can then compare the outcome of the equilibrium to the theoretical optimum and find that the price of anarchy\textsuperscript{13} is tightly bound to 3/4. We can interpret this result as investing 1/4 of the optimal utility for finding an agreement that does not deplete the common pool.

If we do not impose any restrictions on the mechanisms (i.e. there is complete information and no rule violation), the price of anarchy can take values closer to 1, as stated in \cite{100}. This assumption, however, cannot be made in the scope of this thesis, therefore this game-theoretic approach cannot be used. Furthermore, in the above game the allocated resource is considered to be the appropriated resource. Cheating agents are not taken into account, which further restricts the applicability of the game. The institutional approach has measures in place that defend the pool against illicit appropriation and we

\textsuperscript{11}That is the amount we use for simulation, see Chapters 4 and 6.
\textsuperscript{12}This function is concave, strictly increasing, continuous and differentiable.
\textsuperscript{13}The price of anarchy here is defined as the worst game-theoretic outcome divided by the optimal outcome, in some literature the definition is the inverse.
will see later that we can achieve a lower price of anarchy (i.e. higher efficiency) for the system operation.

2.8. Summary and Problem Specification

In this chapter, we presented different types of open systems and different types of resource replenishment. We discussed how the problem of resource allocation is managed in human societies and presented an approach that we will adapt for use in open systems considered in this work. This approach is about the evolution of institutions for CPR management and characterised by a collection of rules, created by the appropriating parties of a resource, that operate on three different levels and follow eight principles for the design of those rules.

We then presented approaches from the literature that can be used to manage resource allocation in open systems. These are computational social choice and game theory. We argue that, for the type of systems considered in this work, envy-free fair division is not applicable due to the high complexity of the division procedures, and the presented game is not applicable due to the requirement of perfect information and fully complying agents at all times.

The subject matter of this thesis is exogenous resource replenishment where the common pool is refilled by the environment and not the agents themselves. We consider the case where no human intervention in the allocation process is required and the system acts fully autonomously.

This leads us to our first research question:

\[ QI \]
Can the problem of resource allocation in open systems be addressed by modelling an institution for self-governance?

We will examine the effect that the use of institutions in an open management approach has on the sustainability of the resource, and we develop a testbed for this resource allocation scenario. Agent based modelling is particularly suitable for simulating social interactions. To implement the testbed, we use Presage\(^2\), which is a time driven simulation platform based on Java that allows for rapid prototyping of agent societies. We extend Presage2 with Drools\(^5\), a business logic integration platform, to be able to formalise the institution in a rule based environment.


3. Self-Organisation

3.1. Introduction

In this chapter, we firstly describe the methodology of sociologically-inspired computing that allows us to integrate theories from social science into our computer model of open systems. Then, we introduce a framework for dynamically changing the specification of an open system at runtime that uses a multi-level protocol stack. This framework is then compared to Ostrom’s approach of institutional change using multiple levels of analysis [85]. We can now carry out the first step of the SIC methodology. To that end, we discuss what roles, rules and methods can be defined for the implementation of the open system and show how the rules can be nested within each other. Subsequently, we instantiate the formal model that is used for management of a common-pool resource in self-organising open systems and present the simulation platform we choose to implement this model. We then present Drools Expert [94] that uses a production rule system for expressing and evaluating rules in the computer model. Finally, we give the formal axiomatisation of the first six design principles for creating institutions for resource management, which also answers the question whether it is possible to encode Ostrom’s design principles using norm-governed systems. Up to Section 3.6, we based most of this chapter on our work described in [93], the formalisation of principles (from Section 3.8) is presented with regards to the implementation in Drools.

3.2. Sociologically-inspired computing

Social sciences are a great source of inspiration when trying to solve problems that apply to open systems. Social systems are often concerned with the emergence and maintenance of roles, power-relations or institutions, which are issues that have been studied by the social sciences and formalised into concepts [38]. However, in most cases it is not possible to directly apply the findings of these sciences to artificial systems. Therefore, researchers developed methodologies that formalise and adapt these findings and bring them into appropriate shape for experimentation. One of these methodologies
3. Self-Organisation

for sociologically-inspired computing, that we previously presented in [93], follows the synthetic method of Steels [111] and will be revisited here.

Steels’ methodology is based on the inductive methods used in most scientific research. It takes as input observed phenomena (not limited to social sciences) and three types of theories that result from these observations. The first type is observational theory which analyses the nature of the observed phenomena and is used for classifying and discovering patterns. The second type is mechanistic theory which defines the techniques, structures and processes that cause the observed phenomena. The third type is explanatory theory which is concerned with the underlying reasons of observed phenomena and what made them occur in the first place. These theories are engineered into a counterpart to the real world that allows for customised experimentation. The output of these experiments is compared with the observed phenomena, and either theory or counterpart revised. These counterparts can be anything from a mathematical predictor, over computer models to artificial systems. There needs to be a valid link between the counterpart and the original context, but the counterpart does not need to have the same functionality in every aspect.

Advantages of this method include the systematic exploration of variations that would not be possible in the original context, such as in evacuation scenarios. In some cases, the described method can be used to substitute functionality from the original context should that fail, e.g. in the field of prosthetics. Our aim is to apply such a method to artificial societies such as open systems, rather than modelling human societies using real-world data.

There are various methodologies in the literature that resemble the idea of Steels, for example the Prometheus methodology, a step-by-step guide to assist developers of multi-agent systems in design, documentation and building phase [88], or the CoSMos process [4], which also provides a link from theory to computer model. The Prometheus methodology is specifically designed for BDI-agents. It evolved out of industrial experience and encapsulates the necessary details for developing intelligent agent systems, so that it can be used by non-experts. The CoSMos project looks at the use of simulations for scientific research and for engineering, especially swarm robotics. Their approach focuses on agent-based simulations where complex behaviour emerges from interactions within the environment. They describe three phases that are required for engineering a simulation platform. These are the discovery phase, the development phase and the exploration phase. Both methodologies, however, are less general in their use cases than Steels’ methodology.

In Figure 3.1 we define the process from observed phenomena to observed performance for our computer model. We start with the observed phenomena, such as a (human) or-

\footnote{For example, we don’t attempt to model the full spectrum of human capabilities.}
3.3. Dynamic specification of open systems

In this section, we introduce the framework of Artikis [8] that we will use for changing specifications of open systems at runtime. First, we present the type of agents in open systems, some alternatives for formalising institutions in open systems and the concept of institutionalised power. Then, we discuss how to perform a rule-based specification change using a multi-level protocol stack.
3. Self-Organisation

3.3.1. Agents for open systems

Agents are computational entities that represent a physical object or ideological component, such as people, organisations or institutions. The agents have the same (or for the experimentation essential) functionalities and internal structures, and are bound by the prevailing laws in their environment. To navigate within the environment and to decide what actions to perform when, agents are equipped with the ability to sense, interact with their surroundings and pursue some goal, to different degrees of automation.

*Autonomous agents* are agents that can dynamically adapt to consequences of their own actions within the environment. This definition stems from Smithers [110] which we adopt here. The difference to an automatic system is that “it is not clear, not even to the original designer, how a system will respond because it has precisely been set up so that responses evolve and change to cope with novel situations” [111, p. 85]. Accordingly, autonomous systems cannot be controlled in the same way as automatic ones.

Structural reorganisation is crucial for multi-agent systems in an open and dynamic environment. The agents have to be able to evaluate and decide on an appropriate organisational and institutional structure depending on that environment (as argued by [36] or [72]), hence autonomy is a requirement for agents considered in this work.

3.3.2. Institutions for open systems

There are several approaches to leverage the concept of institutions for use in computer science. In [41], a (human) institution of a fish market is extended to the notion of electronic institutions, where autonomous agents interact with each other. These interactions are realised by message interchanges and can modify the commitments or obligations of participating agents, depending on their role in the institution. They describe a declarative language, ISLANDER [40], for specifying the components of the institution: the structure, the scenes\(^2\), the rules and the roles. The specified electronic institution can then be executed with the software platform AMELI [5]. Other languages that can be used to formalise electronic institutions include LAO (Logic of Agent Organization) [35] or the Event Calculus [64].

The type of electronic institution considered in this work can change its specification at runtime. We follow the approach of [8] on dynamic specifications for multi-agent systems, as explained below.

---

\(^2\)For each type of activity that happens in the context of an institution, there is a corresponding scene where the actions take place.
3.3.3. Institutionalised power

In order to apply and change the rules in an institutional setting, the agents need to know to what rules they are allowed to use in what context and what role an agent should occupy to be eligible to make changes to a specific rule. We can use the concepts of role, role assignment [102] and institutionalised power to meet these requirements.

*Institutionalised power* refers to agents being empowered to perform specific actions in an institutional (or norm-governed) setting depending on the role they occupy and the environment. This concept is usually combined with the concepts of permissions, obligations and practical possibilities, these ideas have been explored in [59].

It is therefore necessary to define protocols that deal with role assignment, in order to appoint a specific agent to a role. If the appointed agent leaves the system, performs incorrectly or is unable to fulfil the obligations connected to that role, it must be possible to change which agent occupies that role. To achieve such a change in roles (and rules) during runtime, we need a dynamic specification of norm-governed systems.

Other approaches to role-based collaboration and coordination in multi-agent systems are reviewed in [26]. [122] presents a methodology to examine the implications on the society (macro-level) and the agents (micro-level) of systems with multiple interacting roles, however, their organisational structure is assumed to be static.

3.3.4. Rule-based specification change

Agents in the type of open systems considered here, are modelled according to characteristics and behaviour that are borrowed from theories in social science, meaning that in “artificial agent societies, the designers can impose these norms on the agents” [104]. This can be achieved by the use of norms. Following [117], their function is to eliminate selfish or malicious behaviour without dictating the design or restricting the autonomy of the individual agents.

Typically this is done by using norms that result in a particular agent behaviour, and the agents furthermore have to be able to develop new norms. Several implications of norms for compliant behaviour can be found in [12] or [123].

For simulation, legal, social and organisational systems have frequently been formalised in terms of norm-governed systems. Artikis developed a dynamic executable specification for those types of systems in open agent societies, which is used to modify protocols at runtime [8]. His framework has three components: a norm-governed system specification, a protocol-stack that defines how to change the specification, and a topological space that expresses the ‘distance’ (e.g. cost to move) between any two specification instances.
An advantage of this framework is, that the distinction between physical capability, institutionalised power and permission is maintained and we can express five social constraints with a dynamic specification. These constraints are the physical capabilities, the institutionalised powers, the agents’ obligations, prohibitions and permissions, sanction and enforcement policies that deal with violations of required, prohibited or permitted actions, and the designated roles of empowered agents.

The modification of a specification in this framework is facilitated by a communication language. In order to modify a protocol at some object level, the agents can initiate the protocol change from some higher level, the meta protocol. To change this meta protocol, they can go to the meta-meta level, and so on. The hierarchy of up to \( k \) levels is shown in Figure 3.2, and from some level \( l \), any level that lies below can be changed.

![Figure 3.2.: K-level protocol stack for dynamic specifications](image-url)

The specification points of a protocol that can be changed are the degrees of freedom of the system, and any one set of specification points is a specification instance. The rules that define under what circumstances the agents are allowed to initiate a change on these specification points from some higher level are stated in transition protocols. These transition protocols also define what roles the agents have to occupy to initiate the change and what points (or degrees of freedom) can be changed with the execution of the \( l \)-level protocol.

This leads us to our second research question:

\[ Q2 \] Can Ostrom’s design principles be encoded in norm-governed systems?
3.4. Relating the work of Ostrom and Artikis

In this section, we describe how institutions for enduring CPR management can be formalised with the help of a dynamic specification. Figure 3.3 illustrates how we can relate the concepts of institutional change in human societies to dynamic specifications in open systems.

As an example, let us consider the issue of winner determination at the middle level (operational choice or meta level) for both the institution and the dynamic specification. Assume that there are several degrees of freedom for a winner determination method, e.g. majority or Borda count, that define how votes are being aggregated.

Firstly, consider the dynamic specification on the right side and an ‘active’ winner determination method. This method indicates how votes have to be counted should the decision to change a specification point on the object level be made (by another protocol in the meta level). In order to change the winner determination method used in the meta level, a protocol on the meta-meta level has to be initiated. This can be done by following the rules stated in the corresponding transition protocol.

Then, we consider the winner determination method on the middle level of the institution, the collective choice. The ‘active’ method itself is stated in the rules, modifications
3. Self-Organisation

to rules on the operational level are carried out by choices on the analysis side that are bounded by the rule side. Modifications to the winner determination method can be made by the analysis side on the constitutional level, but whether this can be done is stated in the corresponding rule side (constitutional).

We can detect three categories of protocols: transition protocols, rule protocols and protocols that lead to actions. For institutions, the ‘action protocols’ lead to effects on a lower level or the environment, for dynamic specifications effect is initiated by ‘downwards’ transition protocols.

The main difference between both methods is, that in the institutional case, transition protocols and rule protocols are gathered on the rule side (the higher level for transition protocols) and the action protocols represent the analysis side. In the dynamic specification space, the rule protocols and action protocols are gathered in the level protocols, leaving the transition protocol as a protocol to act between the levels.

The good news is that, although there is a slight conceptual difference in the two level structures, the nesting of the levels has as effect that they will produce the exact same output. Hence this is a good leap forward in answering Question Q2.

3.5. Formal characterisation

In this section, we discuss the processes we need for the operation of an open system for common-pool resource management. Table 3.1 shows at what level of the institution or dynamic specification these processes take place, the rules needed for these processes are explained after.

The processes on the institutional side (left) are stated as defined by Ostrom [85, p. 53], examples of processes on the protocol side are stated as defined by Artikis [8]. The processes (i.e. the sets of rules) for appropriation, monitoring and enforcement in the institutional context are used for operating on a daily basis and can be found on the lowest level. The corresponding level on the protocol stack is the objective level, where we can define processes for resource allocation or exclusion (for rule enforcement), but also other methods like monitoring or sanctioning are possible. The next institutional level contains processes for policy making, adjudication and management, which correspond to the processes of rule configuration, dispute resolution and role assignment on the meta-level of the protocol stack. From a computational perspective, governance and formulation of rules represents a very sophisticated form of reasoning and requires some form of invention. For this reason, we do not formalise a corresponding process on the protocol stack. Furthermore, the decisions on an initial set of rules are be made by the system designer.
Table 3.1.: Example processes at different levels in institution and dynamic specification

<table>
<thead>
<tr>
<th>Institutional Levels</th>
<th>Protocol Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Governance Formulation</td>
<td>Constitutional Choice ⇔ Meta-Meta-Level Protocol</td>
</tr>
<tr>
<td>Policy Making Adjudication Management</td>
<td>Collective Choice ⇔ Meta-Level Protocol</td>
</tr>
<tr>
<td>Appropriation Monitoring Enforcement</td>
<td>Operational Choice ⇔ Object-Level Protocol</td>
</tr>
</tbody>
</table>

In the following, we define the sets of institutional rules that are needed to perform the resource allocation, and the roles that empower appointed agents to perform specific actions to change facts in the institution or in the environment.

### 3.5.1. Roles

According to Ostrom, creating roles is the first step one should take in a situation using rules, as explained in Section 2.6.1. There are four roles that can be assigned, namely member, head, monitor and gatekeeper. The role member is the standard role for an agent and to signal membership to an organisation, in order to take part in a resource-allocation process. The role gatekeeper controls this membership by assigning the role of member to agents. The monitor performs the monitoring process and reports conspicuous behaviour. The head is empowered to revoke the role of member, for example in cases of misbehaviour, and to assign the roles of gatekeeper and monitor. Furthermore, the head conducts voting procedures and performs the resource allocation according to the operational rules, for example. It is possible for an agent to hold more than one role at a time, for example the head, gatekeeper and monitor will always also be a member of the same organisation. There is one default agent role that is not assigned, which is the role of non-member, meaning that this agent does not belong to any organisation. There are no institutional duties attached to this role, but a non-member can still alter physical facts of the resource.

The roles of head, gatekeeper and monitor are in charge of ‘defending’ the resource against illicit access. We contend that the creation of these roles does not contradict the
openness of the system, as the operation of these roles merely ensures that the principles for self-organising resource allocation can come into effect. They do not discriminate against agents that want to join the system as long as the system can ‘afford’ more members (that comply with the rules) and so ensure the longevity of the system.

3.5.2. Rules and methods

In order to structure the rules used in this work, we define five methods that are invoked by these rules. In other words, each of these methods represents a subset of rules that serve a specific purpose. The five methods are acMethod, exMethod, raMethod, adrMethod and wdMethod.

The first one, acMethod, is the method for access control that defines how the process of granting access to an organisation is executed. Typically, agents apply for access to the gatekeeper who determines whether to assign them the role of member or not. There can be several rules present for access control with a further rule that states which one to use. Examples for acMethod are ‘attribute based’, where the applicant has to satisfy some qualification criteria and is automatically admitted, or ‘discretionary’, where the applicant must satisfy the gatekeeper’s criteria who is acting on behalf of the organisation\(^3\). This method also includes monitoring of agents that are not granted access to the organisation.

The second method is exMethod, the exclusion method that is used by the head to revoke an agent’s membership or to (temporarily) exclude a member from appropriating. Both exclusions are evoked by rules that regulate monitoring and sanctioning procedures. These rules specify, for example, how and how often a member is monitored or the dependency between grade of sanctioning and offence.

Thirdly, the raMethod defines rules for the actual resource allocation. These include rules about how to partition the resource, for example largest or smallest allocation first, taking equal rations, form a priority queue, etc. This corresponds to the function \(m_t\) from Section 2.4.1.

The adrMethod, alternative dispute resolution method, prescribes how to resolve conflicts in the organisation. Depending on the nature of the conflict or institutional level where it occurred, there are different methods in place to give the involved parties the chance to settle their dispute.

Finally, there is wdMethod, the winner determination method that has to be defined. At the constitutional level, this method is assumed be fixed, but by constitutional choice,

\(^3\)A well-known example of organisations that choose discretionary access control as their acMethod are London’s night clubs.
the agents can decide to change the \textit{wdMethod} in the collective-choice rules, that are used to aggregate votes for collective choices. Examples for different winner determination methods are plurality, runoff, Borda count, approval voting, etc.

\textbf{3.5.3. Nesting of rules, roles, methods}

The first three methods are part of the operational rules, whereas \textit{adrMethod} and \textit{wdMethod} are part of the operational-choice rules, and \textit{wdMethod} is also part of the constitutional rules. Figure 3.4 and 3.5 show examples for the nesting of the above mentioned roles, rules and methods.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3_4.pdf}
\caption{Nesting of rules for resource allocation}
\end{figure}

The first example (Figure 3.4) shows some of the rules that influence a resource-allocation process. At the constitutional level, there is a fixed \textit{wdMethod} and a set of votes (from a set of designated agents voting on a method for winner determination). The \textit{head} aggregates the votes and declares the winner. This process is an instance of constitutional choice and alters the \textit{wdMethod} at the collective-choice level.
3. Self-Organisation

At the collective-choice level, there is the (new) \textit{wdMethod} and another set of votes (from a set of designated \textit{member} agents voting on a method for resource allocation). Again, the \textit{head} aggregates the votes and declares the winner, and this collective choice alters the rule denoting the \textit{raMethod} at the operational level.

To determine the actual resource allocation at the operational level, the \textit{head} takes a set of member demands and applies the (new) \textit{raMethod} which usually depends on the available resource in the environment. So far, the rules only altered institutional facts, but the allocation from this last operational choice empowers the agents to perform the appropriation of the allocated resource, i.e. the alteration of the actually pooled resource.

![Diagram of rule nesting for role assignment]

Figure 3.5.: Nesting of rules for role assignment

The example in Figure 3.5 shows the rule nesting for role assignment. At the constitutional level, the same procedures take place as in the first example. This time, designated agents decide on two \textit{wdMethods} to be changed at the collective-choice level, again declared by the \textit{head}. One of these \textit{wdMethods} is used to determine the winner of a voting procedure for the (new) \textit{gatekeeper} and the other \textit{wdMethod} is used to determine the winner of a voting procedure for the (new) \textit{acMethod}. Both processes take place at the collective-choice level and both winners, role and rule, are updated at the operational level. The \textit{gatekeeper} now performs access control according to the \textit{acMethod} in place, and determines for each applying agent that is not a member of the organisation whether to grant access and assign the role of \textit{member} or not.

We will show later how the methods presented here can be formalised for use in the computer model. First we choose an appropriate formalism, as mentioned in Section 3.2.
3.6. Open systems model

In the next step of the formal characterisation, we will instantiate a formal model for the management of a common-pool resource. The idea is that the agents will form into clusters (or organisations) and set up an institution. This institution corresponds to a specification instance of the dynamic specification for norm-governed systems and will specify how to manage membership, how to allocate resources, how to enforce the policies, and so on.

The characterisation for the complete multi-agent system $S$ is defined by

$$S_t = \langle A, L, d, E \rangle_t$$

where at time $t \in \mathbb{N}$ (which is omitted if clear from context):

- $A$ is the set of agents,
- $L$ is the set of dynamic norm-governed system specifications, representing a specification space,
- $d$ is the distance function on $L$,
- $E$ is the environment of the system.

The distance that $d$ defines between two specification instances can be regarded as information about whether a change from one instance to another is feasible or advisable. The environment of the system contains facts about the number and state of common-pool resources accessible to the agents, and facts about what clusters have formed to manage these resources. The first category of facts is called brute facts and denotes physical states, whereas the second category describes the institutional facts that are conceptual, but not physical facts.

Now let $C$ be a cluster of agents that manage a particular resource. The formal model for each cluster is:

$$C_t = \langle M, I, \epsilon \rangle_t$$

where at any time $t$ we have:

- $M$, the set of *member* agents so that $M \subseteq A$,
- $I$, the institution as a specific instance $I \subseteq L$, and

---

4There might be no or no direct path from one instance to another, or following the direct path might be too costly to change.
3. Self-Organisation

- $\epsilon$, the local environment as a set of $Bf$ and $If$.

The institutional facts $If$ of the local environment $\epsilon$ describe values that are determined by the conventional state, including roles assigned or the amount of resource allocated to a particular cluster member. The brute facts $Bf$ include ‘physical’ facts such as the resource level of the common pool and the agents’ appropriations, for example.

In order to illustrate the model consider the following example. We have a set of agents $A = \{a_1, a_2, a_3, b_1, b_2\}$ and two common pools in the environment $\{P_1, P_2\} \in \mathcal{E}$. Let furthermore $\{raMethod, wdMethod\} \in \mathcal{L}$, with the resource-allocation methods $raMethod = \{ration, queue\}$ and the winner-determination methods $wdMethod = \{plurality, Borda count, instant runoff\}$. The distance function $d$ can be interpreted as a cost to switch from one specification instance to another. For example, depending on what $wdMethod$ is used, it might cost different amounts to perform a voting procedure in order to get from $raMethod = ration$ to $raMethod = queue$. Furthermore, $\mathcal{L}$ should contain rules that specify under what conditions a change to those methods can be initiated.

Now assume we have two groups of agents, and each group forms a cluster to perform a resource allocation using one of the pools. The members of the first cluster $C^a$ are $\mathcal{M}^a = \{a_1, a_2, a_3\}$ and the other agents, $\mathcal{M}^b = \{b_1, b_2\}$, form cluster $C^b$. The institution of $C^a$ is $I^a = \{queue, plurality\}$, representing one instance of $\mathcal{L}$, for example. The environment $\epsilon$ of this cluster would contain the pool $P_1$ as a brute fact $Bf$ and the agents $a_1$, $a_2$ and $a_3$ in the role of $member$ as institutional facts $If$.

### 3.7. Simulation Platform

In order to implement the open systems model from the previous section as a computer model, we can make use of a multi-agent based simulation platform. We expect the following functionalities from that simulation platform:

- Sophisticated functionalities for agent programming—Agents are not only considered as particles but as heterogeneous entities that have individual goals and strategies.

- Rule-engine integration—Facilitates the implementation of rules as an executable specification.

- Abstraction—Ability to leverage the infrastructure of a platform in order to deal with high level objects and concepts, rather than low level implementation issues.

- Parameterisable specification—For setting up the simulation specification to match parameters to an initial state.
3.8. Formalism: Drools

- Batch execution—Appropriate tools to automatically run repeats or parameter space sweeps, particularly useful when running experiments on a high performance computing cluster.

- Database connection—Methods for storing structured and unstructured data for analysis and post processing.

There are various platforms that can be used for agent simulation and most platforms are tailored to specific needs. The authors of [82] present a survey of a range of different platforms including their use case. For example, SimAgent is a simulation platform that specialises in multi-agent systems with complex environments and human-like intelligent agents. However, it requires the use of Poplog for implementation and there seem to be no interfaces to other programming languages available, which is why we did not choose to use this particular platform. Another alternative from that survey is Brahms which was built for organisational processes, but it is not open source and uses a custom-built language for implementation which limits functionality.

Popular, more generic platforms are NetLogo or MASON. In NetLogo, agents can act simultaneously and independently of each other, however an implementation into different class files is not supported and the implementation of complex models is problematic, as discussed in an evaluation of agent-based simulation platforms [67]. MASON has several advantages, such as the implementation of agents as class files, easier parameterisation and performance speed. Presage2 can be applied to a wide range of situations and complements other agent-based modelling tools by providing support for all the requirements outlined above. We do not have to make sacrifices on the agent architecture and can easily simulate a large number of repeats. Presage2 is designed to manipulate policies and has a ready usable database connection supporting several popular database management systems, we therefore choose this platform over MASON.

3.8. Formalism: Drools

In this section, we will present the formalism we use for expressing rules in the computer model of the sociologically-inspired computing methodology. There are various alternatives to represent and reason about events and their effects. In previous work [93], we used the Event Calculus [64] for the logical axomatisation and implemented the axioms of the specification in C++ as state-transition constraints, similar to [42]. This method is inelegant and error-prone, however, at the time there was no efficient dialect of the EC that would scale up for the kind of experiments used here. Recently, such a dialect has been developed by Artikis et al. [11]. Here, we will use a different formalism that

\[^5\text{The release of this dialect was too recent to be used in this work.}\]
can be directly implemented and allows us to process more events over a longer time frame during experimentation.

### 3.8.1. Rule-based systems

In this section, we give a Drools oriented overview of the tools we use. To build an open system based on rules, we use the concept of production systems [22], which are forward-chaining reasoning engines. They store so called production rules for knowledge representation in the production memory (PM), and keep facts and inferences they make about the rules in the working memory (WM) which is constantly changing during system operation. A typical production system is shown in Figure 3.6.

![Diagram of a production system](image.png)

**Figure 3.6.: Basic operation of a production system**

A production rule is usually written in the following form:

```
if <conditions> then <actions>
```

On the left-hand side of the rule (LHS) there are conditions which will be tested against the current state of the working memory, also called pattern matching. If they evaluate to `true`, a set of actions, also called consequences, on the right-hand side of the rule (RHS) will follow that modify the working memory. Subsequent evaluations of rules follow according to these steps, as illustrated in Figure 3.6:

1. Find matches, i.e. rules in the production memory whose conditions are satisfied with respect to the current working memory.
2. Resolve conflicts among the rules that are found and decide which rules to execute (i.e. to fire) in what order.
3. Change the working memory according to the actions of the selected rules.
These three steps are repeated until there are no more rules to fire.

The working memory holds a set of working memory elements (WME) that are tuples of the form [22, p. 119]:

\[(type \ attribute_1 : value_1 \ ... \ attribute_n : value_n)\].

Each WME is an existential sentence:

\[\exists x : [type (x) \land attribute_1 (x) = value_1 \land ... \land attribute_n (x) = value_n]\]

The predicates type, attribute_i and value_i are all atoms, and the order of attributes is not important. An example of a working memory element is:

\[(agent \ id:7 \ active:true)\].

Working memory elements are also referred to as facts or objects, and attributes as fields.

In order for a rule to fire, each of condition of the conjunction has to match a WME and each condition can either be

- an atom,
- a variable (such as id:i),
- an evaluable expression (such as id:(i + 2) where i is mentioned elsewhere),
- a test (such as id:(< 7 \land > 3)), or
- a conjunction (\land), disjunction (\lor) or negation (\neg) of a combination of the above

Note that, for a negation to evaluate to true, none of the working memory elements can match the condition of the negation at the time of assertion.

Once a rule has fired, all consequences have to be executed in order. They can be one of the following types:

- insert pattern,
- remove i, or
- modify i \{attribute_j = new_value\}.

The pattern is written in the form of a WME and is directly added into the working memory; i is the i-th condition of the rule’s left-hand side and is either removed completely or modified as specified.

These are just the basic functionalities of a production system, a general computational
3. Self-Organisation

framework. Advantages of this framework are the modularity, meaning that the rules work independently; control, meaning there are no complex control mechanisms hidden in the implementation; and transparency, meaning that the rules are easily understandable as the rules are described in a terminology close to natural language.

This framework is used to build expert systems which include ontological models for domain representation and facilities for knowledge acquisition and explanation. Well-known examples of expert systems include Mycin, developed for assisting medical diagnoses [108] and Xcon (formerly R1), the first rule-based system that allowed subsets of rules to work independently from each other, presented in [76].

3.8.2. Drools Expert

The rule-based programming environment we use here is called Drools Expert [94], a declarative, rule-based, coding environment implemented in Java. Drools uses a production rule system based on the Rete algorithm, an efficient method for matching a large collection of patterns to objects, the details are presented in [44].

In the following we will explain the syntax for creating rules in the Drools Expert system. Again, each rule consists of a when...then body, labelled with a starting (rule) and an ending keyword (end), and some optional rule attributes:

```
rule "rule name"
  <rule-attributes>
when
  <conditions>
then
  <consequences>
end
```

Rules of this form are stored in the production memory for pattern matching. A pattern in Drools looks as follows:

```
Type( field1 == value1, ... , fieldN == valueN )
```

An example of a member that had been sanctioned 9 time slices ago and is currently serving a sentence is:

```
Member( confined == 9, served == false, isFree == false)
```

The fact type or attributes (here fields) of a condition can be referred to by the use of pattern binding variables that are prefixed by $, though this is an optional character.

64
3.8. Formalism: Drools

These binding variables are then used to reference the matched objects in either another condition (LHS) or in the consequences on the RHS:

\[
\text{Type1}(\text{field1} == \text{value1}, \$\text{var1 : field1}) \\
\$\text{type : Type2}(\text{field1} == \$\text{var1})
\]

An example of using binding variables in a rule is:

```drools
rule "Sentence served"
when
\$m : Member\( (\text{confined} >= 10, \text{served} == \text{false})\)
then
    modify(\$m){
        \text{served} = \text{true}
    }
    insert( new Task\( (\text{freeMember} = \$m)\) )
end
```

This rule says that whenever there is a member in the working memory whose status is 'sentence not fully served', then modify that status of the field `served` to true when the time it had been confined reaches 10. We also insert a fact into the WM that can trigger a notification being issued to that member. A rule that is triggered by this fact could be:

```drools
rule "Notify end of sentence"
when Task(\$m : freeMember)
then
    sendNotification(\$m);
    modify(\$m){
        \text{isFree} = \text{true}
    }
end
```

This rule contains a function that sends a notification to the member (the binding variable \$m of the `sendNotification` attribute) and modifies the member’s status of being free to `true`. If there is no `Task()` element in the working memory, this rule will not fire.

In order to declare a new fact type, we can use the syntax on the left, an example using the served sentence scenario is given on the right:
To create a rule, there are several keywords that are recognised within a specific context, denoting conditional elements, operators, and rule attributes, and can be used in other places if wanted. A summary of soft keywords that are used in this work is given in Table 3.2. Drools’ reserved keywords, namely `true`, `false` and `null`, cannot be used as identifiers when writing rules.

<table>
<thead>
<tr>
<th>conditional element</th>
<th>operator</th>
<th>rule attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td><code>&lt; &lt;= &gt;=</code></td>
<td><code>no-loop</code></td>
</tr>
<tr>
<td>or</td>
<td><code>contains</code></td>
<td><code>ruleflow-group</code></td>
</tr>
<tr>
<td>not</td>
<td><code>not contains</code></td>
<td><code>salience</code></td>
</tr>
<tr>
<td>exists</td>
<td><code>memberOf</code></td>
<td></td>
</tr>
<tr>
<td>forall</td>
<td><code>not memberOf</code></td>
<td></td>
</tr>
<tr>
<td>from</td>
<td></td>
<td></td>
</tr>
<tr>
<td>collect</td>
<td></td>
<td></td>
</tr>
<tr>
<td>accumulate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The conditional elements `collect` and `accumulate` allow rules to reason over a range of objects. Typically objects get collected into a list or set, `accumulate` furthermore allows for custom actions on the collected objects in order to return a result object, such as the sum of all sentenced values of accumulated members.

The operator keywords `contains` and `memberOf` are used to check whether a field that is a collection or list contains a specific object. For `memberOf` the collection must be a (bound) variable.

The option to have rule attributes provides a declarative way to affect the rule selection and we will describe all three attributes in turn:

- **no-loop**: When a rule fires, it can modify a fact in the working memory that causes the rule to fire again. This attribute is used to skip the creation of a second activation (with the current set of facts), and so avoids infinite loops\(^6\).

\(^6\) Though this does not protect from the fact that two rules can alter the working memory in such a way that they are called alternately on repeat. To prevent such behaviour, facts such as
3.9. Formal axiomatisation of six principles

- **ruleflow-group**: This attribute controls the rule flow, which is a feature to regulate the firing of rules. The rules are labelled by a ruleflow-group identifier and will only fire when their group is active.

- **salience**: This attribute describes the priority of a rule (within ruleflow groups if they are used) and can have negative or positive values, the default is 0. When choosing what rule to execute, the rule with higher salience value is given priority.

The right-hand side of the rule states the consequences or actions that follow in that order when the rule is fired. Typically, the RHS is a short list of actions.

As mentioned above, the keywords that evoke some action on the RHS are:

- **insert(new F)**: This statement inserts a new fact into the working memory, the attributes have to be provided depending on the fact type.

- **remove($f)**: The chosen fact will be deleted from the working memory.

- **modify($f)**: This statement modifies the attributes of the fact as stated.

To simulate resource allocation in open systems, Drools Expert is integrated into the Java simulation platform Presage2 [73]. In every time slice (or round) of operation, the rules will be matched against the WM and executed according to a set ruleflow, as we will explain later.

### 3.9. Formal axiomatisation of six principles

We will now give a candidate axiomatisation of the first six principles for governing common-pool resources, see Table 2.1 on page 38. This serves as a proof of concept for answering \([QI]\) and brings together several strands of research in access control, voting, and alternative dispute resolution. For each method there are various alternative formalisations in various complexities possible. Choosing the complexity becomes important when the cost of adaptation, i.e. shifting to another degree of freedom, has to be considered.

The fact types (classes) of **member**, **non-member** and **institution** are declared here. This provides an overview over the fields that are used for pattern matching of rules to the working memory in this section\(^7\). The institutional roles **member**, **head**, **monitor** and **gatekeeper** all have the same basic constructor, with the **name** being the unique identifier of an agent taking on different roles. The discriminating factors of these roles are the

\[^7\] The declarations are merely indicative and there are more fields defined for each fact type, for example what set of principles is active in the institution, the number of sanctions of a member, etc.
actions they are empowered, permitted and obligated to perform, which is here not part of the role description but typically constrained by the conditions of the rules. A complete class diagram will be given in the implementation section (Section 4.2.2).

Note that the pool is represented differently in the three declarations. Whereas the institution’s pool attribute is the actual CommonPool fact, member and non-member are related to a pool through its unique identifier, the id field of CommonPool, which is of integer type.

Although open systems operate in a distributed manner, we will account for all the actions that take place in a centralised manner. It is therefore important to document what action has been executed by what agent in what time slice.

3.9.1. **P1**: Clearly defined boundaries

This principle defines the boundaries of the system, what agents are authorised by the institution to withdraw resources (member) and closes access to outsiders (non-member). These three aspects are represented using different rules and methods. Suitable acMethods use role-based access control as discussed in [101] and define a role-assignment protocol, following [13] for example.

```plaintext
rule "Grant access to non-member"
when
   Institution($iid:id, $r:round, pr1==true)
   $gk: Gatekeeper(instId==$iid)
   $nm: NonMember($n:name)
   $a: App(name==$n, instId==$iid, round <$r, state=="open")
   AccCond(checkBy==$gk, applicant==$n, evaluateTo=true)
then
   retract($nm);
   insert( new Member($nm, $iid) );
   modify($a){
       state="closed"
```
This rule says that, whenever there is an open application for membership (App) to some institution (instId) in some round earlier than the current one ($r), and furthermore the gatekeeper evaluates that the access conditions (AccCond) are met, then the non-member will become a member, i.e. the $nm fact is retracted and a new member fact naming the institution inserted. Also, the status of the application is set to closed. There are some additional constraints such as that the gatekeeper has to be a member of the same institution and $P1$ is being used.

The access conditions that have to be fulfilled so that a non-member is granted access depend on acMethod which is stated in further rules. If the access control method is discretionary, then the gatekeeper can tie the decision to conditions such as the number of active members in the institution.

Principle $P1$ also makes sure that no unauthorised members appropriate the resource, and a non-member’s status can be set to inactive whenever they were caught unrightfully appropriating. An example of an exclusion rule is:

```
rule "Sanction monitored non-members"
when
    Institution($iid:id, $p:pool, $r:round, pr1==true)
    $hd: Head(instId==$iid)
    $mon: Monitor(instId==$iid)
    $nm: NonMember($n:name, pool==$p.id)
    Monitored(agent==$n, round==$r, monitor==$mon)
    not Sanctioned(agent==$n, round==$r, inst==$iid)
    Appropriated(agent==$n, round==$r, pool==$p, quantity > 0)
then
    insert ( new Sanctioned($name, $round, 1, $iid, $hd) );
    modify ($nm){
        setActive(false)
    }
end
```

The conditions of this rule say that a non-member has to be monitored by the monitor performing an appropriation (quantity > 0) from the pooled resource that the institution is accountable for, but sanctioning has not yet occurred. Only then, a Sanction fact is inserted by the head into the WM and the activity of the non-member set to false.

Similar role-assignment axioms are needed for the head to reassign roles to agents that are already members of the institution, or to exclude members from the institution depending on the sanctioning mechanism.
3. Self-Organisation

3.9.2. **P2**: Congruence between appropriation and provision rules and local conditions

This principle is concerned with the appropriation rules themselves. These rules restrict, for example, time and quantity of access and make sure they are in line with the current state of the environment. We present an example of an allocation procedure using *queue* as *raMethod*. The first rule below ensures that *members* of that institution place at most one demand per round, and only if they are not still in the demand queue (*demQ*) of agents that had not been allocated in the previous round. The issue of ‘valid’ and ‘invalid’ requests (or demands) is discussed in [101].

```plaintext
rule "Member demands"
when
  $i: Institution($iid:id, $p:pool, $r:round, pr2==true)
  $m: Member($n:name, instId == $iid)
  not Demand(agent==$n, round <$r, this memberOf $i.demQ)
then
  insert(new Demand($n, $r, $m.demand($i, $p), $p.getId()));
end
```

Note that the *Demand* fact has a function as third attribute, *demand($i, $p)*. This function is *member* internal and not directly governed by the rules, though a considerate agent will take the local environment and rules into account when placing a demand for resources.

The second rule of *raMethod* regulates how the resource allocation is to be performed.

```plaintext
rule "Perform resource allocation"
when
  $i: Institution($iid:id, $p:pool, $r:round, pr2==true)
  $hd: Head(instId==$iid)
  $dL: List() from collect( Demand(pool==$p.id, round==$r))
then
  Set<Allocation> allocations = $hd.allocate($i, $p, $dL);
  for(Allocation a : allocations) {
    insert(a);
  }
end
```

The core allocation function, *allocate($i, $p, $dL)*, is executed by the *head* as an agent-internal function (implemented in Java), using a list of demands placed in the current round ($dL) and the demand queue (field $i) from the previous round. There

---

8 Here the queue is represented as ordered list of demands.
are rules about how to place agents into the queue and how to allocate the demands with respect to the pooled resource $p$, which a compliant agent will follow.

There are several rules and methods to regulate the process of allocating resources, in Algorithm 1 we present $raMethod=\text{queue}$.

### Algorithm 1: allocate() for $raMethod=\text{queue}$

```plaintext
level ← pool resource level;
demQ ← demand queue from previous round + new demands;

while (demQ is not empty) ∧ (level $\geq$ demand $d$ from demQ.front) do
  allocate $d$ to corresponding agent;
  deduct $d$ from level;
  delete demQ.front;
```

The agents’ demands are added to the remaining demand queue from the previous round, then the demands in the queue are allocated until there is no resource left in the pool. The allocation does not alter the brute facts of the resource, this is done in the appropriation phase afterwards.

#### 3.9.3. $P3$: Collective-choice arrangements

This principle establishes that individuals that are affected by the operational rules also have the right to participate in their modification, e.g. through votes. Several factors have to be taken into account when setting up voting procedures, as discussed in [91]. Firstly, we need to define the rightful agents that are legitimate to vote (empowerment); secondly, a mechanism to count the votes has to be specified; and thirdly, we need an agent occupying a designated role that initiates the voting procedure and one that is obligated to declare the correct outcome of the vote.

Initiation of the voting procedure for new head and/or new $raMethod$:

```plaintext
rule "Call for votes"
when
  $i$: Institution($iid:id$, pr3==true)
  $hd$: Head(instId==$iid$)
then
  CallForVote cfv = $hd$.callForVotes($i$);
  modify($i$) {
    setVoteHead( cfv.isHead() ),
    setVoteRaMethod( cfv.isRaMethod() )
  }
```
A *head* can initiate two voting procedures (*voteHead* or *voteRaMethod*) with this call for votes. Note that the *callForVotes($i$)* function can only be called by the head.

Here we present a collective choice on an operational rule, i.e. a voting procedure on *raMethod* where the *members* can vote on the new method to be either *ration* or *queue*:

```plaintext
rule "Vote for raMethod"
when
  $i$: Institution($iid:id, $p:pool, voteRaMethod==true, $r:round, pr3==true)
  $m$: Member($n:name, instId==$iid, pool==$p.id)
not Vote(voter==$n, round==$r, ballot="raMethod")
then
  Vote v = $m.vote($i, $p, "raMethod");
  v.setVoter($m.getName());
  v.setRound($r);
  insert(v);
end
```

For each agent that is a *member* of the institution, the *vote()* function returns a vote which depends on the agent’s (internal) preferences. Subsequently, the votes (pairs of *(ballot, value)*) are counted:

```plaintext
rule "Count votes"
when
  Institution($iid:id, $r:round, pr3==true)
  $hd$: Head(instId==$iid)
  $ag$: Member(name==$n, instId==$iid)
  Vote(voter==$ag, round==$r, ballot=="raMethod")
not VoteCount(institution==$iid, ballot==$b, round==$r)
$vL$: List(size > 0) from accumulate(
  $v$: Vote(voter, round==$r, ballot==$b) and
  Member(name==$voter, instId==$iid), collectList($v))
then
  HashMap<Integer,Integer> tally = new HashMap<Integer, Integer>();
  for(Object o : $vL) {
    Vote v = (Vote) o;
    if(tally.containsKey(v.getValue())) {
      tally.put(v.getValue(), tally.get(v.getValue()) + 1);
    } else {
      tally.put(v.getValue(), 1);
    }
  }
```

72
If there is at least one agent that voted in some ballot, then the rule accumulates all votes of that ballot from the WM onto a list ($vL$). Each value that occurs in the votes becomes a key in the tally map, and the number of occurrences of that value is mapped to the key. Note the not VoteCount() condition that ensures that the rule is executed once for the same set of institution, ballot and round. The head then inserts the result of the voting procedure (tally) into the WM as a field in VoteCount. This fact is used in the next rule where the same agent declares the winner, i.e. the new raMethod, and updates the operational rule in the institution:

```plaintext
rule "Declare winner and update raMethod"
when
$I$: Institution($iid:id, $r:round, pr3==true, wdm="plur")
$hd$: Head(instId==$iid)
$vc$: VoteCount(ballot=="raMethod", round==$r, inst==$iid, 
    head==$hd)
not Declared(inst==$iid, ballot==$vc.ballot, round==$r)
then
    Integer forQueue = $vc.result.get(RaMethod.Q.ordinal());
    Integer forRation = $vc.result.get(RaMethod.R.ordinal());
    if (forQueue==null) forQueue = 0;
    if (forRation==null) forRation = 0;
    if (forQueue > forRation) {
        modify($i){
            setAllocationMethod( RaMethod.Q );
        }
    } else if (forRation > forQueue) {
        modify($i){
            setAllocationMethod( RaMethod.R );
        }
    }

    insert( new Declared($iid, $hd, $vc.ballot, $vc.round,
        $i.getAllocationMethod().ordinal() ) );
end
```

The winner determination method used here is plurality (wdm="plur") and in case of a tie there are no changes to raMethod. There are many alternatives to the wdMethod used here, and many alternatives to the procedure of voting. In [91], a full formalisation of a voting protocol based Robert’s Rules of Order [98] is presented, including specific issues such as enforcing one-member-one-vote or private ballots.
3. Self-Organisation

3.9.4. **P4: Monitoring**

This principle is about monitoring the conditions of the resource and appropriator behaviour, and the cost involved in that process. Logic-based representations are very well suited for event recognition and environment monitoring, refer to [9] for a review of suitable approaches.

Here, an agent in the role of *monitor* is appointed to report any offences regarding appropriations exceeding allocations. First, the *monitor* has to decide what *members* to watch during the appropriation action:

```
rule "Member monitoring list"
when
  $i: Institution($iid:id, $p:pool, $r:round, pr4==true)
  $mon: Monitor(instId==$iid)
  $mS: Set() from collect( Member(instId==$iid) )
  not TaskExecuted(inst==$i, task="monitor", round==$r)
then
  Set<String> monitored = $mon.monitor($i, $p, $mS);
  for( String ag : monitored ) {
    insert( new Monitored(ag, $round, $mon) );
  }
  insert( new TaskExecuted($i, "monitor", $round) );
end
```

The set `monitored` is a collection of agents names, specifying what agents are monitored.

The following rule deducts a cost (`monitorCost`) from the resource that has to be paid for each monitoring action:

```
rule "Deduct member monitoring cost"
when
  $i: Institution($iid:id, $p:pool, $r:round)
  $mon: Monitor(instId==$iid)
  Member($n:name, instId==$iid, pool==$p.id)
  $mo: Monitored(agent==$n, round==$r, monitor==$mon)
  not Deducted(action==$mo)
then
  modify($p){
    setResourceLevel(getResourceLevel()-$i.getMonitorCost())
  }
  insert( new Deducted($mo) );
end
```

As a result of the monitoring process, a *member* can be sanctioned by the *head* according
3.9. Formal axiomatisation of six principles

to the next rule. For this case we assume that there is no graduation in the sanctioning method, which means that a sanction at level 1 leads to an immediate exclusion (here merged in one rule).

```plaintext
rule "Sanction members"
when
  Institution($iid:id, $p:pool, $r:round, pr5==false)
  $hd: Head(instId==$iid)
  $mon: Monitor(instId==$iid)
  $m: Member($n:name, instId==$iid, pool==p.id)
  Monitored(agent==$n, round==$r, monitor==$mon)
  ( Allocation(agent==$n, round==$r, $alloc:quantity, pool==$p)
    and
    Appropriated(agent==$n, round==$r, $approp:quantity, pool==$p, $approp > $alloc)
  )
or
  ( not Allocation(agent==$n, round==$r, pool==$p)
    and
    Appropriated(agent==$n, round==$r, $approp:quantity, pool==$p, $approp > 0)
  )
then
  insert( new Sanctioned($n, $r, 1, $iid, $hd) );
  retract($m);
  insert( new NonMember($m) );
end
```

The conditions on the LHS say that there are two cases in which a member can be sanctioned/excluded. Firstly, if the quantity an agent appropriated exceeded its allocation ($approp > alloc) and secondly, if there is no Allocation() fact for that agent in the WM but it performed an appropriation ($approp > 0). These two cases could be split at the conditional argument or and put into two separate rules, with the rest of the conditions and consequences staying the same.

Note how the last two rules did not state pr4==true, this is implied as no monitoring action could have occurred without that principle in the first place.

3.9.5. P5: Graduated sanctions

This principle is concerned with a flexible scale of sanctions, which is applied to agents that violate a rule of the institution. The following rule shows how agents are sanctioned
3. Self-Organisation

for repeated violations:

```plaintext
rule "Sanction members (repeated offences)"
when
  Institution($iid:id, $p:pool, $r:round, pr5=true)
  $hd: Head(instId==$iid)
  $mon: Monitor(instId==$iid)
  $m: Member($n:name, instId==$iid, pool==$p.id)
  Monitored(agent==$n, round==$r, monitor==$mon)
  ( Allocation(agent==$n, round==$r, $alloc:quantity, pool==$p)
    and
    Appropriated(agent==$n, round==$r, $approp:quantity, pool==$p, $approp > $alloc)
  )
  or
  ( not Allocation(agent==$n, round==$r, pool==$p)
    and
    Appropriated(agent==$n, round==$r, $approp:quantity, pool==$p, $approp > 0)
  )
  not Sanctioned(agent==$n, round==$r)
  Sanctioned(agent==$n, $level:level)
  not Sanctioned(agent==$n, $l:level, $l > $level)
then
  insert( new Sanctioned($n, $r, $level+1, $iid, $hd) );
  modify ($m){
    setActive(false)
  }
end
```

For this rule, there are the following conditions necessary. Firstly, the monitor has monitored a member and detected an offence (and/or-construct) by this agent. Furthermore, the agent has not been sanctioned in this round and $level$ is its number of previous sanctions, i.e. there is no sanction fact with a higher level in the WM.

There should also be a rule, similar to “Sanctioning monitored members”, that sanctions an agent for a first offence, i.e. inserts a Sanction() (at level=1) fact and sets the agent’s activity to false, rather than excluding the agent from the institution. This rule is required so that the conditions of the above rule (more precisely Sanctioned(agent==$n, $level:level), the so far highest level) can be met.

---

9 We assume here that an agent is sanctioned for every offence, but a policy where an agent is only sanctioned once every three offences, for example, is also possible.
In the consequences of the rule, the head inserts a sanctioning fact whose level is increased by 1 to the previous sanction, and modifies the member’s activity to false.

A member can regain access to the resource\textsuperscript{10}, i.e. its activity state is reset to true, depending on the terms of the sanction.

```plaintext
rule "Member back after sanction"
when
  $i$: Institution($iid:id, $r:round, pr5==true)
  $gk$: Gatekeeper(instId==$iid)
  $m$: Member($n:name, instId==$iid, active==false)
  $s$: Sanctioned(agent==$n, $sanRd:round, $level:level, ($sanRd + $level*$i.excludetime < $round), $level <= $i.maxSanctionLevel)
  not Sanctioned(agent == $n, $rd:round, $rd > $sanRd)
then
  insert( new Included($n, $r, $iid, $gk) );
  modify ($m){
    setActive(true)
  }
end
```

Since the time of the most recent sanction ($\$sanRd$), an agent must be inactive for a number of rounds which is given by the level of sanction times a constant `excludetime` (specified by the institution). Furthermore, the level of sanction must not be higher than the allowed maximum (maxSanctionLevel), then the gatekeeper modifies the agent’s activity state to true and inserts an Included fact.

This rule is part of acMethod and the readmission of a member can require this agent to go through an application procedure beforehand. For cases where maxSanctionLevel is exceeded, another rule describing the exclusion procedure should follow, according to the rules of exMethod.

### 3.9.6. P6: Conflict-resolution mechanisms

This principle states that the institution should include mechanisms to resolve conflicts between agents rapidly and at a low cost. Different types of conflict can occur when self-organising a resource-allocation process. One type is concerned with the conflict between rules, meaning that some action can be permitted and prohibited at the same time for a particular agent. In [117], mechanisms for the detection and resolution of

\textsuperscript{10}According to the rules of this institution, an agent remains a member during the graduated sanctioning process (until exclusion), though its status is set to inactive for that time frame.
such normative conflicts are presented. Another type of conflict occurs when agents do not act according to the rules, are not aware of some rule or make mistakes applying the rules, which leads to intentional and unintentional violations.

Alternative dispute-resolution (ADR) mechanisms are well suited for resolving conflicts in institutions as an alternative to litigation, as they can preserve and even strengthen the relationship among the involved parties. In contrast to a litigation process, ADR does not require filing lawsuits and going to trial (even though only a small proportion arrive at a verdict as often settlements are reached before or cases break down), thus the resolution of conflict comes at a lower cost, in shorter time and with limited damage. ADR mechanisms include mediation and arbitration, which allow the disputed parties to solve their conflict with greater control and in a more creative way than if left to a judge or jury.

The method presented here is just a simple appeals procedure, though more refined mechanisms can be defined. The evolution of online dispute resolution and cultural implications are presented in [62].

In the following rule, a member appeals against a sanction:

```plaintext
rule "Appeal against sanction"
when
  $i: Institution($iid:id, $r:round, pr6==true)
  $m: Member($n:name, instId==$iid, active==false)
  $san: Sanctioned(agent==$n, round==$r)
  not Appealed(agent==$n, round==$r)
then
  insert( new Appealed($n, $r, $iid) );
end
```

In this rule, an agent appeals after every sanction, but a discretionary method is also possible. An appeal can only be made once per time slice and the sanction is upheld by the head with:

```plaintext
rule "Uphold sanction"
when
  $i: Institution($iid:id, $r:round, pr6==true)
  $hd: Head(instId==$iid)
  $m: Member($n:name, instId==$iid, active==false)
  $san: Sanctioned(agent==$n, round==$r)
  not (Appealed(agent==$n, $aRd:round, inst==$iid, $r - $aRd <= $i.appealtime) )
  not (Included(agent==$n, $iRd:round, inst==$iid, $r - $iRd <= $i.appealtime) )
```
3.10. Summary

In this chapter, we brought together the necessary tools to be able to perform the first step of the SIC methodology, the formal characterisation. We described a framework that allows us to address the three levels of analysis and rules using a multi-level protocol stack, which maintains the functionality of a (human) institution to change rules at runtime. Using an appropriate formalism, we then gave an axiomatisation of Ostrom’s first six design principles that make use of the roles, rules and methods defined before.

The rules presented here are only a subset of rules necessary for implementing the required methods, further rules will be presented when we describe the algorithm of the testbed. Nevertheless, concluding from Section 3.4 and 3.5, we showed that the process of formal characterisation from the SIC methodology allows us to axiomatise the rules of the institution. This axiomatisation represents a dynamic system specification \( \mathcal{L} \) which is used to perform the resource-allocation process according to the Ostrom’s design principles. Therefore, we can positively answer \( Q_2 \), it is indeed possible to use the dynamic specification for norm-governed systems to model socio-economic principles for enduring CPR management.

Now the question remains:

\[ Q_3 \]

Is it possible to use the formal axiomatisation to specify and implement a testbed that ascertains the sufficiency of these principles for enduring open systems?

```plaintext
then
    insert(new Upheld($n, $r, $iid, $hd));
retract($san);
end
```

The sanction is upheld by the head if the time of last appeal and last inclusion dates back long enough. By that we mean that the agent must have been active in the institution for at least \texttt{appealtime} rounds without being accused of wrongful appropriations.

The \texttt{Upheld()} fact then allows the gatekeeper to grant the member access to the resource again, i.e. \texttt{$m.setActive(true)}\), using a rule similar to “Member back after sanction”.
4. Self-Organising Resource Allocation

4.1. Introduction

The formal characterisation of Ostrom’s first six principles using an appropriate formalism was the first step towards simulating an open system. Following further the methodology for sociologically-inspired computing, see Figure 3.1, the next two steps are the principled operationalisation and controlled experimentation.

We first show how to embed the formalisation of agents and rules into the Presage2 simulation platform and define the classes and agent states in Java, and the control loop using the rule-flow group functionality provided by Drools. We then describe the agent behaviour in detail, the implementation of specific rules and class functions can be found in the appendix.

The next step is the setup of the testbed for experimentation. We describe the sets of parameters that are used to initialise the sets of experiments, and what aspect of self-organisation a specific set of parameters is testing. Possible parameters are the set of principles in use, specific agent characteristics or environmental factors that influence the pooled resource. We then evaluate the data obtained from experimentation and show that a full set of principles is beneficial for the endurance of the system operation.

For the first set of experiments, we used the same parameters as in our publication [93], but using the new testbed. Three more sets of experiments follow to examine the effect of different environmental conditions that were previously unexplored.

4.2. Principled operationalisation

In this step, we have to define the implementation of the environment and the individual agent behaviour. According to [41], the used formalism has to reflect the key components of a specification for ‘electronic institutions’, i.e. it should be able to express the rules that constitute the institution, the roles and the environment in which the agents can communicate with each other, see also [10].
4. Self-Organising Resource Allocation

4.2.1. Testbed specification in Presage2

To facilitate the integration of the key elements relevant for this simulation, we make use of Presage2, a multi-agent based animation and simulation platform, that we present in more detail in [73].

The core of the Presage2 platform controls the interaction of three main parts: the simulator, the libraries and the experimentation tools. In the simulator, there is the state engine and the rule engine, wrapped together by the core which also initialises the simulation loop. The state engine is responsible for storing and updating the simulation state, and has an interface to the rule engine’s working memory. There are several packages that can be used to customise a simulation, such as libraries for the environment, agents or a (communication) network, and experimental tools including databases and a batch executor.

Figure 4.1 shows the process of creating a simulation with Presage2. In the phase of principled operationalisation, we embed the rules, the facts and the agents into the simulation specification.

For the simulation in this work, we set up Drools as our rule engine. We create a package containing the roles of agents including the ‘internal’ procedures of agents in...
that role (which can be violated), and we further implement a package containing the facts that are used by the working memory. These facts include brute facts, such as the replenishment mechanism of the resource; institutional facts, such as the raMethod and facts about actions that are performed by the agents, such as allocations, demands, etc.

Each round of the simulation is regulated by Drools’ ruleflow groups. There are nine groups whose rules are fired when the group is activated, independently of the rules in other groups. The flow of groups is shown in Figure 4.2. Each round of an experimental run starts with an Init phase where the round is incremented. If the resource has not been depleted in some previous round (which leads to the abortion of the run), the resource is refilled and the next ruleflow group activated. The remaining ruleflow groups will be discussed in detail later.

### 4.2.2. Classes

An agent is permitted, prohibited or obligated to perform certain actions depending on the role it occupies in the institution. Figure 4.3 is a class diagram showing the different roles with respect to the institution, including a selection of class fields and methods. The full list of class fields can be found in Appendix B. Each class object that is created during simulation, is inserted into the working memory, along with the fields as

\[
\text{ClassName}( \text{field1} = \text{value1}, \ldots, \text{fieldN} = \text{valueN} )
\]

The roles member and non-member are mutually exclusive, and they get determined at the start of a simulation run via acMethod. The institution defines that an agent has to be a member\(^1\) in order to take on the role of a head, gatekeeper or monitor, and an agent can be assigned more than one role.

Each agent can be uniquely identified by its name, further fields that all agents have in common are activity, stating whether an agent has the capability (or not) to take

\(^1\)We assume that an agent can be a member of only one institution at a time.
resources\(^2\) and its degree of compliancy, a private field stating how prone the agent is to unrightfully appropriate the resource (compliancy_degree). In addition, a member contains fields stating its institution (inst_id) and the associated common-pool resource (pool_id), and fields keeping track of the offences and level of sanctioning\(^3\) (sanction_level).

All agents are able to perform appropriate actions, we recall that that only a member is permitted to appropriate the resource, but both active member and non-member have the capabilities to do so (the non-member stores the information what pool to appropriate from in pool_id). In addition, all agents are able to revise their behaviour (rev_behaviour) according to their own judgement and to what they perceive in the environment, such as sanctioning events.

In the role of member an agent is furthermore empowered to vote, place a request for resources (demand) and appeal against a sanctioning procedure. The actions that an agent decides to take depend on its internal state and how it perceives the environment.

\(^2\)For this implementation, active=false means that it is impossible for an agent to appropriate the resource, i.e. the brute facts cannot be changed in that state.

\(^3\)As already said, here, each offence increases the level of sanctioning by 1 straight away.
4.2. Principled operationalisation

The actions that an agent in the role of gatekeeper can perform are the assignment of the member role (assign) and granting a member access again to the resource after a served sentence or successful appeals procedure (include). The monitor is empowered to carry out the monitoring of members (monitoring) and non-members (monitoring_out), and is obligated to report any offences that then initiate a sanctioning process, performed by the head (sanction). Additional actions the head is empowered or obliged to perform are calling for votes (cfv), declaring the outcome of a voting procedure (declare), the exclusion of an agent (exclude) and uphold sanctions after a member appealed.

The fields of an institution represent a collection of institutional facts (If), namely the used resource allocation method (raMethod), the access control method (acMethod), the winner determination method (wdMethod) and the method for alternative dispute resolution (adrMethod). Whether and in what way these methods are represented in the rules, depends on the set of principles that the agents use to govern their institution\textsuperscript{4}. Furthermore, there is the frequency with what monitoring is carried out during the appropriation phase (monitoring_freq) and when guarding the boundaries of the institution (monitoring_freq_out). The penalty that is applied for a sanction uses exclude_time\textsuperscript{5}.

The maximum level of sanctions (max_sanction_level) and the time an agent has to go unsanctioned so that an appeal is granted (appeal_time) are also part of the institutional facts. Being a construct of rules, the institution does not have any methods attached.

Finally, there is the common pool. The only method in this class is refill which happens at the start of every round. The field of the pool are brute facts (Bf) in the environment and include the currently available resource (resource_level), the parameters for the refill (refill_scheme), the maximum level to which the resource can be pooled (max_level) and a factor that describes whether unintentional violations happen during the appropriation process (unintent_violation).

In this testbed, the powers, permissions and obligations of a certain role are not implemented as part of the rule engine, but are expressed through the Java class methods and checked against the working memory. For example, if an agent wants to perform an allocation action which is part of the head class, it can only do so if there is a fact in the WM stating this agent as the head of the corresponding cluster. Typically, the rules in the production memory include these checks as conditions in the left-hand side of a single rule.

\textsuperscript{4}The presence of principles is not decided by the agents themselves but is an experimental parameter.
\textsuperscript{5}The penalty applied in Section 3.9, was to sanction a member for \texttt{sanction\_level*exclude\_time} rounds.
4. Self-Organising Resource Allocation

4.2.3. States

The agents in the simulation are heterogeneous, meaning that they have different characteristics that influence their behaviour when reacting to changes in the environment $\epsilon$ and specification instance $I$. Depending on the principles that are used in the run and the agents’ behaviour during resource appropriation, the activity status of an agent can be changed, see Figure 4.4.

There are four possible states that an agent can be in, they are active member, inactive member, active non-member and inactive non-member. At the start of a run, acMethod decides what agent is assigned the role member of an institution or non-member, both in active mode ($P1$).

A member remains active, if it does not appropriate more than its allocation or if the wrongful behaviour has not been detected by the monitor of the institution, thus not reported. Any detected noncompliance is reported ($P4$) and the agent’s status is set to inactive. Only in the following two cases, an agent’s state can be set to active again (i.e. included again in the allocation process): The level of sanction is below or equal to the maximum level allowed ($P5$) and the agent served its sentence, or the sanction has been upheld due to a successful appeals procedure ($P6$). In all other cases, the agent is excluded from the institution and becomes an active non-member. This agent is not permitted to become a member again in any subsequent round. Furthermore, a non-member is not permitted to appropriate the resource, but should it be reported doing so, the agent’s state is set to inactive which means it becomes impossible for the agent to appropriate the resource ($P1$).

Figure 4.4.: Agent activity state chart
4.2.4. Testbed control loop

In this section, we present the control loop used for self-organising a resource-allocation process in an open system. We first describe what facts are inserted into the working memory, how the rules are organised inside in the production memory, and then address the main methods that define the agents’ internal decision-making process (see Figure 4.1).

Initialisation and termination criteria

This is the initialisation\textsuperscript{6} for one cluster $C$ of a multi-agent system $S$ governing one common-pool resource (for a reminder of variables see Section 3.6).

\begin{itemize}
  \item $\exists p \in \epsilon \subseteq \mathcal{E}$:
    \begin{itemize}
      \item \texttt{insert( new CommonPool(p.id, resourceLevel==maxLevel) );}
    \end{itemize}
  \item $\exists i \in \mathcal{L}$:
    \begin{itemize}
      \item \texttt{insert( new Institution(i.id, round==0, pool==p, pr1==true) );}
    \end{itemize}
  \item $\forall m \in \mathcal{M} \subseteq \mathcal{A}$:
    \begin{itemize}
      \item \texttt{insert( new Member(m.name, instId==i.id, pool==p.id, active==true) );}
    \end{itemize}
  \item $\forall nm \in \mathcal{A} \setminus \mathcal{M}$:
    \begin{itemize}
      \item \texttt{insert( new NonMember(nm.name, pool==p.id, active==true) );}
    \end{itemize}
  \item $\exists h \in \mathcal{M}$ (repeat for Gatekeeper and Monitor):
    \begin{itemize}
      \item \texttt{retract( Member(name==h.name) );}
      \item \texttt{insert( new Head(name==h.name, instId==i.id, pool==p.id, active==true) );}
    \end{itemize}
\end{itemize}

The conditions that terminate an experimental run are the absence of a pool or institution, or that the time frame set for the run is over:

\begin{itemize}
  \item \texttt{CommonPool( resourceLevel < 0 )} retracts the CommonPool
  \item \texttt{not( exists(Member(instId==i.id)) )} retracts the Institution
  \item Institution(round \geq finishTime)
\end{itemize}

Ruleflow and control loop

In this section we present a ruleflow diagram that shows the ruleflow of the implementation, a control loop in algorithmic form is shown in Appendix A.17.

\textsuperscript{6}Note that the declaration of the facts’ field variables is merely representative.
Figure 4.5 shows how the rules are arranged within the ruleflow groups, most rules have been discussed in a basic version in Section 3.9. The dotted lines within a ruleflow group partition the rules according to their salience, with the rules on the left having priority over the rules on the right of the line. Furthermore, rules that fire only if a certain principle is active, are tagged with that principle on the bottom right. Arrows between rules mean that a rule that is a head can only fire, if a corresponding fact has been created by the tail rule. The very last rule to fire in each round is "create log files" which is for data collection purposes.

In Section 4.2.2 we described how the actions an agent can perform are connected to the role this agent occupies. Now, we illustrate how changes in the environment $\epsilon$ (both brute ($Bf$) and institutional ($If$) facts) and the use of design principles for CPR government invoke a sequence of agent actions and rule consequences.

Initially, we set $P1$ of the institution to active ($pr1=true$), refill the pool to its maximum level and assign the roles of member, non-member, head, monitor and gatekeeper. The algorithm starts in round $r = 0$ and cycles until one of three termination criteria is true. The corresponding code can be found in Appendix A.16 to which the following lines refer. The three criteria are: the predefined finishing time is reached, i.e. $r \geq finishTime$ (line 2); the resource is depleted, i.e. resourceLevel $< 0$ (lines 13, 17–21); or there are no members left in the cluster, i.e. $\mathcal{M} = \{\}$ (lines 28, 30). This is considering a single pool and institution per run, otherwise the run continues until all common-pool resources are depleted or clusters memberless.

In each round, the agents perform the following actions, depending on the selection of corresponding principles (see page 38) and ruleflow groups (Figure 4.2):

- **cfv**: With $P3$ in use, the collective-choice arrangements, the head calls for a vote on the resource-allocation method.
- **vote**: After a successful call for vote, all member agents vote for their preferred method and the head declares the winner to be the new raMethod. If $P3$ is not used, the method for the current cycle is automatically reset at periodic intervals according $\epsilon$.
- **demand**: In order to conduct a resource-allocation process, $P2$ has to be selected, so that active members are empowered to place demands.
- **allocate**: The head will then allocate the demands according to raMethod, such that the sum of allocations does not exceed the current resource level. Furthermore, the monitor creates lists of agents that it will monitor on appropriation, depending on the use of $P1$ and $P4$.
- **appropriate**: Subsequently, all agents perform an appropriate action, where even
4.2. Principled operationalisation

Figure 4.5.: Rule flow in the resource allocation process
4. Self-Organising Resource Allocation

the non-members have the power, though not permission, to appropriate from the common pool.

- **REPORT**: With $P_4$ in use, the monitor monitors the members on its list and reports the offences to the head. The same happens with the non-members on the list, if $P_1$ is in use. The head then applies a sanction to the misbehaving agents and the sanction level increases by 1 with each offence. Sanctioned non-members get their activity status set to false, which inhibits further appropriations. The remaining non-members revise their appropriation behaviour (i.e. ask the question whether it pays off to illicitly appropriate from this particular common pool in the future) which might change their degree of compliancy. In case $P_5$ is selected, the sanctioning method within the institution is graduated and an agent’s activity state is set to false after an offence. This state temporarily inhibits the member from appropriating resources.

- **APPEAL**: In case $P_6$ is selected, the sanctioned (inactive) member is empowered to appeal against the sanction and the head can uphold the sanction or reject the appeal, resulting in a decrease of sanctioning level, or not.

- **EXCLUDE**: The last ruleflow group is concerned with the reported members. If graduated sanctions are not used (no $P_5$), the member's sanction after being reported is the exclusion from the institution (remember $P_1$ controls who is and is not a member of the institution). A member is also excluded if $P_5$ is active and the member’s level of sanctioning has exceeded the limit.

If $P_5$ is used and the sanctioning level has not reached the limit, a member is empowered to apply for readmission after it has served its sentence. The duration of the sentence is determined by the sanctioning level and some fixed constant (excludetime), during which the agent will revise its attitude towards wrongful appropriation. After a successful application, the member is included back into the resource-allocation process, i.e. the gatekeeper sets the agent’s activity state to true. When a member becomes active again, it will revise its appropriation behaviour. When an agent successfully passes an appeals procedure ($P_6$), its activity state is changed back to active straight away.

- **INIT**: After all ruleflow groups have been activated to fire their rules, this process repeats. In this first ruleflow group, the round is incremented and the state of the system reviewed. If the resource level is $< 0$, there are no members left or the round reaches the finishing time, the execution of the run is aborted. Otherwise any roles of head, monitor or gatekeeper that got excluded from the institution in the previous round are reassigned, in this case to some member at random. Then the members are counted (for $P_1$ and $P_2$), the common pool is refilled and the

---

7 Remember that a head, monitor or gatekeeper is a member as well.
next ruleflow group (cfv) is activated.

Experimental data is collected with the rule “create log files”, which is fired as last rule (lowest salience) in the last ruleflow group in each cycle (EXCLUDE).

Agent behaviour

Every agent’s goal in this testbed is to obtain the resource needed to accomplish their task\(^8\). A second goal is to create a resource allocation process that ensures these needs can be met over a long time frame. Therefore, the agents try to match the collective need to the pooled resource. They demand some amount of resource in a time-slice or round \(r\) instead of appropriating it directly and a designated member (the head) will perform the allocation of resources according to an agreed method. It is also the head that initiates a vote on the resource-allocation method. Depending on what goal a particular agent values more, this agent will then only appropriate what the head allocated or exceed that allowance. In certain circumstances an agent unintentionally appropriates more than its allocation. In case a sanctioning scheme has been set up and the agents’ compliancy with the rules is monitored and enforced, the agent can take certain actions to mitigate the sanctioning effects. The agent can either appeal to the sanction to be reconsidered in the allocation process, or it can revise its appropriating behaviour and prevent future sanctions by complying with the rules after being excluded from appropriating for a certain time frame.

In the following, we describe how the agent behaviour is implemented in the different agent classes. The methods in a class (see Figure 4.3) describe the actions that an object of that class is empowered, permitted or obligated to perform. The implementation of these methods is outlined in the following, the line numbers refer to the corresponding code that can be found in Appendix A. Not all rules are implemented as stand-alone class functions, those that are less subjective to the influence of an agent’s characteristics (e.g. role assignment or resource replenishment) are often included in the rules\(^9\).

We start with the methods of the different agent roles, first the member methods (preceded by a reference to the code):

- A.1 demand(): The agents in the open system are assumed to require a standard amount of resource to maintain their operations, their preferredRequest, though they will not place a demand in every round. This amount depends on the compliancy degree (\(= 1\) for compliant, \(> 1\) for noncompliant agents) and the preferred amount becomes higher for noncompliant agents (by the fac-

---

\(^8\)What specific tasks these are is of no concern to this context.

\(^9\)That is purely for ease of implementation, all functions could have been implemented in the corresponding Java class as well.
4. Self-Organising Resource Allocation

tor compliancy_degree). Depending on raMethod they will adjust their demand. For raMethod=queue the demand is the preferred amount (line 5), but demand() is only called if they are not waiting in the queue already. For raMethod=ration the agent becomes more considerate and takes other agents’ demands into account as well as some estimation of the monitoring cost. If the current resource level (minus monitoring cost if pr4=\text{true}, line 12) per agent is smaller than their preferred amount they will demand that (multiplied by their compliancy degree, line 10), otherwise they stick to the usual demand (line 15).

- A.2 vote(): In this case, the vote is cast for raMethod. If the current resource level is smaller than 75\% of the maximum amount possible, then a member votes for ration (line 5), otherwise for queue (line 7). Noncompliant agents are more reluctant in voting for ration and divide the threshold by their compliancy degree (line 4).

- A.3 appropriate(): If P2 is not in use, an agent will appropriate the preferred amount at the same rate it would place a demand (line 16). With P2, an agent will appropriate what it has been allocated (line 13) or, if naughty, take what it demanded (line 10) or top up its allocation if considered too low (line 8). In case unintentional violation upon appropriation happens (which does not affect each agent in every round, line 19), an agent’s appropriation is increased or diminished by a factor depending on the noise_level and average allocation (line 25 or 27 respectively). Note that a member only appropriates from the resource if they are in active state (lines 2, 35).

- A.4 rev_behaviour(): A member is willing to change its behaviour depending on its current level of sanction and maximum level allowed as per the institution (line 2). Changes in behaviour will affect the compliancy degree and the amount of preferred resource (lines 3, 4).

- A.5 appeal(): This method is part of the “members appeal against sanction” rule, which regulates the whole process form appeal to readmission of a member. If P4 is used, a member will appeal as soon as it is being sanctioned (line 6, 10).

- A.5 & A.8 apply(): Being part of both “members appeal against sanction” and “include member after sanction”, the application procedure is implicitly executed when the conditions for inclusion are met.

These are the non-member methods:

- A.6 appropriate(): The frequency of non-member appropriation depends on its compliancy degree and the perceived safety (of not being caught) with respect to this common pool and its institution (line 3). An active agent then appropriates its preferred amount (line 4), inactive non-members appropriate nothing (line 2).
4.2. Principled operationalisation

- **A.14 rev_behaviour()**: The revision of behaviour for non-member takes place as part of the “sanction monitored non-members” rule. Every time, a non-member is sanctioned (line 58), appropriating from that pool is perceived to be less safe and the frequency of further appropriation goes down (line 63).

The methods of the head, monitor and gatekeeper roles are as follows:

- **A.7 assign()**: Agents are assigned membership to a cluster and further roles at the start of a simulation run. When an agent leaves its roles as head, monitor or gatekeeper, for example due to exclusion after a sanctioning procedure, this role to be reassigned to an entitled agent. The rule “assign new head” assigns the role of head to a member at random (line 8), should there be no agent occupying this particular role.

- **A.5 & A.8 include()**: This method is part of two rules, “members appeal against sanction” and “include member after sanction”. The LHS of the first rule (A.5) contains conditions about the last times the member got sanctioned and appealed (lines 7, 8), and if they are within the allowance, the member is automatically included (line 13). The LHS of the second rule (A.8) contains conditions about the last times the members got sanctioned and the level of sanction (lines 7,8). If the conditions for inclusion are met, the agents get the chance to revise their behaviour (line 11 and A.4) and the member gets included (line 13).

- **A.9 exclude()**: There are two rules for excluding a member from the cluster, one where only P4 is in use (“member exclusion”) and one where P5 is used in addition (“member exclusion with graduated sanctions”). The first rule excludes a member after their first sanction, i.e. retracts the member and inserts a new non-member (lines 8, 9), and if they are within the allowance, the member is automatically included (line 13). The second rule checks whether the level of sanctions reached the maximum amount and excludes the member if that is the case (lines 19, 20). Once a member is excluded, its compliancy degree goes back to the value it was before any revision of behaviour.

- **A.10 monitor() & monitor_out()**: Depending on the principles in use (lines 5, 13), the monitor will create a list with members and non-members that are to monitor in that round (lines 8, 16). After the execution of this method (line 33), the Monitored facts are inserted into the working memory (line 35).

- **A.11 cfv()**: If P3 is used (line 2), the head calls for votes in the current round. Different strategies to decide when to call for a vote can be used, here the head initiates a voting procedure for raMethod in every round (line 3), which gets inserted

---

10 This rule does not specify that it is the gatekeeper who performs the role assignment. It is possible to include such a constraint, but then there has to be a rule handling the case when there is no gatekeeper and what role is empowered to assign a gatekeeper role.
4. Self-Organising Resource Allocation

as a fact into the WM (lines 19, 20).

- **A.12 declare()**: In order to declare the result of a vote, the head has to count the votes\(^\text{11}\) for each possible raMethod (lines 11–18, 28–31). The winner is declared according to wdMethod (here plurality, lines 32–36) and updated in the WM (line 37).

- **A.13 allocate()**: The head allocates the resource (which is viewed as an institutional fact here) to members with pending demands according to raMethod, either QUEUE or RATION. Before deciding how to split up the resource between the members, he deducts any cost for monitoring (line 3). If raMethod is QUEUE, then the allocation happens as described in Section 3.9.2, making sure only agents that are not in the queue from a previous round get added to it at random (lines 9–19). If raMethod is RATION, then the resource is divided equally between all demanding members (lines 31–46).

- **A.14 sanction()**: The sanctioning procedure is split into three rules. The first rule concerns members that have not yet been sanctioned and fires when \(P_4\) is used, as a consequence of monitoring actions in that round (line 7). The amount a monitored agent has been appropriating is compared with the amount it was allocated (lines 10–14) and a sanction follows in case of wrongful behaviour (line 18), which sets the agent’s activity state to false (line 20). The second rule concerns repeated offences (\(P_5\)) and evaluates monitored members that have been sanctioned already (line 39). Again, allocations are compared with appropriations (lines 33–37) and agents sanctioned at one level higher than the previous time (lines 39, 40, 42). The last rule concerns monitored non-members. They get sanctioned (line 58) upon detected misappropriation (line 56) and their activity state set to false (line 60). As non-members do not have the ability to get into active state again, no rule for repeated offences is needed.

- **A.5 uphold()**: This method is part of the “members appeal against sanction” rule. A sanction is upheld (i.e. retracted, line 11), when the supposedly offensive member successfully passes both tests on time of last sanctioning (line 8) and time of last appeal (line 7).

The final method described here belongs to the common pool:

- **A.15 refill()** There are four different refill schemes that can be used, HIGH, MODERATE, LOW or CUSTOM. The first three schemes refill the pool with half the maximum resource level\(^\text{12}\) \(P_{\text{max}}/2\) for the first 50 time slices (line 47), then multiply

\(^{11}\) This is a general rule for counting votes and also works for \(\text{ballot}=\text{"head"}\).

\(^{12}\) Jumping ahead to the simulation parameters, this would be the amount that all agents can appropriate using the standard request.
this amount with a high, moderate or low factor (lines 49, 51 or 53). The custom scheme changes the refill rates every 50 time slices and the factor multiplied with $P_{max}/2$ takes values in three ranges, high, moderate or low (lines 10–29).

4.3. Controlled experimentation

In this section we prepare the testbed for resource allocation in open systems, which we created in the last section. We present a subset of the parameters that can be manipulated in our testbed and describe how these parameters are used for experimentation with the testbed. Remember, the open system is $S = \langle A, \mathcal{L}, d, \mathcal{E} \rangle$ at the top level with a cluster $\mathcal{C} = \langle \mathcal{M}, I, \epsilon \rangle$ at the local level that aims to self-organise the resource-allocation process, see Section 3.6. The full specification of parameter settings can be found in Appendix B.

4.3.1. Experimentation with brute facts

The parameters of the environment $\epsilon$ are chosen such that they avoid both superabundance and prolonged insufficiency. In the first case, self-organisation with the aim to impose restrictions on appropriation would be redundant, the second case means that the system would be in constant crisis.

The refill scheme as well as the resource level $P$, size of the pool (maximum resource level $P_{max}$) and environmental factors that lead to unintentional appropriations, are part of $Bf$ in $\epsilon$, the local environment. The threshold $D$ (see Section 2.4.1) that defines the critical level of resource to remain in the pool in order to avoid depletion is chosen as 0.

Refill Scheme The default scheme that is used for resource replenishment is custom with alternating periods of high ($h$), moderate ($m$) and low ($l$) replenishment. This ensures that there are sufficient resources in the long term, provided that the appropriators avoid depleting the common-pool during periods of scarcity. For comparison, we experiment with three more refill schemes: HIGH, MODERATE and LOW.

The refill rates for the custom scheme change every 50 time slices and correspond to half the size of the common pool$^{13}$ times a factor $h$, $m$ or $l$. The sequence of factors is:

$$h \rightarrow h \rightarrow m \rightarrow l \rightarrow h \rightarrow h \rightarrow l \rightarrow m \rightarrow l \rightarrow h$$

$^{13}$maxLevel/2 corresponds to $\text{standardRequest*agents}$, the amount that all member in the institution can appropriate together (when compliant).
where the refill factors are within the following ranges (theoretically, the range of \( l \) starts from 0 but the lowest factor actually used is 0.5, see Appendix A.15):

\[
h \in (0.9, 1], \quad m \in (0.7, 0.9], \quad l \in [0.0, 0.7]
\]

If the refill scheme is HIGH, MODERATE or LOW, the rate of refill starts with 50 time slices of refill at factor 1.0, followed by the factors

\[
h = 0.95, \quad m = 0.8, \quad \text{or} \quad l = 0.5.
\]

**Unintentional Appropriation**  Unintentional errors on appropriation have as effect that in 5% of appropriations, a member will take up to 10% more or less resource than it intended to. The default for unintentional errors to happen is set to false.

**Agent Population**  We start each simulation run with 100 member and 20 non-member agents, all of which are fully compliant (compliancyDegree=1.0) by default. The demand of a member amounts to 50 resource units (when compliant) and is placed in 90% of rounds (when permitted). The first member is assigned the role of head. As role assignment is not the main focus of this experimentation, we decided to integrate the roles of monitor and gatekeeper into the head role.

To experiment with noncompliant agents, we can change the compliancy degree\(^{14}\) of the agents. This results in agents appropriating up to 20% more than their allocation, or appropriate without prior allocation. There are separate parameters to specify how many of the members and non-members are noncompliant (numCheat and outNumCheat).

### 4.3.2. Experimenting with institutional facts

Anything that is a conventional rather than physical state is part of If. That is all the rules and roles in the institution that govern the resource-allocation process, but none of the appropriation or refill phase.

**Principles 1, 2 and 3**  \( P1 \) is needed for the institution to exist, if there are no members that make up rules there is no institution, and is therefore set to true at start. The changeable parameter of this principle is the non-member monitoring frequency and takes an either high (default of 10% non-member per round) or low (1%) value at a relatively low cost (5 resource units per observation).

\(^{14}\)The compliancy degree is a measure of the quantity rather than the frequency of appropriating resources. A degree higher than 1 means that more resources than allocated may be appropriated.
Adding $P_2$ and $P_3$ ensures the manageability of the resource-allocation process. Without $P_2$ all members appropriate without going through a demand or allocation phase, and without $P_3$ there will be an automatic assignment of $raMethod$. This assignment then happens once every 50 time slices, depending on $P$ at that time, i.e. ‘queue’ for $P \geq 0.75P_{max}$, and ‘ration’ otherwise.

**Principles 4, 5 and 6** $P_4$, $P_5$ and $P_6$ are used to protect the common-pool resource from wrongful appropriations through misbehaving members. The changes that can be made to $P_4$ include the frequency of member monitoring which is set to either high (10% by default) or low (1%) at a cost of 50 units. Using $P_5$ does not exclude the member from the cluster for a first offence, but penalises them according to the level of sanction. Up to (and including) the maximum level of sanctioning (3), the member state is set inactive for $5 \times \text{sanctionLevel}$ rounds, for more offences, a member gets excluded. $P_6$ defines the appeals procedure and a member’s sanction is upheld if this agent has not been reported (no sanctions or appeals) within the last 30 rounds (default for appealtime).

### 4.3.3. Goal

The main goal of the open system in each simulation run is the maximisation of the institution’s lifespan $[0, T]$, that is until $P_T < 0$, $M_T = \emptyset$ or $T = 500$. Furthermore, an appropriate trade-off between resource level and amount of members is important, as only sufficient membership can protect the resource against non-members and will be useful in case there is a cost of ownership to cover.

### 4.4. Results

This section describes the experiments conducted with the testbed and evaluates the data obtained. We start with a thorough analysis of four experiments using the CUSTOM refill scheme and present the results of the HIGH, MODERATE and LOW refill schemes after. Table B.6 in the appendix shows a summary of how the parameters are set up for the individual experiments conducted in this chapter, Table B.7 shows the corresponding figures, and all measurements that are taken for experimentation are listed (page 202).

Each of the experimental runs was performed over 100 trials and the following graphs are shown over time (rounds), and the value of the resource level (left) as well as the number of active agents (right) were averaged over all existing clusters in that time slice. A third curve (axis on the far right) shows the number of clusters out of 100 trials that were still active, i.e. their resource had not yet been depleted and there were members.
remaining in the institution. The ranges for the refill rates used during a certain time frame are displayed in rectangles as $h=$ high, $m=$ moderate and $l=$ low.

### 4.4.1. Existence and management principles

Figure 4.6 shows an environment where all *members* comply with the rules and no unintentional errors take place, though 50% of *non-members* appropriate the resource illicitly. With this almost perfect population profile, we test Principles $P1$, $P2$ and $P3$, whose main function is to manage the resource allocation. Four runs with the same environmental settings were tested using a different institutional setup.

![Graphs showing resource level and remaining agents/Clusters for Principles 1, 2, 3]
4.4. Results

To ensure the existence of the institution, we use $P1$ in every trial. It clearly defines the members of the institutions as well as the boundaries through a monitoring procedure. Two different settings of monitoring frequency are used, high (h) and low (l), in order to test the impact of enforcing the boundaries of the common pool. The first run shows the lifespan of $C$ with only $Pr1h$, whereas the second has $P2$ added but the monitoring frequency is set to low, $Pr1l/2$. In these two runs, the resources in all 100 trials got depleted, for the first run this was in round 152, for the second in round 306, both at times when the refill rate became low. We can see that neither a proper monitoring of the border nor the use of a resource-allocation method alone is enough for a sustainable resource and enduring cluster.

The third run has a high monitoring frequency again, $Pr1h2$. This time, about 4/5 of the trials suffer from resource depletion between round 122 and 150, a few more later until only 2 of the clusters remain running by the end of round 500. The high monitoring frequency combined with an automatic resource-allocation policy effects that the institution now better responds to non-member appropriation.

With $P3$ added, 99 trials reach the finish time without the resource being depleted. Enabling the agents to choose $ramMethod$ according to their needs has notable advantages over an automatic assignment. The results from these four runs suggest that all the principles are needed for an appropriate management of the CPR with respect to the prevailing agent population.

4.4.2. Protection principles

Again, we are comparing four runs to each other, see Figure 4.7, this time with a population of 50% noncompliant members and non-members, furthermore unintentional errors happen during the allocation process. From now on, $P1$ is used with a high monitoring frequency.

The run with $P1$–$P3$ in use, results in all but 2 trials being depleted of resources over the course of 500 rounds, due to the (unintentionally) noncompliant behaviour of the agents. Adding $P4$ detects the noncompliant member and excludes them from further allocations. 82 clusters endure until the end, but on average only 21.4 active members remain per cluster. This low number of members is a consequence of the non-graded sanctioning policy that also excludes agents that unintentionally appropriated too much resource. With $P5$ used, noncompliant members get the chance to revise their behaviour and as a result 76.4 members are active in round 500 and no depletion of resource occurs.

An additional $P6$ results in few depletions, 6 in total, all of them before round $t = 30$. This time corresponds to appealtime and suggest that at the beginning of the life cycle, it is not clear whether an agent intentionally or unintentionally appropriates more resource.
4. Self-Organising Resource Allocation

![Diagrams](image_url)

Figure 4.7: $P_4$, $P_5$ and $P_6$: Lifespan of a cluster with 50% noncompliant members and non-members, with unintentional violations.

than allocated, but the policy is to let them off. An advantage of $P_6$ is that more active members remain (86.0) in the institution, as opposed to the runs where a monitoring procedure is in place ($Pr_1$-4 and $Pr_1$-5).

The resource level Figure 4.7(a) shows that there is a high correlation between $P$ and $|\mathcal{M}|$. Especially in $Pr_1$-4, there remain too few members to be able to deplete the resource, yet agents are still excluded. This shows how important $P_5$ is, but also implies that $exMethod$ is needed that can adjust the monitoring frequency (given that there is a cost implied) and grades of sanctioning to the amount of resource and number of agents.
4.4.3. Alternatives from the parameter set

With the following experiments we show that it is important to carefully tune the implementation of the principles to the environment \( \epsilon \). To this end, we conduct two sets of experiments with different agent populations and test \( P_4 \) using a high and low monitoring frequency.

![Graph showing resource level and remaining agents/clusters for different monitoring frequencies.]  

Figure 4.8: \( P_4 \) ‘good’ population: Lifespan of a cluster with all members compliant and 50% noncompliant non-members, no unintentional violation.

The first set of experiments contains three runs where all members are compliant with the rules, no unintentional errors happen and 50% of non-members appropriate illicitly from the resource. These are the same population settings as in Section 4.4.1. Figure 4.8 shows that (as before with \( P_1, P_2 \) and \( P_3 \)) only 1 cluster depletes their resource during the life cycle. Adding \( P_4 \) with a high monitoring frequency (Pr1-4h) considerably
diminishes the amount of resource left at the end of a round during times of medium
and low replenishment, 3 clusters deplete their resource midway. Setting the monitoring
frequency to low (Pr1-4l) leaves the members with sufficient resources. Given that there
are only compliant members in the institution all of them remain in their cluster in all
trials, member monitoring is theoretically not needed. The experiments confirm that
run Pr1-3 is a best and Pr1-4l a close fit for the environment.

The second set of experiments also contains three runs, but this time there are 50%
noncompliant members in addition to 50% noncompliant non-members. Again, there is
no unintentional appropriation of resources.

Figure 4.9: $P_4$ ‘bad’ population: Lifespan of a cluster with 50% noncompliant members
and non-members, no unintentional violation.

Figure 4.9 shows that when no monitoring takes place (Pr1-3) all agents remain in the
clusters, but during times of low and moderate resource refill they manage to deplete the resource, so that there are only 9 clusters that endure until the end. This time, the expense of additional resources for monitoring pays off when adding $P_4$ with a high monitoring frequency. 88 clusters are able to sustain their resource and they exclude almost all of their noncompliant members, so that only 56.9 remain on average. When the monitoring frequency is set to low, the institution is not as successful in catching the misbehaving members, in total 67 remain on average. The downside of a higher noncompliant membership is that they manage to deplete the resource in 65 trials.

4.4.4. Comparing the refill schemes

After running experiments with the \textsc{custom} refill scheme, the scheme that we used for experimentation in \cite{93}, we repeated each run with the same environmental and institutional profile but different refill schemes for this work. The refill rate for the \textsc{high} scheme is $h = 0.95P_{\text{max}}/2$, for \textsc{moderate} it is $m = 0.8P_{\text{max}}/2$ and for \textsc{low} $l = 0.5P_{\text{max}}/2$, after an initial 50 rounds of refill rate $P_{\text{max}}/2^{15}$.

We have three measurements per run. Firstly (a), the agents that remained active at the end of a trial given as average per remaining cluster (including standard deviation as error bars). Secondly (b), the number of trials where the cluster endured throughout 500 rounds. Where no cluster endured, the round when the last cluster got bankrupt is mentioned instead. And thirdly (c), the appropriated resource per cluster over the course of the whole life cycle (as average including the standard deviation as error bars), which includes member and non-member appropriations. A grey bar at the right of each group is the theoretical maximum of resource that all members can appropriate in any one trial given the current refill scheme and when appropriating the standard of 50 resource units in 90% of rounds. This last measure can be regarded as the efficiency of the resource allocation and uses the concept of utilitarian social welfare according to \cite{28}. The efficiency is an objective measure for conducting a quantitative evaluation of the system.

Existence and management principles

In Figure 4.10, we can see these three measurements for four different refill schemes, using the same population profile as in Section 4.4.1.

The principles that are used for the experiments do not include a monitoring or sanctioning procedure, therefore all members remain in the clusters and only the runs where

\footnote{This initial refill period is set to match the theoretic demand, based on the fact that with $P_1$ the initial number of members should have been chosen smaller otherwise.}
4. Self-Organising Resource Allocation

Figure 4.10: P1, P2 and P3: Comparison of refill schemes in an environment with all members compliant, 50% of non-members noncompliant, no unintentional violation.

the pool of all trials was depleted has no members left, compare Subfigure (a) and (b). For the case with high refill rate, all combinations of principles are successful and the efficiency, see Subfigure (c), reaches the theoretical maximum\(^{16}\). For a moderate or low refill rate P1 and P2 only are not enough to sustain the resource, furthermore, the efficiency is relatively low with values of 14% (m) and 18% (l) of the respective maxima.

The addition of P3 in these cases results in only few trials being depleted and the efficiency increases to 98% or 93% for a moderate or low rate respectively. For the three runs with a constant refill rate, the difference between a low and high monitoring level

\(^{16}\)In all these cases the standard deviation is typically higher for runs with more depletions, especially if the times of depletion are spread out so that the agents get a longer or shorter time frame to appropriate, see Figure 4.6, C.1, C.5 and C.9.
does not become apparent within a group as the times of resource depletion are relatively early. When the rates are mixed as in the custom scheme, we can see that although high monitoring comes at a higher cost, the amount of resource that the agents are able to appropriate over 500 rounds increases by 7%.

**Protection principles**

The next set of experiments, see Figure 4.11, has 50% noncompliant members and non-members as population profile, and on top of that, unintentional errors occur upon resource appropriation.

Figure 4.11.: P4, P5 and P6: Comparison of schemes in an environment with 50% noncompliant members and non-members, with unintentional violations.

The lack of an *exMethod* (Pr1-3) shows that, apart from a high refill rate, the members are not excluded from the institution but deplete the resource in nearly all trials. As
4. Self-Organising Resource Allocation

As a result, the efficiency lies by only 40%, 26% or 24% for custom, moderate or low rates respectively. If the refill rate is high, then Pr1-3 proves to be most efficient and even surpasses the theoretical maximum by 3% due to successful (but not depleting) illicit resource allocation. With a standard deviation\textsuperscript{17} of 10%, we can see that the agents are able to achieve a considerable increase on appropriation.

Adding $P_4$, member monitoring and exclusion at a first offence, brings the membership down below 29 remaining agents per remaining cluster (which are 48 for a low refill rate, but above 82 clusters for all other schemes). Both decreased membership and remaining clusters have an effect on efficiency which is 56\% (m) at its best. The fact that there are less members excluded for a high than for a moderate or low refill rate is due to the fact that noncompliant members only cheat on appropriation if they did not get an allocation close to the amount they asked for (see A.3, lines 6–8).

With $P_5$ in use, all clusters manage their resources in a sustainable way so that no depletion occurs in none of the refill schemes. Furthermore, membership increases by a factor between 3.3 and 4 in comparison to Pr1-4. This is due to the fact that with the graduated sanctioning mechanism, noncompliant members now get the chance to revise their attitude towards resource allocation after each offence. The efficiency for these experiments increases by a factor between 1.5 and 2.1.

When we add $P_6$ to the institutional rules there are two effects: on the one hand, the membership further increases to up to 99 for a high refill rate, and on the other hand, the number of remaining cluster decreases and is 90 at its lowest point (l). This is again caused by the fact that during an initial period of appealtime, the members are let off for intentional misappropriation, resulting in resources being depleted during that time. Afterwards however, the $adrMethod$ manages to keep the membership higher than is for the case without $P_6$. In terms of efficiency, this has on average benefits for a high and medium refill rate, though the standard deviation is relatively high due to early dropouts (compare Figures C.1, C.5 and C.9 in the appendix). For a low refill rate the number of early resource depletions is too high, so that the agents in the remaining clusters are not able to achieve a higher efficiency than in the previous run (Pr1-5).

Depending on the nature of the open system, the outcome of either Pr1-5 or Pr1-6 might be preferable in order to achieve higher membership or higher sustainability. And there is always the option of combining the two approaches and take the appeals procedure in effect after the initial period of appealtime only.

\textsuperscript{17}This time, the variation in the standard resource is further influenced by the fluctuation of active member agents, see Figures 4.7, C.2, C.6 and C.10.
Figure 4.12: $P4$ ‘good’ population: Comparison of refill schemes in an environment with all members compliant and 50% noncompliant non-members, no unintentional violation.

**Alternatives from the parameter set**

Now we examine the effect of changing an institutional field variable, the member monitoring frequency, on two different population profiles, the ‘good’ one with no noncompliant members (50% noncompliant non-members), and the ‘bad’ one with 50% noncompliant members (and non-members). In both cases no unintentional violation of appropriation rules occurs. The monitoring frequency can take two values, high (Pr1-4h) and low (Pr1-4l), and is compared to the case where no monitoring takes place (Pr1-3).

The first set of experiments using a ‘good’ population is shown in Figure 4.12. As expected, no members are excluded from the cluster (a) and only when the refill rate is low, the non-members manage to deplete the resource completely (see (b) custom
4. Self-Organising Resource Allocation

Figure 4.13: $P_4$ ‘bad’ population: Comparison of refill schemes in an environment with 50% noncompliant members and non-members, no unintentional violation.

and low). As a result, the efficiency in the all refill schemes apart from low reaches the theoretical limit (or by 99%). With a high monitoring rate, there are more resources depleted than without monitoring at all, but with a low monitoring, the amount of remaining institutions is as high or even higher$^{18}$. This also increases the efficiency of the runs to a value the same as or slightly higher than without monitoring.

The second set of experiments is with a ‘bad’ agent population, see Figure 4.13. For $Pr_{1-3}$, all members remain active in their institution as long as the resource is not

$^{18}$An explanation for more remaining clusters is that the agents are likely to overestimate the monitoring cost and lower their demands ($raMethod$ is ration with this refill rate). This means they do not appropriate all possible resource and illicit appropriations from non-members do not deplete the resource as easily. Underestimation of monitoring costs results in an allocation $\leq$ demand, thus has no effect for a ‘good’ population.
depleted, which is the major problem for this population profile. Only with a high refill rate 97 clusters endure, in the other cases this amount decreases to 9 (custom) or 1 (m), with a low rate the resource in all trials is depleted by round 350. The efficiency is accordingly low, between 23% and 47%, apart from the high refill scheme where the theoretical maximum is exceeded by as much as 4%.

Introducing monitoring with a high frequency (Pr1–4h) brings the membership down to 57 for custom and low refill schemes, but the number of remaining clusters increase substantially for custom, moderate and low schemes. For high and moderate there remain 99 clusters and for custom and low remain 88 and 37 respectively. The efficiency increases by a factor between 1.5 and 3.6 in all cases apart from the high scheme where it decreases by 12% due to fewer members in only two remaining clusters more and high monitoring costs. Changing the monitoring frequency to low results in more members staying active, but decreasing cluster numbers (apart from high refill which is now 100). This affects the efficiency which is decreased in comparison to the high monitoring frequency, but still higher than with no monitoring taking place, in cases for custom, moderate of low refill. For a high refill rate, the efficiency surpasses the theoretic maximum by 3%.

### 4.5. Evaluation

The data obtained from the testbed simulation and presented in Figures 4.6–4.9 (custom refill scheme) demonstrates well that it depends on the specific agent population and environment how the principles will perform. When the agents conform to the rules, Principles $P1–P3$ are enough to manage the resource-allocation process. As the assumption on the compliancy of the agents is relaxed, these principles are no longer enough to ensure the sustainability of the resource and Principles $P4–P6$ become necessary. The findings from these experiments coincide with the findings we obtained from experimentation with our previous testbed [93], which adds credibility to our results.

In order to test and compare the robustness of principles with respect to prolonged periods of scarce resources, we have simulated three more replenishment schemes, HIGH, MODERATE and LOW. The results obtained from these simulations reinforce the findings from the first set of experiments. Principles $P1$, $P4$, $P5$ and $P6$ are used to respond to unintentional and intentional violation of $If$, the institutional facts, and to change the agents’ activity state according to Figure 4.4 (page 86). Moreover, Principles $P2$ and $P3$ allow the agents to adapt to changes of $Bf$, the brute facts in the environment, such as a high or low resource level.

As we include more and more principles for creating rules in the institution, we can greatly increase efficiency (i.e. utilitarian social welfare) in these systems. The only exceptions to this are cases with abundance on resources where all cost spent on managing
4. Self-Organising Resource Allocation

and monitoring is wasted (HIGH refill scheme). For all other refill schemes we observe the following pattern:

- **P1, P2, P3** (‘good’ population, no unintentional violation, Figure 4.10(c)): The efficiency increases with the rate of refill and increased number of used principles, which is the much stronger effect.

- **P4, P5, P6** (‘bad’ population, unintentional violation, Figure 4.11(c)): The efficiency increases with the rate of refill and increased number of used principles in the case of custom and moderate refill. For a low refill rate, the leniency of the \textit{adrMethod} results in a slight decrease of efficiency when \textit{P6} is used.

- **P4**, high/low monitoring frequency (‘good’ population, no unintentional violation, Figure 4.12(c)): The efficiency increases with the refill rate, but becomes lower for high expenses on monitoring.

- **P4**, high/low monitoring frequency (‘bad’ population, no unintentional violation, Figure 4.13(c)): The efficiency increases with the refill rate and becomes considerably higher for high expenses on monitoring.

The contrast of efficiency in the last two sets of experiments is compared in Figure 4.14. The plain bars denote the ‘good’ population and the striped bars the ‘bad’ one. Throughout, the ‘good’ agents are more efficient than the ‘bad’ agents apart from when there is resource in abundance. If we compare the outcome of a ‘good’ population to the price of anarchy in the proportional allocation mechanism in Section 2.7.4 (where compliant \textit{members} and non-\textit{members} are a precondition), we can see that in all cases but one (that is 0.73 for Pr1-4h at a low refill rate), the price of anarchy is close to 1. This is a much improved result to the game-theoretic approach where the price of anarchy is 0.75 and can only become 1 if we assume perfect information (i.e. the appropriation sequence of the agents is publicly known at all times).

Comparing these two sets of experiments makes it even more apparent that the selection of principles according to the environment is crucial. In cases where monitoring is not strictly needed, that is for compliant agents or lots of resource, the clusters are better off with a low monitoring frequency (see also Figures 4.12(c) and 4.13(c–high)). But in cases where the resource becomes sparse (due to agent behaviour or refill rates), a high monitoring frequency leads to a more sustainable resource and enduring clusters (compare Figures 4.13(b) and (c)).

The broad range of experiments that we conducted allows us to answer [Q3]. We successfully implemented a testbed using the formal axiomatisation given in Section 3.9 and showed that the design principles for CPR institutions (page 38) are sufficient conditions for creating enduring open multi-agent systems that can manage a common-pool resource in a sustainable manner.
There are a few limitations in this testbed. They become apparent especially where the success of the chosen principle (or rather the chosen field value) is related to the agent behaviour, such as the high or low monitoring frequency in the last two sets of experiments; or to the environment, such as the use of \textit{P6} combined with a low refill rate in the second set of experiments.

This shows that all principles have to be carefully implemented with respect to the local environment $\epsilon$ and that a trade-off for a single parameter that suits all possibilities cannot be made. Consequently, we need additional mechanisms for the agents to firstly learn about the environmental states (including the behaviour of other agents) and secondly respond to them in an appropriate manner. There is a wide range of independent and interdependent variables that all have to be analysed and the following question arises:

\begin{itemize}
\item \textbf{Q4} Can we equip the agents with mechanisms to evaluate the self-organisation?
\end{itemize}

One of the challenges is to estimate the overall utility one can obtain by making changes to the institution $\mathcal{I} \in \mathcal{L}$. This utility depends on the cost such a change involves, according to the distance function $d$ (see Section 3.6), to make the change in the first place, the new running cost of the altered $\mathcal{I}$, and time frame that such a change is going to be suitable for $\epsilon$. According to [83], understanding the cost of a system is key to understanding the benefits of the institution.

To meet this challenge, we need appropriate mechanisms for runtime self-analysis. Such
mechanisms include perceiving agents that reflect on events and deduce consequences from prospective actions, i.e. agents that are self-aware and choose their policies in line with the environment and so favourably influence the behaviour of noncompliant agents. In the following we will explore the possibilities of responding to particular agent behaviour within the institutional context.

4.6. Summary

In this chapter, we showed that it is possible to use the formal axiomatisation of Ostrom’s design principles to implement a testbed for managing resource allocation in open systems. We furthermore showed that this is not just possible, but also that these principles are sufficient conditions for creating enduring open systems for sustainable resource management. We therefore conducted a range of experiments, using different sets of parameters to test the influence of $P_1$–$P_6$ on the variation of resource replenishment and agent behaviour. We found that when the agent population predominantly complies with the rules, $P_1$–$P_3$ are sufficient to manage the system. When we relax the assumption of compliance, $P_4$–$P_6$ are needed to make the system robust to rule violations, but it can manage a remarkable 50% noncompliant agents.
5. Awareness and Fairness

5.1. Introduction

To answer Q4, we go back to the methodology of sociologically-inspired computing (Section 3.2) and evaluate the theory, formalism and computer model. The preformal theory of Elinor Ostrom expresses explicit requirements on the self-governance of a common-pool resource, but the assumption of human actors that reason about their individual, sociological and organisational context is merely implicit.

In open systems however, the assumption on the behaviour of agents has to be made explicit. To this end we complement the preformal theory with the theory of self-awareness, which gives agents the capability of reasoning about themselves and consequences of their actions with respect to the institution and environment. This enables them to not only decide on rules that are suitable for sustaining the resource, but also on rules they are satisfied with and perceive as fair.

In the context of an organisation (or agent cluster), the fairness of concern is typically from the area of organisational fairness. Here, we let the agents evaluate the fairness of procedures within their cluster, which has an impact on their satisfaction. Several options will be made available to the agents for reacting to perceived unfairness and reduced satisfaction.

We start with an analysis of awareness and self-awareness in the field of psychology and neuroscience that we also presented in [105], and then define with what level of self-awareness to endow the agents in open systems solving the resource-allocation problem. Afterwards, we illustrate the concept of organisational justice from a psychology and behavioural economics point of view. In order to explicitly express the reasoning process leading to fair and satisfying procedures and outcome, we utilise the agents’ self-awareness in combination with a theory of fairness perception with respect to procedural justice. We include the formalisation of P7 and P8 into the computer model, which serve as a tool for the agents to self-organise the procedures in the open system.
5. Awareness and Fairness

5.2. Self-awareness for self-organisation

Autonomic systems have the capability to perceive the environment in which they are embedded and leverage this information for self-organisation, self-adaptation, self-healing, etc. In order to do this successfully, the components can benefit from some notion of awareness or self-awareness.

5.2.1. Self-awareness in computer science

Computer science has a well established idea of awareness. Due to the lack of a universally agreed definition—also in neuroscience and psychology there are many crossovers to what it means to be aware or self-aware—computer scientists rely mostly on an intuitive definition of awareness. Only few attempts to categorise (self-)awareness have been made in surveys and reviews on this matter, as in for instance [3] or [71].

In artificial intelligence, the idea of (self-)awareness is related to artificial consciousness, introspection and meaning. [75] discuss self-awareness in humans and machines, focussing on the usefulness of self-awareness for machines, whereas [107] focuses on how the knowledge derived from self-awareness can be represented. In networks, the idea of (self-)awareness involves performance measuring against established metrics, for example Quality of Service, and changing the structure or behaviour of the network accordingly. Examples include proactive node, link and path discovery as discussed in [48], and the ability to self-configure, self-provision and self-monitor a network to guarantee quality of service [66]. Autonomous systems are systems that can manage, organise and repair themselves even under the assumption of conflicting goals\(^1\). However, according to [97], to initiate self-repair, the system needs to be aware itself that there is something wrong with it. To achieve this, the system has to learn about its performance over time and infer from the system’s parameters and performance what actions to choose and what outcomes to predict with respect to the environment. Self-awareness can furthermore be interpreted as a form of introspection on beliefs, desires and intentions, as used by [89] to develop an architecture for emotional agents. In a sociologically-inspired system, self-aware agents reason about their role or position within that system and in relation to other agents for social or organisational purposes [72].

---

\(^1\)For a CPR that would be the maximisation of individual appropriations vs. the sustainability of the resource.
5.2.2. Levels of awareness in social science

Our aim here is not to attempt a definition for self-awareness, but to present a scheme for different levels of awareness, following some of our earlier work [105]. These are inspired by Morin’s levels of consciousness and self-awareness [79] and complemented by models about the self from psychology or neuroscience. In [99], for example, five levels of awareness are presented as they develop in children, and [29] discusses to what degree self-awareness can emerge from actions. Figure 5.1 gives an overview over how we define these levels of awareness which we explain in detail after.

Figure 5.1.: From unconsciousness to meta self-awareness: Different levels of awareness

**Unconsciousness** At the lowest level is unconsciousness which can be further divided into deep and light unconsciousness. Deep unconsciousness refers to a person deeply asleep or in a coma, and light unconsciousness to a person that is dreaming or mentally active, but not processing any internal or external information.

**Consciousness** The next level, consciousness, is split up into minimal, spatial and recursive consciousness. A minimally conscious person is awake, registers stimuli, and processes automated cognitive or sensorimotor programmes. A spatially conscious person is aware of their position and (voluntary) movement relative to the environment. A recursively conscious person is able to relate past and future events to the present state,

---

115
5. Awareness and Fairness

to perceive their phenomenological content, and to respond to actions with a delay [37]. So far these instances of awareness do not include a notion of the ‘self’.

**Self-awareness** The next level is *self-awareness* which is sometimes also referred to as *meta consciousness*, though the term self-awareness covers a wider spectrum [106]. This level is concerned with the construction of a coherent mental model of the self, and can be divided into five more specific levels, namely self-consciousness, and public, private, persistent and predictive self-awareness.

*Self-consciousness* extends the level of recursive consciousness and includes an analysis of the self over time and differentiates the self from others. With *public self-awareness* we mean perceptual information that relates to behaviour and physical appearance, meaning a person is aware of their own representation and the effects of their actions [2]. *Private self-awareness* is the conceptual information about the self, which is gained through reflecting on internal aspects like emotions, goals and physiological sensations. With *persistent self-awareness*, the reflection on the self is extended over time, from the past to some anticipated future. *Predictive self-awareness* goes one step further and not only mentally models the self, but also the inner states of others. That includes the reflection on future environmental states, and consequences of, or reactions on actions.

**Meta self-awareness** At the top level of the scheme is *meta self-awareness*, the awareness of being self-aware or self-observing, split up into self-concept and iterative self-awareness. *Self-concept* is the ability to analyse own thought processes in order to abstract from the self, leading to a classification according to roles, value, identity and character [80]. Finally, *iterative self-awareness* is an extension from predictive self-awareness where a person constructs “mental models of other people’s mental models of someone’s mind” [81].

The above levels are not strictly dependent on each other. Empathy, a sign of predictive self-awareness, for example, does not necessarily presume public self-awareness. We tried to produce a coherent scheme that includes the various levels and notions of awareness, aiming for a clear distinction between each one of them, despite the ambiguity of terminology and level transition in the literature.

5.2.3. Awareness in open systems

For creating and revising rules of an institution, awareness naturally influences human decision making. The level of awareness used depends on individual capabilities and the task they want to achieve. Typically, most levels of (self-)awareness are used, even if
5.2. Self-awareness for self-organisation

the individual is unaware of that. In artificial systems however, only a subset of the presented levels of awareness can be reasonably applied to a certain process\textsuperscript{2} and we will identify how open systems for resource allocation considered in this work can benefit from using some notion of awareness.

As mentioned in Section 4.5, we need to ensure that the rules (and their parameters) are chosen such that the resulting norms for appropriation are robust to varying environmental conditions. The problem is that even though norms define how an agent is expected (by other agents) to behave in the cluster or system, autonomous agents can still deviate from this behaviour. We already looked at possible sanctions when an agent does not follow the rules. Now we want to look at sanctioning methods if an agent does not follow the norm\textsuperscript{3}, and possible reactions are to stop interacting with the agent that violated the norm [104] or with the cluster as a whole, or to change the rules to match the norm.

To this end, the agents will be endowed with the capability of exhibiting predictive self-awareness. They will simulate internally how they expect an agent occupying a particular role to perform, according to their individual perception of norms. As an example take the head performing a resource allocation. The state of the environment and institutional facts are evaluated by the individual members (here these are the allocations with respect to the resource level) and if the changes they observe do not coincide with their expectations, they can decide to collectively reassign the role of head to a different member.

Self-awareness can also be helpful in judging the rules in place and setting their parameters accordingly. In order to adjust the monitoring frequency used in $P_4$, for example, the members have to evaluate the consequences of agent behaviour given the state of the environment. Then they can decide whether it is sensible or necessary to increase (e.g. for noncompliant agents appropriating from a scarce resource) or decrease (e.g. ‘good’ agents or plenty of resource) the monitoring frequency. Judging these conditions requires recursive consciousness and public self-awareness as the basic functionalities. More sophisticated reasoning is possible using higher levels of awareness. Other examples that we mentioned before and where awareness can be a beneficial concept include the revision of an agent’s behaviour (by the agent itself) or the definition of a refined appeals procedure.

Regarding the answer to Q4. Endowing the agents with self-awareness constitutes one such mechanism that can be used for evaluating the self-organisation of open systems from an internal perspective. As opposed to the quantitative evaluation we conducted

---

\textsuperscript{2}For example, some applications benefit from modelling spatial consciousness, for others this is not even an option.

\textsuperscript{3}In this context, to follow a norm means to perform as ‘expected’, i.e. there are no explicit rules prescribing certain behaviour.
in Chapter 4, this represents a qualitative measure of self-organisation. This leads us to the next question:

\[ Q5 \] Does a qualitative evaluation of processes enable the agents to make more informed choices with respect to self-organisation?

5.3. Organisational justice in open systems

Two of the major reasons for groups developing rules are to maximise the satisfaction of the involved parties \[14\] and to change the structure of incentives in particular situations \[84\]. One aspect that influences satisfaction in an institutional setting is how the individual actors perceive fairness within the organisation. This section introduces the field of organisational justice \[51\], elaborating on the area of procedural justice for resource allocation. We explain how procedural justice is evaluated and formed by human actors and how we can apply this theory to agents of an open system using the concept of (self-)awareness.

5.3.1. Organisational justice

Typically, the term fairness or justice is used in situations where conflicts arise, such as the distribution of a common-pool resource, payments of premiums or cuts \[74\], or in arbitration. Organisational justice comes in three main forms and is concerned with the perception of fairness in organisations \[30\]. For each of the three forms we give a typical statement from a member of an organisation experiencing this type of justice \[63\].

- **distributive justice**: “I receive fair rewards in this organization.”
- **procedural justice**: “This organization makes decisions in fair ways.”
- **interactional justice**: “In interpersonal encounters, my supervisor gives me a fair treatment.”

**Distributive justice** is concerned with the fairness of distributing resources such as pay, rewards, promotions or the outcome of dispute resolutions. That is, how the prevailing rules are being applied. In the context of open systems, one example of unfair distribution is the effect of noncompliant behaviour, meaning the end distribution of resources (individual appropriation) is not performed according to the rules previously specified.

**Procedural justice** is concerned with the decision-making processes that lead to a particular outcome and the amount of control that the procedures allow individuals over the outcome \[19\], e.g. the combination of allocation and sanctioning procedures. As direct
control over the outcome is often not possible, people try to control it indirectly. In the open systems scenario, that includes the agents’ right to vote on a specific issue for it to be changed.

Interactional justice is concerned with fairness related to the nature of interpersonal treatment by colleagues and authorities. To realise interactional injustice in an artificial environment one would need to implement the equivalent to rude behaviour or propagation of misleading information.

In [116], several studies are performed to test the effects of distributive and procedural justice and the results suggest that a variation on fairness of the procedure has much stronger influence on satisfaction than a variation on the outcome. Reviews of further studies confirm that procedural justice is much better suited to predict organisational commitment and trust towards supervisors than distributive justice, which is typically related to specific outcome only, such as payments for example [78]. Interactional justice leads to a further increase (or decrease if absent) on job performance and motivation for compliance [32].

5.3.2. Procedural justice perceptions

The concept of procedural justice relates to the perception that an individual has on the procedures regulating the distribution of resources. This means that the satisfaction is not only influenced by the final distribution, but also by the sequence of events that take place prior to the actual distribution, such as the selection of a distributor and the process of decision making.

In [70], Leventhal defines seven procedural components that are typically evaluated by individuals to establish a fairness judgement for an allocation process. For a fairness judgement, an actor does not have to perform a ‘full’ evaluation, but evaluates any of the seven components and aggregates the partial results to a final judgement.

Considering the institutional approach for managing resource allocation in open systems, the procedural components to be evaluated are (fully or partially) integrated in the principles for designing institutions for long-enduring CPR management.

- Selection of agents—Appropriate agents who serve as decision makers or information collectors have to be chosen. This element is related to $P3$ and $P4$, meaning that decisions and monitoring are made by the agents themselves (or appointed roles).

- Setting ground rules—Potential receivers of rewards are informed about performance goals and evaluation criteria that must be met to obtain the reward. This is similar to $P2$ and $P5$ which regulates the process of demand and allocation, and
5. Awareness and Fairness

possible sanctions (negative reward) if the ‘compliancy criteria’ is not met.

○ Gathering information—Reliable information has to be collected to be able to evaluate potential receivers, also the type of information that can be used should be declared. $P_1$ and $P_4$ are in charge of monitoring agents and the environment, thus cover this element.

○ Decision structure—A variety of processes are possible that define the structure of how the final allocation takes place and who is eligible to perform it, see $P_2$.

○ Appeals—Dissatisfied agents should be given the possibility to appeal to a distribution or to actions that lead to a particular outcome. Like in $P_6$, the appeals procedure can be informal or more structured.

○ Safeguards—There have to be procedures that serve as ‘safeguards’, meaning they ensure that the allocation process is performed according to the rules specified and that nobody illicitly takes more reward than allocated. This element reads just like $P_4$ and $P_5$ and states that there have to be monitoring and sanctioning procedures in place.

○ Change mechanisms—This final set of procedural elements includes methods that regulate how the allocation procedures can be changed in order to correct unfair situations, and is related to $P_3$ and $P_7$.

$P_8$ affects nearly all of the above components, as the fairness judgements can be repeated on every level of a nested enterprise.

Given the overlap of procedural components and design principles, no additional elements need to be integrated into the preformal theory, and the agents can directly evaluate the specification of principles to arrive at their fairness judgements.

### 5.3.3. Three norms of justice

Many different factors are used to assess fairness within a particular context and relate to job satisfaction, compliance with organisational rules, or the intention to leave the organisation or company. According to [32, p. 309], “work attitudes and intent to turnover are sensitive both to rewards and the fairness or the organizational system” and [63] examines how different types of justice affect job satisfaction and the intention to leave the organisation.

In the field of Fairness Theory, the fairness of an event is evaluated by going through three counterfactuals and individuals determine whether the outcome could, would or should have been different given a specific situation, as explained in [43].
The three most frequently used distribution norms in the literature of psychology and behavioural economics are equity, equality and need\(^4\), as for example presented in [70] and [74].

- **Equity** is the allocation of resources that reflects past contributions or achievements of individuals;
- **Equality** means that every individual receives an equal allocation of rewards or resources, regardless of performance or potential; and
- **Need** is the norm for distributing resources according to the relative need of an individual.

Norms evolve differently according to the sociological context, and what distribution norm is used by a certain organisation depends on the organisational culture. Equity is typically used in organisations that want to increase their productivity, equality is used in a context based on teamwork or when future interactions with the individuals are anticipated, and the need norm is often found in settings with close personal relationships. The organisational culture, in turn, is usually biased by the culture of the society an organisation is rooted in. Many studies have been conducted to investigate the cultural differences in fairness perceptions.

In [63], the authors examine how distributive, procedural and interactional justice (as defined in Section 5.3.1) are related to overall fairness and how it affects job satisfaction and the intention to leave in the US, China, Japan and Korea. The findings include that distributive justice had a much lower influence on overall justice perception for Americans and Japanese than it had for Koreans and Chinese. For interactional justice (interpersonal treatment) the proportion of influence was reversed. As a response to perceived fairness, Americans are much more likely to leave a company than Chinese or Koreans, and also the job satisfaction was much more influenced by fairness for Americans than employees of the other countries. Another important factor when evaluating fairness is the ‘power distance’, the unequal distribution of power within an organisation. In countries with a flat hierarchy, employees are much less tolerant to unfair behaviour.

In [69], cultural effects are assessed with respect to (current) situational events and it is argued that socio-economic conditions can override cultural influences when choosing a distribution rule. Resource-scarce countries prefer need as a distribution norm more often than culturally similarly countries that are better off. The same override of cultural influences was observed in different socio-political conditions (compared were former East- and West-Germany) and, as mentioned before, for different situational goals such as solidarity or productivity.

\(^{4}\)Note that these definitions might not be equal to those found in the computer science literature.
Another study of cross-cultural similarities and differences, as presented in [14], came to similar conclusions, that when the basic requirements on food, clothing and shelter are in doubt, need is the preferred distribution norm, and that this preference changes when the minimum requirements are met for the majority of the population\(^5\).

The figures in Table 5.1 are derived from experimental data in [14, p. 62]. We have two different agent populations, a capitalist one and a socialist one. In each population there are different proportions of agents that act according to one of the three distribution norms equity, equality and need.

<table>
<thead>
<tr>
<th>distribution norm</th>
<th>equity</th>
<th>equality</th>
<th>need</th>
<th>(\wp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>capitalist</td>
<td>1/2</td>
<td>1/3</td>
<td>1/6</td>
<td>3/4</td>
</tr>
<tr>
<td>socialist</td>
<td>1/6</td>
<td>1/3</td>
<td>1/2</td>
<td>1/2</td>
</tr>
</tbody>
</table>

Displayed are the proportion of agents that choose that norm under ‘normal’ or ‘regular’ environment conditions, and \(\wp\) is the probability with which an agent is likely to change to a different norm under ‘crisis’ environment conditions. Possible changes are\(^6\)

\[
\text{equity} \xrightarrow{\wp} \text{equality} \quad \text{and} \quad \text{equality} \xrightarrow{\wp} \text{need}.
\]

The next section shows examples for the formal characterisation of rules that promote fair procedures, techniques that can be used for judging fairness will be integrated in the subsequent chapter.

5.4. Fairness in open systems

So far we used one type of fairness measurement in this work. This type evaluates the resource distribution (efficiency) from an external perspective, using the concept of utilitarian social welfare, see Section 4.5. This measurement was taken in retrospect to inform us about the adequacy of parameter choice. Various alternatives for taking measurements from an external viewpoint are defined in [68], using five axioms that categorise a measure as fair. Examples of such fair measures are the Gini coefficient, \(\alpha\)-fair utility or Jain’s index.

\(^5\)In the next chapter, we will use their results to set up the agent population for the simulation testbed.

\(^6\)The original work did not specify whether changes from equity to need directly were made and how high this percentage would be, so we chose to allow on ‘single hops’ only.
In open systems however, an evaluation from an external perspective cannot be used for runtime decision making. Therefore, we use a type of fairness measurement that evaluates the procedure of distribution from an internal perspective, as is the case in the field of organisational justice. This constitutes a subjective evaluation method.

Neither of those two measurements answers the question whether the open system is fair. To answer this question, one has to define first what they understand by ‘fair’. In [120], internal agent fairness is not defined using social concepts, but based on a clear mathematical formulation, which also means that all agents to have the same understanding of (formula for) fair behaviour. Computational social choice [28] is however a concern in [61], where multi-agent systems are used to model fairness factors that make humans deviate from purely rational behaviour.

We will extend the preformal theory with a fairness evaluation method (feMethod), the functionality of judging procedural justice, using two of Leventhal’s procedural components, see Section 5.3.2. Therefore we assume a cluster where the agents have different cultural, political or economic backgrounds, they have to compare the head’s performance to their internal fairness perception and take action for change. The procedural component we address here are ‘selection of agents’ and ‘change mechanisms’, meaning the agents evaluate the decision maker (head) and can make changes to that role or the context they are situated in. For experimenting with these changes, we now axiomatise the remaining two design principles of Elinor Ostrom (page 38), P7 and P8.

The feMethod complements the agent model as follows. The agent in the head role allocates resources to agents according to raMethod=queue. This time, the formation of the demand queue does not happen at random, but according to the head’s internal norm of distribution, either equity, equality or need. This norm does not have to be static but can change according to the environmental circumstances (here the pooled resource). The members then judge individually whether the head uses a (to them) fair procedure. Depending on the outcome, their satisfaction will increase or decrease, and they can subsequently vote for the head to be replaced by another member or leave the cluster at their own discretion.

In this setting, we evaluate the fairness of the distributing agent (head) by using the self-aware capabilities; P7 represents the possibility of choosing an alternative distributor, and P8 is a means to modify the procedural context by migrating to a different ‘department’ of the nested enterprise.

5.4.1. P7: Minimal recognition of rights to organise

This principle says that the right of the appropriators to design the institution according to their own preferences should not be challenged by external governmental authorities.
5. Awareness and Fairness

Without this principle, all efforts of the appropriators to self-organise the resource allocation process can easily be disrupted. Someone that does not agree how the rules of the institution are set up can attain the authorities to override those local rules. As a consequence, the appropriators will not have the power to enforce their own rules. Such interventions from outside can make it very hard to achieve a sustainable organisation of the common-pool resource.

To test \( P7 \), we introduce a scheme where this principle is or is not respected and the decision of the agents might be overruled by a third party. The application we choose concerns the voting procedure.

Firstly the rule that stipulates that the head can be changed according to the vote by the member agents:

```
rule "Recognise winner head"
when
  $i: Institution($iid:id, $r:round, pr7==true)
  Declared(inst==$iid, ballot="head", round==$r, $w:winner)
  $h: Head(instId==$iid, this!=$w)
then
  retract ($h);
  insert (new Member($h));
  retract ($w);
  insert (new Head($w));
end
```

The rule means that when \( P7 \) is active and \( w \) has been declared to be the winner according to the ballot in some round \( r \), this winner becomes the new head—unless the winner is already the current head.

When \( P7 \) is not active, the members’ right to change the role of head is not recognised. The members’ vote for a new head becomes invalid and the role is appointed by an external agency. Therefore we have a separate rule that overrides the agents’ decision:

```
rule "External appointment of head"
ruleflow-group "vote"
when
  $i: Institution($iid:id, round%rate==0, pr7==false)
  $h: Head(instId==$iid)
  $mL: List(size>0) from collect(Member(instId==iid))
then
  retract($h);
  insert(new Member($h));
```

\(^7\)Note that in this scenario, the agents do not even attempt to declare the winner if \( P7 \) is not active.
5.4. Fairness in open systems

In this rule, the appointment of a new head takes place at predefined intervals, every rate time slices. An external agency collects all member agents onto a list ($mL$) and chooses a random member $m$ to be the new head. The old head is retracted from the working memory. This method to appoint a head externally is used here, alternatives include a rotation system or appointment by discretion.

5.4.2. $P8$: Nested enterprises

This principle is about nested enterprises. More complex CPRs that have to deal with different local environments and jurisdictions are organised on the basis of two or more nested layers and rule changes on the bottom layer are made in accordance with the rules on a higher level [85, p. 102]. Without this principle, it can be impossible to meet certain other principles. For example, $P2$ is concerned with the appropriation according to local conditions. As CPRs grow larger, different conditions can emerge at separate areas of the CPR, therefore separate appropriation methods are needed. To this end, we divide the cluster in three subclusters ($C_i$ with $i = 1, 2, 3$) and two layers. The top rule layer defines how the resource $P$ is divided into the three subclusters $P = P_1 + P_2 + P_3$, and in the bottom layer, local rules define how to distribute their resource $P_i$.

```
Member m = (Member) mL.get(random($mL.size()));
retract(m);
insert(new Head(m));
```

In this rule, all subclusters of an institution $i$ that have members appropriating from a common pool $p$ are collected into a list $sL$. The consequence of the rule invokes a method that defines the amount of resource that is distributed to each of the subclusters,

```
rule "Split resources for subclusters"
when
  $p$: CommonPool($l:resourceLevel)
  $i$: Institution($iid:id, pool==$p, $r:round, pr8==true)
  $sL$: List() from collect( SubInst(instId==$iid) )
then
  for(Object s : $sL){
    modify(s){
      setSubResourceLevel($l*s.getSubMemberCount()/$i.getMemberCount())
    }
  }
end
```

In this rule, the appointment of a new head takes place at predefined intervals, every rate time slices. An external agency collects all member agents onto a list ($mL$) and chooses a random member $m$ to be the new head. The old head is retracted from the working memory. This method to appoint a head externally is used here, alternatives include a rotation system or appointment by discretion.
5. Awareness and Fairness

\texttt{setSubResourceLevel()}, which is the number of members in that subcluster divided by the number of members in the whole cluster. Based on [74] where it says that equality “does not differentiate among recipients”, we chose the distribution norm for this top layer to be ‘equality according to member size’ as there is no possibility to further distinguish among the receiving cluster members.

The distribution norm on the bottom layer is chosen according to $P^i$, the resource available to the subcluster $C^i$, and again (as before for the whole cluster) the head decides on an allocation but can be ruled out if the members of $C^i$ evaluate their head to be unfair.

The method the head uses to distribute the resource is influenced by that agent’s internal distribution norm, either equity, equality or need. For equality the members are put into a random queue and allocated what they demanded in that round one after another until there is no resource left. For equity or need the members are put into a queue with all meritorious or needy members at the front of the queue, respectively. Again, the members are allocated what they demanded until there is no resource left.

5.5. Summary

After one round of experimentation, we looked back at Figure 3.1 (on page 49) to re-evaluate the preformal theory, formalism and computer model. The preformal theory used in the first iteration of the methodology (Chapter 3) made implicit assumptions on the behaviour of human actors that we have to make explicit for agents in open systems. These assumptions include the expectation of self-aware actors and a self-organisation that satisfies the actors according to what is perceived fair. The formalism is still fit for purpose, so we did not further elaborate on this aspect.

We introduced the concepts of self-awareness and fairness in social sciences and leveraged them for implementation, which will improve the computer model. To that end, we performed the first step of the second iteration of the SIC methodology, the formal characterisation of $P7$ and $P8$. These two principles serve as example rules that are applied by using self-awareness for judging procedural fairness of the allocation process. Furthermore we answered \ref{Q4}, in that we can indeed equip the agents a mechanism that enables them to evaluate the self-organisation.
6. Fair Self-Organising Resource Allocation

6.1. Introduction

In this chapter, we proceed with the second iteration of the SIC methodology and perform the steps of principled operationalisation and controlled experimentation in line with Chapter 4.

First, we describe how the agents perform their internal fairness judgement and what actions can follow from that. Then, we extend the testbed with that functionality and describe the phases that changed from the previous ruleflow and the functions that extend the agent behaviour according to role. For the controlled experimentation, we introduce a few new parameters and justify their initialisation. We describe the brute facts and the institutional facts that we will use in the experiments and introduce the measurements for data evaluation. Again, all experiments on fairness evaluation that are performed with the CUSTOM refill scheme are analysed in detail and then compared to the HIGH, MODERATE and LOW replenishment schemes. Finally, we evaluate the results obtained from both iterations of the methodology.

6.2. Fairness judgement

To provide the agents with the capability to judge the fairness of the head, we use the idea of predictive self-awareness. Before the appropriation phase, an agent retrieves information about the current resource level and gets a sample of allocations and demands of other member agents. Furthermore, the judging agent has to know the individual profiles of the agents to be allocated, meaning whether these agents have particular achievements or are in a state of need. This profile information could be gained from previous experience of social interactions, here it will be predefined. A publicly self-aware (and noncompliant) agent could choose to influence how its profile is perceived by others and so achieve a higher allocation than it would theoretically be entitled to, but we did not consider this type of intentional misbehaviour here.

The judging agent then uses the retrieved information on the agent allocations to sim-
6. Fair Self-Organising Resource Allocation

ulate internally\footnote{With ‘internal simulation’, we do not mean that the agent recursively calls the simulator, Presage2, but that the agent simulates the allocation according to its internal norm (for itself).}, how it would have allocated the resources according to its internal distribution norm and the sample demands. The outcome represents how the head should have performed in that agents’ view, and relates to the intrinsic motivation of that agent, that is the satisfaction or propensity to change the head or leave the cluster it is member of.

According to [70] and [124], procedural justice is fostered when consistent over time. Therefore an agent’s satisfaction will increase for fair head behaviour and decrease otherwise. In case the head judgement repeatedly yields a bad outcome, the judging agent has different options when the satisfaction falls below a certain threshold. These options are ‘no response’ (agents are self-aware but cannot respond), ‘vote for a new head’ (change the agent that exhibited unfair behaviour), ‘leave the cluster’ (leave the organisation), and ‘create nested subclusters for local reorganisation’ (leave the department but stay in organisation). If the agents experience a positive effect when changing procedures (either procedural components or the whole context, i.e. they are able to respond), their satisfaction will increase.

6.3. Principled operationalisation

To extend the testbed with the aforementioned functionality, we have to integrate the fairness judgement procedure into the ruleflow and the self-awareness component into the agent behaviour. This provides functionality to the agents for a qualitative evaluation of the system.

6.3.1. Ruleflow

The new ruleflow for the testbed is shown in Figure 6.1, the syntax of the diagram is the same as before, all added rules are in displayed in red. This control loop is also given in algorithmic form, in Appendix D.13. The changes only affect five of the nine ruleflow groups, which we will describe in turn.

- \textbf{INIT:} If \textit{P8} is used, three subclusters \( C^i \) are created in round 0 and initialised with \( C^1 = \{ \text{member} \in C \} \), \( C^2 = \{ \} \) and \( C^3 = \{ \} \). In every round, the monitor splits the resource into three parts \( P^i \) after the pool has been refilled, one part for each subcluster according to its member size \( |M^i|, i \in \{1,2,3\} \).

- \textbf{CFV:} In this ruleflow group, no vote is called for a new \textit{raMethod} (which is auto-
6.3. Principled operationalisation

Figure 6.1.: Rule flow in the resource allocation process with fairness judgement
matically set to queue at the start and \texttt{samplingrate=500}, see Table B.3). Instead the head will initiate a voting procedure to reassign the role of head.

- **VOTE**: With $P7$ in use, the members vote for a new head, the votes are counted and the winner is declared and updated. If $P7$ is not active, the winner update is overruled and a head appointed by an external authority at predefined intervals (according to \texttt{samplingrateHead}). The fact whether a new head is appointed or not, has an effect on the member satisfaction.

- **APPROPRIATE**: The \texttt{feMethod} requires that members collect demand and allocation information on a specified amount of member agents (\texttt{judgeSize}). The members then judge the fairness of the head on the basis of this sample. If the head allocation deviates by more than \texttt{judgeTolerance} from the allocation that member simulates internally, the head is considered unfair and the member’s satisfaction is decreased.

- **EXCLUDE**: In this ruleflow group, the members decide, depending on their current satisfaction, whether they would like to leave the cluster (\texttt{satisfaction<leaveSat}) and become a non-member or not. If $P8$ is in use, the members have the possibility to join one of two other subclusters instead of the possibility of leaving.

### 6.3.2. Agent behaviour

In the extended testbed, an agent is furthermore interested in a fair allocation of resources, where ‘fair’ means ‘according to an agents internal distribution norm’. An agent will therefore evaluate the head’s performance by sampling relevant information from the agent population. Judging from these samples, the agent decides whether its norm coincides with the head’s. The agent will express the perceived fairness through a satisfaction value which increases for a positive and decreases for a negative judgement. This judgement will also influence the vote that agent casts when the vote for a head is called. If an agent is constantly dissatisfied with the fairness in the cluster it has the option to leave the cluster completely, switch to a different subcluster or must stay, depending on the testbed setup.

With the extension of the testbed, the different agent roles can utilise additional methods and rules. These are explained here, the implementation details can be found in Appendix D.

We start with the additional member methods (again prefixed by a reference to the code):

- **D.1 vote()**: To cast a head vote, a member checks whether it has judged the current head as unfair in some previous round. If that is not the case, the head is voted for again (line 6). Otherwise, the member makes a list of all candidates, which
6.3. Principled operationalisation

Algorithm 2: vote() by member $m$

$h \leftarrow$ current head;
$\exists$ call for vote on $\text{head}$;
$L \leftarrow m$’s list of heads considered unfair; // list of dropped heads, see judge()

if $h \notin L$ then
  $m$ votes for $h$;
else if $\text{len}(L) \neq |M|$ then
  $m$ votes for random member $a \notin L$;
else
  $m$ votes $\text{null}$;

are the members of the same (sub)institution minus all members that are kept on
the list of $\text{heads}$ that previously dissatisfied the agent (droppedHeads, lines 8–13).
The agent then votes for a random member from this list (line 17), unless the list
is empty, in which case the returned vote is $\text{null}$ (line 15). This procedure is also
outlined in Algorithm 2.

D.2 statevaluation(): The satisfaction evaluation is split into two parts. The
first part concerns the change of satisfaction after a new head is appointed or
elected (initiated by HeadChange fact), and the second part concerns the change of
satisfaction for unfair head behaviour (see judge() or Appendix D.13). When a
head is appointed externally, a member will give this new head some credit and
set its satisfaction to initialSat, the value of initial satisfaction (line 10). When
a head is elected by the cluster, the satisfaction can be set to two values. Either
to initialSat (line 24) or to 1.0 if the new head coincides with the cast vote of
the member (lines 25–27).

D.3 sample(): Creating a selection of member agents defines the first step in
performing a fairness judgement on the head. Therefore, the agents collect a list of
members (line 7) and select a certain number at random, according to judgeSize
(lines 18–20).

D.4 judge(): To judge the current head, information about the demands and allo-
cations from the sampled members is retrieved from the working memory (lines 8,
9) and serves as input for judgeHead(), executed by the judging member agent
(line 13). There are two parts to the method judgeHead() which is the main com-
ponent of self-awareness. The first part is the agent’s internal allocation simulation
using the sample member demands. The agent calculates how many members can
obtain an allocation, given judgeSize and the current resource level (lines 26–30). Subsequently, the allocation is simulated according to the agent’s distribution
Algorithm 3: judge() by member m with distribution norm equity

Sample ← set of sampled member demands;
m counts meritorious and needy members ∈ Sample: mS and nS;
m counts meritorious and needy members ∈ Sample that got allocated: mA and nA;
possAll ← ⌊resource level standard request · [Sample] / |M|⌋; // possible allocations, proportional to sample

// Simulated allocations:
if possAll > mS then // more allocations possible than meritorious demanded
    m allocates all mS meritorious members;
    m allocates rest of possAll to needy members;
else
    m allocates possAll meritorious members;
    m allocates nothing to needy members;

// Fairness evaluation:
if |m’s allocations to meritorious − mA| > judgeTolerance then
    satisfaction(m) ← satisfaction(m) - decreaseFactor · satisfaction(m);
else if |m’s allocations to needy − nA| > judgeTolerance then
    satisfaction(m) ← satisfaction(m) - decreaseFactor · satisfaction(m);
else
    satisfaction(m) ← satisfaction(m) + increaseFactor · (1 − satisfaction(m)); // head considered fair

norm, either regularNorm or crisisNorm (lines 20–24). If this norm is equity, then
the agent computes how many meritorious members can be allocated their demand,
any remainder goes to the needy members (lines 54–62). For need this is the reverse
(lines 71–79), and for equality the two profiles are allocated to equal parts (lines 64–69).
The second part of judgeHead() evaluates the deviation of actual allocations by the head
to the simulated allocations by the agent. If the result deviates by more than judgeTolerance
(lines 82–93), the head is judged unfair and added to the droppedHeads list if not there yet. Simultaneously,
the member satisfaction decreases by the decreaseFactor (lines 83, 89). Given a positive judgement
on the head’s fairness, the satisfaction of the judging member is increased by
(1-satisfaction)·increaseFactor. An example of judge() by a member using equity as distribution norm
is given in Algorithm 3.

○ D.5 leave(): A member can leave the cluster if it is dissatisfied due to multiple unfair allocations by the head. When its satisfaction is lower than leaveSat (line 7), the member fact is retracted and a non-member fact inserted into the working memory with a satisfaction of 0 (line 10). The head will not leave the cluster.
6.3. Principled operationalisation

- **D.6 change()**: If P8 is in use, three subclusters are created\(^2\), so that instead of leaving the cluster, the member agents can choose a different subcluster if dissatisfied (line 10). Given that there is a different head in the new subcluster, the satisfaction gets set to initialSat (line 14) upon migration.

The field methods of head and gatekeeper are extended as follows:

- **D.7 declare()**: After the votes are counted (line 6), the head has to declare the outcome according to the following wdMethod. Firstly, the head checks whether more than half the members voted for null (no head), in which case the institution is retracted from the WM (lines 10, 11) which is the end for this (sub)cluster. If this is not the case and more than half the members vote for the current head, this agent is declared the winner (line 14, 15, 31). If this does not hold either, the agent with the highest vote (even if below 50%) is declared the winner (lines 20–26, 31), in case of a tie one agent is selected at random (line 29).

- **D.8 update()**: If P7 is used and a winner different from the current head has been declared in the same round (lines 5–7), then the current head resigns (lines 10, 11) and the agent occupying the head role is updated to the winner (lines 12, 13). A HeadChange fact is inserted into the WM which triggers the member satisfaction evaluation.

- **D.9 allocate()**: The allocation method is now executed according to three distribution norms. Which one is used depends on the head’s internal judgment and the pooled resource (minus any monitoring cost, lines 3, 9–12). If the norm is equity, the member demands are shuffled and one after another added to an empty queue, a meritorious member at the front, a needy member at the back (lines 15–23). For need being the norm, adding the member to the queue takes place the opposite way, a needy member is added to the front and a meritorious one to the back (lines 31–39). For equality, the demands are shuffled and added to the queue in the new order (lines 25–29). Subsequently, the members are allocated their demands in order of the queue until there is no more resource left (lines 44–52).

- **D.10 split()**: For implementation purposes we slightly deviate from the order of pool refill (depending on the refill scheme and maxLevel) and then splitting the resource over the subclusters according to membership. Instead, the gatekeeper of C first lowers the maximum resource level of the three subclusters \(C_i\) and then the refill rule fires for each of them (lines 5, 12). In round 0, maxLevel is initialised in terms of the membership (2*standardrequest*agents), so both methods yield the same result.

\(^2\)For implementation purposes, instead of 1 cluster having 3 subclusters, we created 3 clusters \(C_i\) at start with the same rules in the institution, i.e. \(I^1 = I^2 = I^3\) which state that there will be 3 separate heads, unless a subcluster is empty, but only one gatekeeper for the whole cluster.
There is one method which is not executed by a member or the environment $\epsilon$ of the cluster, but an external agency not in $C$. This method is:

- **D.11 appoint()**: Without $P7$, an external agency appoints a new head for a cluster or subcluster at fixed intervals ($\text{samplingrateHead}$, line 4). Therefore, a list of agents is created with members other than the head (line 6) and one of them chosen at random (line 11), unless empty. This chosen agent is appointed the role of head, and the current head becomes a member.

### 6.4. Controlled experimentation

We now prepare the extended testbed for experimentation. Again, there is a subset of parameters that we manipulate for the individual experimental runs, which we will explain in the following. The complete list of additional field values and existing field values that have changed from their previous default is given in Appendix D.12.

#### 6.4.1. Parameters

A few of the new parameters we introduced for the extended version of testbed are explained here.

- **judgeSize=10, judgeTolerance=2**—We cannot rely on the fact that every sample drawn from the agent population represents $1-\text{noRequestPercentage}$ demanding and profilePerc meritorious members, and allocated members depending on the refill rate $r$, demand, profile and sample size. Depending on the distribution norm used, these allocations occur with a certain probability (the member’s profile is referred to in brackets), see Table 6.1. Recall that $r = 1$ is enough resource to allocate all 100 agents. For judging a head’s allocation procedure, we introduce a tolerance of 2, so that the fluctuation in sampling (size 10) has less impact. The probability of getting the sample ‘right’ with respect to sample size and tolerance are 91% for meritorious/needy\(^3\), 99% for demand/no demand, and at least 91% for allocation/no allocation, depending on the refill rate (worst for $r = 0.5$, up to 100% for $r \geq \text{demand}$). Rawls calls this an instance of imperfect procedural justice [95, p. 86], even though an evaluation has been carried out carefully and according to the rules it may still yield the wrong result.

- **initialSat=0.8, leaveSat=0.4, increaseFactor=0.1, decreasefactor=0.25**—The initial satisfaction of 0.8 gives the members the possibility to increase their sat-

---

\(^3\)Hypergeometric distribution: $\mathbb{P}(3 \leq X \leq 7) = \left(\binom{50}{5}^2 + 2 \cdot \binom{50}{4} \binom{50}{6} + 2 \cdot \binom{50}{7} \binom{50}{3} \right) / \binom{100}{10} = 0.9084$
### 6.4. Controlled experimentation

Table 6.1.: Probability of allocation by distribution norm with $P(\text{agent demanded}) = d$, $P(\text{agent is meritorious}) = m$, refill rate $= r$

<table>
<thead>
<tr>
<th>norm</th>
<th>restrictions</th>
<th>probability of allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>equity</td>
<td>$dm \leq r &lt; d$ (needy)</td>
<td>$(r - dm) / (d - dm)$</td>
</tr>
<tr>
<td></td>
<td>$r &lt; dm$ (merit.)</td>
<td>$r/dm$</td>
</tr>
<tr>
<td></td>
<td>$r &lt; dm$ (needy)</td>
<td>0.0</td>
</tr>
</tbody>
</table>

| equality | $r \geq d$ | 1.0 |
|          | $r < d$ (merit.) | $dmr$ |
|          | $r < d$ (needy) | $d(1 - m)r$ |

| need     | $r \geq d$ | 1.0 |
|          | $d(1 - m) \leq r < d$ (merit.) | $(r - d(1 - m)) / (d - d(1 - m))$ |
|          | $r < d(1 - m)$ (merit.) | 0.0 |
|          | $r < d(1 - m)$ (needy) | $r/d(1 - m)$ |

If they encounter fair head behaviour ($sat_{\text{new}} = sat + 0.1(1 - sat)$) or a winner they voted for ($sat_{\text{new}} = 1$). The chosen threshold for leaving means that an agent leaves the (sub)cluster after being dissatisfied due to unfair head behaviour ($sat_{\text{new}} = sat - 0.25sat$) three times in a row (starting at $sat = 0.8$), or four times with two fair judgements after each dissatisfaction, etc.

- **sampling rate Head = $r$**—The sampling rates chosen match different possible events. The high rate of $r = 12$ represents the upper bound of heads that are voted for when $P7$ (but not $P8$) is used (resulting in 42 appointments, $\max(\text{voted}) = 41.9$) and $r = 15$ represents the lower bound (resulting in 34 appointments, $\min(\text{voted}) = 33$). The choice of $r = 50$ matches the change of refill rate every 50 time slices. Some test runs were performed with all agents having equality as preferred distribution norm, which led to 7 head changes when voting. The rate $r = 75$ results in the same amount of appointments, $r = 25$ is in the sampling rate sequence for good measure.

- **institutions=3**—We initialise $P8$ with three subclusters (for implementation purposes they are three institutions with the same rules), one for each distribution norm *equity*, *equality* and *need*, and no separate subcluster for ‘indecisive’ agents.
6. Fair Self-Organising Resource Allocation

6.4.2. Experimentation with brute facts

Again, we are using different schemes for the resource replenishment, these are custom, high, moderate and low. The schemes are the same as described in Section 4.3.1 with the only difference that for high, moderate and low the initial period of refill at factor 1.0 is omitted. The total amount of refill does not change for the case with three subclusters. Also, this time no unintentional violations will occur.

Furthermore, we only have two types of agent population, one is capitalist and the other socialist. To make the influence of fair or unfair behaviour apparent, both populations consist of fully compliant members, no non-members are present. The agents’ profiles in both cases are 50% meritorious and 50% needy$^4$.

In the capitalist population (as illustrated in Table 5.1), for 50% of the agents equity is their distribution norm of choice and 75% of those will choose equality in times of scarce resource (these agents are called ‘equityChange’) and the remaining ones will not change their norm (‘equityStay’). 33% of agents in the capitalist population choose equality as their regular norm, again 75% will change when the resource gets scarce, this time to need (‘equityChange’), the others stay with equity (‘equityChange’). The remaining 17% of the capitalist agents have need as their distribution norm (‘need’) from which they will not deviate.

For the socialist population, the emphasis is on need which accounts for 50% of the population. The equity norm is only chosen by 17% of agents whereof 50% change to equality in times of scarce resource. 33% of socialist agents choose equality as distribution norm and again, 50% will deviate to need when the resource gets scarce.

6.4.3. Experimentation with institutional facts

As the population in our experiments consists of compliant members, we use $P_1$, $P_2$ and $P_3$ in all runs, and $P_4$, $P_5$ and $P_6$ in none of them$^5$. $P_7$ and $P_8$ will be used on and off for experimentation. As the assignment of the gatekeeper and monitor is not the focus here, their tasks (such as the splitting or resource over three subclusters, for example) will be performed by a random member agent.

$P_7$ ensures that the cluster’s right to organise is recognised by the authorities. This principle is challenged by external authorities prescribing the head of an institution every $r$ time slices, where $rater \in \{12, 15, 25, 50, 75\}$. If $P_7$ is used, the agents are allowed to vote for the head themselves.

$^4$These profiles are needed for both allocation and fairness judgement according to an agent’s distribution norm.

$^5$This results in a collection of rules that are not fired anymore, which are displayed in grey in Figure 6.1.
6.5. Results

P8 concerns the nesting of clusters. With P8 used, the members choose a different subcluster when their satisfaction falls below the threshold of 0.4. Two cases are tested with P8 not used. One where the agents do not leave the cluster at all ($\text{leaveSat}=0.0$) and one where the agents leave the cluster and become non-members when their satisfaction falls below 0.4.

The aim of these sets of experiments, with brute and institutional facts, is to find out whether the use of P7 and P8 result in a higher member satisfaction and lower leaving rate, the main benefits of procedural justice.

6.5. Results

We now describe the experiments we conducted with the extended testbed and evaluate the obtained data. The experimentation was split into four parts according to the refill schemes. Each of the experimental runs was performed over 100 trials and the results averaged. Several measurements (per remaining cluster unless otherwise stated) provide information about the performance of the ‘self-aware open system’:

- **Satisfaction**—This is the measure for the satisfaction per member and time slice in a cluster (or subclusters), i.e. $1/500 \sum_t (\sum_{a \in M_t} \text{sat}(a) / |M_t|)$. For the case when agents can leave the cluster, a measure per agent and time slice that includes non-members is provided as well, i.e. $1/500 \sum_t (\sum_{a \in A_t} \text{sat}(a) / |A_t|)$. The satisfaction is averaged over all agents, including the ones from clusters that did not endure until $t = 500$.

- **Head Changes**—With this measure, we collect all head changes that occurred by the end of a run in a cluster or in all subclusters together, i.e. $\mathbb{1}.\text{size}()$, where $\mathbb{1}: \text{List()}$ from $\text{collect( HeadChange() )}$.

- **Remaining Agents**—This measure counts the agents that have not yet left the cluster, i.e. $|M_{500}|$.

- **End Distribution**—This is the distribution of remaining agents according to preferred distribution norm, i.e. $|\{a \in M_t : \text{norm}(a) = x\}| / |M_t|$ at $t = 500$ for the group of agents with norm $x \in \{\text{equityStay, equityChange, equalityStay, equalityChange, need}\}$.

- **Purity**—This measure has two components. Firstly, it retrieves the subcluster that has the highest share of agents with some norm $x$ in comparison to the other subclusters, i.e. $c = \arg \max_{i \in \{1,2,3\}} |\{a \in M^t_i : \text{norm}(a) = x\}|$, where $M^t_i$ denotes the members of subcluster $C^t$ at time $t$, $i \in \{1,2,3\}$. Secondly, the ‘distribution’ of this share of agents within the subcluster $C^c$ is calculated, which is
6. Fair Self-Organising Resource Allocation

\[ \frac{|\{a \in M_t^C : \text{norm}(a) = x\}|}{|M_t^C|} \]. This value is then multiplied by the ‘concentration’ of members with norm \( x \) in \( C^c \), meaning how big the share of agents with norm \( x \) in this subcluster is in comparison to all agents with norm \( x \) in the whole cluster, i.e. \( \frac{|\{a \in M_t^C : \text{norm}(a) = x\}|}{|\{a \in M_t : \text{norm}(a) = x\}|} \).

The purpose of the first three measurements is obvious. The fourth, end distribution, is used when agents can leave the cluster and allows us to judge whether a group of agent that prefers a certain norm performs better or worse in comparison to the start distribution of agents at \( t = 1 \). The last measurement, purity, is only used when members can change into a different subcluster. It states how well agents of a certain norm group do within a single subcluster and how many other agents are in that subcluster. For example, purity= 1 means that all agents of norm \( x \) are in the same subcluster and they are the only members in that subcluster. Compared to the initial purity value \( (t = 1) \), we can judge whether the agents spread out evenly over the subclusters or whether they prefer to gather with agents having the same norm.

We now present the results of the first refill scheme.

6.5.1. Fairness evaluation for the CUSTOM refill scheme

Figure 6.2 shows the satisfaction measure (a), how many heads changed (b) and how many agents remained in the institution (for the leave case) (c). For (a) and (b) the variations on \( P7 \) are presented along the x-axis and the variations on \( P8 \) are expressed with several graphs. All runs are performed under the CUSTOM refill scheme.

The runs where only Principles \( P1, P2 \) and \( P3 \) were chosen are shown in red triangles (minus the value on the very left) as no leave with rate \( r \in \{12, 15, 25, 50, 75\} \). In these runs \( P7 \) and \( P8 \) are not used, i.e. the agents can express their satisfaction, but they cannot do anything about their situation (which is either leave, go to a different subcluster or vote for a new head).

We can see in Figure 6.2(a) that adding \( P7 \) (vote, leftmost column) leads to an increase in satisfaction between 5% and 11%. The addition of \( P8 \) (change, green) leads to a satisfaction increase by at least 5% for \( r = 12 \) up to at most 10% for \( r = 75 \). The satisfaction rises with \( r \), as the agents have more time to search for an appropriate cluster before a new head is appointed.

The case where \( P8 \) is not used but the agents can leave gives them a limited possibility to express their satisfaction. For the members in this category (leave (rem), blue), the satisfaction is highest compared to other cases (without \( P7 \), but when we take into account that between 70 and 80 agents leave the cluster over the 500 rounds, see (c), the satisfaction averaged over all agents (leave (all), purple) decreases considerably to values
Figure 6.2.: Satisfaction, head change and remaining agents for custom refill scheme

near the leaveSat threshold (grey line). A higher satisfaction for higher \( r \) is caused by the low number of members. As all members are fully allocated, no unfair behaviour is detected and the member satisfaction increases.

The difference between the satisfaction of members and all agents is not as big when \( P7 \) is used, as around 60 members stay in the cluster. The highest satisfaction value of all runs, 0.95, is reached when both principles are used (vote, change).

Figure 6.2(b) shows the changes of head, averaged over 100 trials. Without \( P7 \) the changes are predefined by the rate and are considerably higher for \( P8 \) due to multiple clusters that are being populated after some time. For the case with \( P7 \) however, the possibility to change cluster leads to a lower head change than for the two other cases.
This is particularly important if the procedure of changing the head comes at a certain cost (which we did not implement here), like the monitoring procedure in the first version of the testbed.

The socialist and capitalist runs notably differ for leave only. The satisfaction for all agents is between 8% and 13% higher for capitalists than for socialists, note that with \( P7 \) this is reversed and the difference is smaller (+5%). The absolute values of remaining agents per distribution norm in these runs are shown in Figure 6.3.

The voting case has more agents remaining than when a head is appointed externally. Moreover, the socialist population structure is better preserved in that case. This is due to the fact that 50% (or 67% when resources are low, which is also the time when most
conflicts arise) of agents with ‘need’ as distribution norm are enough mass to successfully call for a new vote as soon as a head’s norm differs, whereas when the head is appointed by an external agency, they just have to stick with it for $r$ time slices or leave. The same holds for the capitalist population, although the effect is not as strong, as for a low resource, merely 13% of the initial population have ‘equity’ as preferred distribution norm and ‘equality’ has a share of 46%.

The percentage distribution of remaining agents can be recognised better using the end distribution measure, see Figure 6.4. It shows well that agents do not leave uniformly in which case all the lines would be flat. Again, for the socialist case, the distribution for vote follows the opposite trend than for rate (apart from equalityStay) but for the
capitalist case the numbers merely differ in magnitude. The groups that profit\(^6\) in the socialist case (a) are need and equality\text{Change}, whereas in the capitalist case (b) they are equality\text{Stay} and equity\text{Change}.

The last measure we discuss is purity (change). Figure 6.5 shows that the agents do not just spread out equally over the subclusters (this theoretical case is represented at the far left), but organise group wise. As they are more likely to migrate when the resource is low, they tend to group ‘need & equality\text{Change}’, ‘equality\text{Stay} & equity\text{Change}’ and ‘equity\text{Stay}’. When the head is appointed externally, the grouping works better for higher rates as there is more time for exploration.

\(^6\)We say a group profits wherever the graph goes up from the perspective of ‘initially’.
6.5. Results

Figure 6.6.: Subcluster change of socialist population (Pr7 and Pr8, 20 head changes)

<table>
<thead>
<tr>
<th>Subcluster</th>
<th>equityStay1</th>
<th>equityChange1</th>
<th>equalityStay1</th>
<th>equalityChange1</th>
<th>need1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subcluster 1</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>14</td>
</tr>
<tr>
<td>Subcluster 2</td>
<td>9</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>1</td>
</tr>
<tr>
<td>Subcluster 3</td>
<td>·</td>
<td>8</td>
<td>18</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>total</td>
<td>9</td>
<td>8</td>
<td>18</td>
<td>14</td>
<td>51</td>
</tr>
</tbody>
</table>

We now show the subcluster migration for two singular trials. Both have a socialist population structure and for the first trial, Pr7 was used, see Figure 6.6. The refill rates during certain time frames are shown in the rectangles, above those are strokes of different length indicating how many head changes occurred at what time. All 20 changes occur during periods of low resource refill. This is mainly due to the fact that at times of high refill all members that demand resources (90%) can be fully allocated. Furthermore, for times of medium refill rates only 22% of agents that demanded resource cannot be allocated, which, if a representative sample of 10 members is drawn (see Section 6.4.1), might be seen as fluctuation and is to 97% covered by the tolerance of 2.

How the agents are distributed over the three subclusters by the end of this trial can be seen in Table 6.2. These figures indicate that the predominant distribution norm for the first subcluster is ‘need’, for the second it is ‘equity’, and ‘equality’ for the third. From that, we conclude that the self-organisation into different subclusters according to the agents’ individual fairness perception is working well.

The second trial we present, see Figure 6.7, uses again a socialist population structure.
6. Fair Self-Organising Resource Allocation

Figure 6.7.: Subcluster change of socialist population (rate = 15 and Pr8, 82 head changes)

Table 6.3.: End distribution of socialist population (rate = 15 and Pr8)

<table>
<thead>
<tr>
<th>subcluster 1</th>
<th>equityStay</th>
<th>equityChange</th>
<th>equalityStay</th>
<th>equalityChange</th>
<th>need</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>subcluster 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>subcluster 3</td>
<td>8</td>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>8</td>
<td>5</td>
<td>11</td>
<td>22</td>
<td>54</td>
</tr>
</tbody>
</table>

and P8, but the head is appointed at rate r = 15, i.e. P7 is not used. Other than in the first trial, we observe a high number of subcluster changes for agents preferring the need norm. This is due to the fact that in this trial the agents cannot express their dissatisfaction other than through leaving the subcluster, whereas in the previous case voting for a new head is the first resolution method.

Also the other groups of agents change subclusters more often than in the previous trial, and their distribution at t = 500 within the subclusters is shown in Table 6.3. This time, both subclusters 1 and 2 have ‘need’ as their predominant distribution norm, and the third subcluster is shared by ‘equity’ and ‘equality’.

Two more trials similar to the above can be found in Appendix E, this time the population structure is capitalist but the use of principles is the same. Again, we can see (Figure E.1 and Table E.1) that P7 enables the agents to organise themselves into subclusters according to preference. Without this principle (Figure E.2 and Table E.2) the agents migrate much more frequently. Furthermore, the groups of agents with the same
distribution norm are even more defragmented than in the corresponding socialist trial with no $P_7$, and no clear classification into predominant norms is possible.

We briefly summarise the outcomes of these first sets of experiments:

- The satisfaction is best when Principles $P_7$ and $P_8$ are in use. When $P_7$ is not used, the satisfaction rises with $r$. When $P_8$ is not in use the satisfaction is lowest on average for agents that have the possibility to leave, but highest for the few agents that remain in the cluster, see Figure 6.2.

- The head changes we observe when $P_7$ is in use are about the same as the changes that occur for external appointments with $r = 12$ or $r = 15$. When $P_8$ is added, the external appointments increase significantly, whereas there are even less heads changed by vote for than without $P_8$.

- The numbers of remaining agents are considerably higher with $P_7$ than when a new head is appointed externally. Without $P_7$, the small membership results in all members being allocated resources in every round, therefore the satisfaction only decreases shortly when a head change occurs. This effect becomes apparent for high $r$ when the accumulated satisfaction increases.

- The end distribution of remaining agents shows that agents do not leave the cluster uniformly. When $P_7$ is used, the agents that remain in the cluster more likely are the agents in line with the current population structure (socialist or capitalist). When $P_7$ is not used, this trend is stronger for the capitalist, but reversed for the socialist population, see Figure 6.4.

- When $P_8$ is used, the agents can migrate from one subclusters to another. The purity measure shows that they are indeed able to form subclusters of like-minded members, which works best when the head changes are low and the rate of migration decreases (Figures 6.6 and 6.7).

### 6.5.2. Fairness evaluation for HIGH, MODERATE and LOW refill schemes

This section evaluates the experimental runs with the refill schemes HIGH, MODERATE and LOW, where the refill rate is constant over the whole 500 rounds. Again, there are two population structures (socialist and capitalist), all agents comply with the rules of appropriation, and Principles $P_7$ and $P_8$ have been tested with different parameters.

Figures 6.8 and 6.9 show the satisfaction, head changes and remaining members of the sets of experiments with a high and moderate rate of replenishment respectively. For the high case, there are no differences due to the variations on $P_8$. When there are enough resources so that every agent gets allocated in every round, no unfair behaviour is
detected and all members stay in the cluster. As none of the agents becomes dissatisfied, no head changes occur for the case when $P7$ is used and the satisfaction is nearly 1.0. When the head is appointed by an external agency, the agents satisfaction decreases at each time of appointment and re-increases in subsequent time slices. This has as effect that for frequent head changes the satisfaction is lower on average than for less frequent ones. When the refill rate is moderate, the satisfaction differs a little according to the chosen instance of $P8$, is slightly higher for the capitalist than for the socialist population, and the satisfaction is higher when $P7$ is used as when it is not. This time, between 7.2 and 12.1 new heads are voted for, including the case when $P8$ is active and three subclusters can vote for their separate heads. When $P7$ is not used, the head changes with $P8$ increase by at least 61% in comparison to the cases where members
6.5. Results

Figure 6.9.: Satisfaction, head change and remaining agents for moderate refill scheme

cannot change to a different subcluster. Furthermore, only up to 2.3% of agents leave the cluster when it is possible.

The experimental runs with a low replenishment rate have a very different outcome in satisfaction, head changes and remaining agents, see Figure 6.10. For the first time, not all the clusters endure until the end. When $P7$ is used and agents can migrate, the number of head changes amounts to a value between the numbers for appointment with rate 12 and 15. This means for the satisfaction, that it is only up to 5% higher for the vote case than in comparison to $r \in \{12, 15\}$ and the satisfactions at $r = 50$ and $r = 75$ surpass the satisfaction from the vote case.

When $P8$ is not used, the difference in head changes between voting and appointment is
substantial. The highest value, 425, is from the case where leaving is allowed, $P7$ is used and the population profile is socialist. In only 13 of the 100 trials, the cluster endures until $t = 500$ with an average of 64 remaining agents. In all other cases, more than half the members vote **null** at some $t < 500$ which leads the end of the institution and subsequent abortion of the trial. This is also reflected in the satisfaction value which is the average per agent and time slice, including the runs that ended prematurely. The satisfaction for all agents in this case is 0.39, which is lower than `leaveSat`.

For a capitalist population, 56 trials endure until the end with on average 352 head changes, which results in more remaining agents (57.4) and a 7% higher overall satisfaction. When $P7$ is not used, a predefined number of head changes occur, which
lead to a greater overall dissatisfaction (as low as 0.2) and even fewer remaining agents (19.2). However, this time, and in all other remaining trials with the low refill scheme, all clusters endure until the end as there are no voting conflicts.

The end distribution of the two population structures can be seen in Figure 6.11. In comparison to the initial distribution, more agents that prefer the need norm (for a low refill these are equalityChange and need) are present in the socialist population when $P7$ is used. For the capitalist population, the profiting agents have equality as their preferred norm (equityChange and equalityStay). When the head is appointed externally, these distribution differences increase for the capitalist population, but for the socialist one, ‘need’ loosens out and ‘equity’ profits this time. As no agents or almost none leave when the replenishment rate is high or medium, the end distribution in these cases is (quasi) the same as the initial one, see Figures E.3 and E.4 in the appendix.

When the agents are not able to leave the cluster, the capitalist population votes for 245 new heads if $P8$ is used, the socialist population vote for 197, see Figure 6.10(b). In both cases, the satisfaction reaches 0.74. Without $P8$ and a fixed amount of head changes, the satisfaction drops down by 13% to 25%, as the agents have no means to express their preferences. Contrary to the other refill schemes, the satisfaction now decreases with decreasing head changes ($r$ increasing), as there are fewer resources resulting in a higher probability of perceived unfairness (average satisfaction below $\text{initialSat}$).

The next measure we discuss is the purity for runs when $P8$ is used. For the case with high refill rates, no member changed subcluster and the purity does not deviate from the initial value. When the refill rate is moderate, only very few agents migrate and the purity remains nearly the same. These two cases are shown in Figure E.5 and Figure E.6 in the appendix.

A low replenishment rate, causes the agents to change subcluster frequently until they find themselves with likeminded agents in the cluster. Figure 6.12 shows that the purity increases substantially for all groups of agents that prefer the same distribution norm.

The purity in these three figures is also reflected in the figures and tables hereafter that show the migration and end distribution of members in four individual trials.

The first two trials in Figure 6.13 use $P7$ and $P8$ with a socialist population and a high (a) or moderate (b) refill scheme. As mentioned before, no subcluster changes happen when the replenishment rate is high and only a few when the rate is moderate, mainly by agents preferring equity as sole distribution norm (equalityStay). The appendix contains Table E.3 with the final distribution of agents over the subclusters in both trials.

The third trial, again with $P7$, $P8$ and a socialist population, is shown in Figure 6.14. This time, many more agents change subcluster in the first half of the trial. In the second half, they manage to organise themselves into sufficiently similar clusters and vote for
appropriate heads (370 head changes in total) so that only a few more changes follow towards the end of the trial. So overall, the rate of subcluster change and head change seem to be inversely proportional for this trial.

In the end, the agents reach the following distribution over the three subclusters, see Table 6.4. The prevailing norm in the first subcluster is ‘need’ and in the third ‘equity’. The agents in the second one come from various groups, but all of equityStay and the remaining agents of ‘equality’ are included.

In order to compare the case when the agents are able to vote for their head to the case when the head is appointed externally, we include Figure 6.15. Again the refill rate is low and the end distribution is in the appendix, Table E.4. In this trial, the head
changes ‘only’ 102 times, but all of them are critical to the cluster as the appointment is made at random. This has as effect that even the majority group in a subcluster has to migrate (see for example need1 → need2 in (a)). Still, the agents manage to organise into subclusters intermittently. When a new head is appointed the agents can be forced to change again and the stability of the subclusters does not endure.

6.5.3. Evaluation

The data obtained from the four refill schemes allows us to draw several conclusions on the benefits of Principles \( P7 \) and \( P8 \).
There are two variations that do not use any of these two principles, one is ‘no leave’, the other ‘leave’ with different rates of head appointment. In all experiments, the satisfaction of the ‘no leave’ case is higher than for the ‘leave’ case, that is because all agents remain in the cluster until the end, adding to the total satisfaction. In the other case, however, agents leave the cluster as soon as they are dissatisfied and do not contribute to the satisfaction any longer. Which agents they are, depends on the population structure and the agents’ individual preference of distribution norm. On the one hand, this leads to a much increased satisfaction for the remaining members, but on the other hand, the overall satisfaction becomes very low as the resource replenishment is low. If we merely care about the members that remain in the cluster, however small that number might be, then this should be the preferred strategy. If membership is important, more functionality (here principles) is needed to avoid the relatively low satisfaction of the ‘no leave’ case.

Adding $P7$, i.e. enabling the agents to choose their own head, surpasses the satisfaction of all other agents but in a few cases. For the custom refill, the satisfaction for remaining members is higher by at most 2.7% for the rates 50, and 75, and when the refill scheme
6.5. Results

Figure 6.14.: Subcluster change of socialist population (Pr7 and Pr8, 370 head changes), low refill scheme

is moderate and the appointment rate is 75, they are about the same for remaining and all members. For the low refill scheme, the satisfaction increases between 15% and 31% for all appointment rates when we consider the remaining agents only. Although more agents remain in clusters where agents can vote for their head as opposed to have it appointed, in many cases the members of these clusters cannot find a head that suits them, so when they run out of possibilities, the clusters have to be terminated prematurely. According to the rules, the agents move a head onto their droppedHeads lists as soon as they are dissatisfied with its performance once, and there is no possibility for a head to get off these individual lists. We contend that a more graduated scheme where, if a head performs well for several time slices after it misbehaved (even if the behaviour is misjudged), the head can be taken off the list again, can solve this issue. However, taking into account that not all clusters (and corresponding members) make it until round 500, the satisfaction still reaches a very high value compared to when a head is appointed and all clusters endure.

$P8$ allows the agents to create subclusters within their cluster, each with their own head. They will move to a different subcluster after several instances of dissatisfaction in the current subcluster. With this principle, the agents can organise themselves into more homogenous groups (see purity measurement) and within that group an appointed head is more likely to satisfy the members, and it is easier for minority groups to have their preferred head elected, who would otherwise be out ruled. The satisfaction with $P8$ is highest in comparison to all other runs of the ‘no leave’ or ‘leave’ cases.

$^7$Effectively, this is a graduated scheme.
When we compare the CUSTOM scheme with the other refill schemes, we can see that the agents organise during low refill periods and the agent distribution (either end distribution or purity) are similar to the runs where the LOW scheme was used.

The satisfaction of the socialist and capitalist population structures are different by less than 1% and the head changes by at most 2.4 changes, although the structure within the subclusters is very different for both cases. This suggests that both structures are able to self-organise according to their needs and that they perform equally well.

We did not yet answer Question \textbf{Q5}, whether the qualitative evaluation enables the agents to make more informed choices with respect to self-organisation. Consider the case of fairness evaluation in combination with \textbf{P7}, for example. Here, self-awareness allows the agents to make informed choices as opposed to the random choice an external agency makes when appointing a different agent to the role of head. The results suggest that the informed choices are also the better choices, at least when it comes to the individual agent satisfaction. That is at least in all cases but one, when the agents fail to agree on a head (see Figure 6.10). The extended version of the testbed does not exploit the full spectrum of awareness. We believe that enabling the agents with a level of recursive consciousness and persistent self-awareness, i.e. the reflection over time with respect to the environment and the self, can resolve the issue of repeatedly voting for new heads although the environmental circumstances did not yet change. Appropriate functionalities can include a forgiving \texttt{droppedHeads} list on the agents’ side and exculpation procedures (\texttt{adrMethod}) for the head. It depends to what extent the agents can utilise the knowledge that they gained from evaluation (principles in use)
and to what extent such an evaluation is possible (level of awareness), but as we can see from the results presented here, adding a mechanism for evaluation from a system’s perspective enhances the success of self-organisation due to more informed agent choices.

6.6. Summary

In this chapter, we presented the principled operationalisation and controlled experimentation of the second iteration of the SIC methodology, in order to integrate a subjective evaluation mechanism. We showed that if the agents fail to introspect on fairness (on the example of the present allocation process), they run the risk of being exploited by the head agent without knowing about it. That is, the head might implement a fairness norm they do not approve of and therefore some individuals might get less resource allocated than they would otherwise. We tested this effect by endowing the agent with self-awareness but not giving them the capabilities to react on their justice perceptions. We saw that the satisfaction was significantly lower for these cases. In a second set of experiments, we gave the agent the capability to leave the cluster completely, which resulted in a higher satisfaction for the remaining members, but the total average outcome became worse as there was no alternative provided for the leaving agents.

The agents could take two sets of actions to counteract unfairness, depending on the principles used. $P7$ enables them to vote for an alternative head when the present head is perceived unfair, and $P8$ enabled them to create subclusters within the main cluster where each subcluster had their own head performing the resource-allocation process. Upon introspection, the agents chose whether to vote for an alternative head with a more suitable norm, or to migrate subclusters where there might be more agents sharing the same norm. Both methods lead to a significant increase in satisfaction and show that agents that subjectively evaluate the system performance make better choices for a successful self-organisation.

Deciding on how to instantiate the principles depends mainly on the environment and objectives of the open system, and the potential behaviour of the agents. The absence of $P7$ can hinder the self-organised execution of all other principles and impede sufficient membership. If maximising the total number of members is one of the goals then $P8$ can provide a valuable addition to prevent alienating members.
7. Summary, Conclusion and Future Work

In this chapter, we summarise the main findings of the testbed for self-organised resource allocation in open systems and the testbed extension, and we summarise the results of the five research questions that have been answered throughout this work. We then illustrate the limitations to this work and present future research directions.

7.1. Final summary

We considered the problem of self-organising resource allocation in open systems with the aim to develop a simulation testbed to test an institutional approach of the allocation process. We then used objective and subjective evaluation methods for judging the longevity, occupancy and fairness of the system. For the development we mainly considered two theories, one from political economy to address the issue of resource allocation, and one from computer science to address the issue of integrating the first theory into the open systems testbed. The obtained results show from an objective perspective (the designer’s view) that combining both theories leads to a sustainable (self-)management of the resource and has particular advantage when the endurance of the system is more important than short-term optimality.

In order to evaluate the system from a subjective perspective (the agent’s view), we took two more theories into consideration: One from the field of psychology and neuroscience on awareness and self-awareness, and one from psychology and organisational economics on organisational justice. We then extended the testbed functionalities and endowed the agents with the capability of exhibiting self-awareness. That allowed them to evaluate the fairness of the resource-allocation process and to take appropriate action to influence the processes or their position within the system.

We make three main contributions in this work: The first formal characterisation of Ostrom’s design principles for creating institutions for resource management in a rule management system; the design and implementation of a large-scale, reusable experimental testbed embedded in a multi-agent systems simulator, which transforms the design principles into policies and procedures that can be directly executed; and the two
sets of experiments conducted with this testbed showing that the Ostrom’s design principles are sufficient conditions for electronic institutions for enduring self-organisation of the system, and that the individual and collective performance can be further improved by mechanisms for subjective system evaluation. This provides a basis for evaluating procedural fairness and for cost-effective prevention of noncompliance. We equipped the open systems considered in this work, i.e. systems of rules and organisational structure, with the means to ensure a system’s operation in the best interests of its components, given only the sum of the components’ subjective opinions and their interactions with which to work.

7.1.1. Conclusions from the testbed and its extension

We modelled the open system using a formal characterisation of Ostrom’s eight design principles for managing common-pool resource institutions [85]. In Table 7.1 we show the benefits of each principle (see page 38) individually.

Table 7.1.: Benefits of Ostrom’s principles for resource allocation in open systems

<table>
<thead>
<tr>
<th>Principle</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>robustness to intentional violation by outsiders/overpopulation</td>
</tr>
<tr>
<td>$P_2$</td>
<td>robustness to environmental variation</td>
</tr>
<tr>
<td>$P_3$</td>
<td>robustness to environmental variation</td>
</tr>
<tr>
<td></td>
<td>robustness to ‘unfair’ behaviour</td>
</tr>
<tr>
<td>$P_4$</td>
<td>robustness to noncompliant behaviour</td>
</tr>
<tr>
<td>$P_5$</td>
<td>mitigation of intentional violation</td>
</tr>
<tr>
<td></td>
<td>tolerance to unintentional violation</td>
</tr>
<tr>
<td>$P_6$</td>
<td>repair of unintentional violation</td>
</tr>
<tr>
<td>$P_7$</td>
<td>robustness to arbitrariness/despotism</td>
</tr>
<tr>
<td>$P_8$</td>
<td>robustness to underpopulation</td>
</tr>
</tbody>
</table>

Principles $P_1$, $P_2$ and $P_3$ are used to respond to variations of brute facts $Bf$ in the environment $\epsilon$, such as the resource level or agents appropriating from the resource\(^1\). In the extended version of the testbed, $P_3$ is furthermore used to respond to variations of

\(^1\)Responding to the number of agents comes in two parts. Firstly, it defines a subset of agents that can appropriate, and secondly, prevents the remaining agents from appropriating.
institutional facts \textit{If}, i.e. to provoke role changes.

Principles \textit{P4}, \textit{P5} and \textit{P6} are used to respond to violation of institutional facts \textit{If}, such as appropriating more resource than was factually allocated as per the rules of the institution. Depending on the type of violation, intentional or unintentional, the principles ensure adequate methods to avoid such violations in future rounds\textsuperscript{2}.

Principles \textit{P7} and \textit{P8} provide an alternative to leaving the cluster based on how the agents perceive the institution, i.e. the implementation of rules and roles.

According to [14], how norms evolve depends on the nature of the population. Given an agent population where agents have roughly the same ideals and comply with the rules, \textit{P1–P3} are sufficient to manage the allocation process and to create an enduring CPR institution. If these assumptions are relaxed, we need \textit{P4–P6} to sanction misbehaviour and \textit{P7} and \textit{P8} to mitigate the conflicts that arise with heterogeneous norm perceptions and not make agents leave. These last two principles were furthermore used a subjective evaluation method to decide what actions to take for improving the system operation.

Accordingly, depending on the prevailing environment and agent population, a subset of principles is enough for a successful deployment of an open system. Deciding which principles should be used in a given environment becomes important when there is a cost of operation to cover, for example of monitoring, see Section 4.4.4. There are other operations that could entail a cost factor, such as the voting procedures, migrating subclusters, appealing to sanctions or if there is a cost of ownership to cover.

For every open system the decisions on how to instantiate the principles have to be carefully tailored to the agent population and (local) environment. Although the system is considered to be open, \textit{P1} can manufacture boundaries that ensure that the system does not get overcrowded. Otherwise, this can result in a harshly competitive environment due to the size of the system and variations on the availability of resources. It can even be to the extent that more resource has to be spent on \textit{P4–P6} for monitoring and sanctioning than what is used by the individual agents for achieving their personal goal, thus making the system operation highly inefficient. Enforcing boundaries becomes less crucial in systems where the operation improves with increasing numbers of members. However, monitoring can become even more important in order to keep ‘good’ members in the system that are resource deprived because of over-appropriating ‘bad’ agents and might leave due to dissatisfaction.

In the first set of experiments, we took \textit{P7} for granted, i.e. there was no possibility for principles to be challenged by external authorities. In the second set of experiments,

\textsuperscript{2}For intentional violations, the agents are given the possibility to revise their behaviour, resulting in fewer violations of this type; unintentional violations cannot be prevented and should not be sanctioned, hence \textit{P6}. 
only \( P3 \) became ineffective in the absence of \( P7 \). However, \( P7 \) is effectively intertwined with all other principles and needs to be present: Spending resources on monitoring agents without the possibility to prosecute them (within legal boundaries) or allocating resources without effect on appropriation volumes questions the benefit of self-organising the resource-allocation process.

Whether or not to instantiate \( P8 \) depends on the specific properties and goals of an open system. For some large systems, a hierarchical structure might be needed with different parts of the system being responsible for either constitutional, collective-choice or operational rules and their analysis. For other types of large systems, it might be enough to set up a flat hierarchy when most of the allocation process can be organised in a distributed fashion.

All eight principles have to be interpreted on a case by case basis, so that they can be implemented in different strengths accordingly. A great advantage of the principles is that they describe a basis for creating flexible rules that can be adapted to the system’s needs at runtime.

So far, we as designers made the decision on what principles to include in the testbed, but for a fully autonomous open system, this functionality has to be made available to the agents themselves as we cannot predict how the system will respond to particular situations, see also [111]. To this end, the agents have to be able to make decisions on a higher level of analysis, e.g. as part of the constitutional choices. The agents can make use of meta self-awareness (see Section 5.2.2), in order to judge the benefits of a new role or principle with respect to the prevailing environment and agent population.

As Ostrom wrote [84], rule changes are the means to change incentives, and in the best case that leads to a compliant population. The cost of adaptation is an important factor to consider, and can be expressed through a distance function between the specification instances of the open system, see Section 3.6. In [57], it is studied whether participants in a common-pool resource game invest enough (of their own) resource to achieve institutional change, and the question is positively answered. As long as the agents can expect benefits from changing a rule, role or principle, such as a higher satisfaction at no higher costs, these changes will be made, as presented in [17], for example.

Although the agents might not be able to take the optimal decisions (as could be possible in a closed system with complete information), giving the open system the possibility of self-analysis greatly improves the decision taking processes. This can be seen from the examples where the agents had to deal with incomplete information, such as monitoring intentional and unintentional violations (recursive consciousness to judge distributive fairness\(^3\)), or voting to appoint a new agent to a role (predictive self-awareness for judging procedural fairness). Fair procedures have a beneficial effect on intrinsic fairness.

\(^3\)That is the appropriation as actual outcome of the distribution process.
motivation\textsuperscript{4} \cite{124} that, if lacking, not only makes agents leave, but also enhances non-compliance \cite{32}, a route that we did not explore in this testbed.

7.1.2. Answering the five questions

Throughout this work, we asked five questions that relate to the process of resource allocation in open systems. The answers to these questions represent the main achievements of the thesis.

\textbf{Q1} Can the problem of resource allocation in open systems be addressed by modelling an institution for self-governance?

Using the methodology of sociologically-inspired computing, as presented in Section 3.2, we can model open systems as common-pool resources. In Section 3.9, we gave an indicative axiomatisation of the first six principles for governing common-pool resources, followed by the remaining two principles in Section 5.4. The axiomatisation provides a proof of concept for answering the first part of this question and brings together several strands of research in access control, voting and alternative dispute resolution. Concluding from the data evaluation in Section 4.4, institutions can be used to manage open systems, and there are multiple alternative formalisations of these mechanisms in various complexities to choose from.

\textbf{Q2} Can Ostrom’s design principles be encoded in norm-governed systems?

In Section 3.4 we bring together the work of Ostrom on socio-economic principles and the work of Artikis on dynamic specifications for norm-governed systems. The main aspect considered here is that the nesting of rules has to be maintained, and the correspondence in structure, i.e. the levels of analysis and levels from the protocol stack, allows for exactly that. Further important aspects when formalising an institution are the roles that the agents play in this institution and the rules defining what actions they are empowered, permitted or obligated to perform. Section 3.5 elaborates on this and gives detailed examples for the nesting of rules and relevance of roles on different levels.

\textbf{Q3} Is it possible to use the formal axiomatisation to specify and implement a testbed that ascertains the sufficiency of these principles for enduring open systems?

We implemented a testbed using all eight principles and tested their intended purposes. Principles \textit{P1–P3} are for managing the resource allocation considering the environment, such as the resource available. With all three principles in use and under the assumption

\textsuperscript{4}Here we can compare it to individual satisfaction.
of compliant agent behaviour\textsuperscript{5}, the resource is sustainable and the open system endures. Principles \textit{P4–P6} are used for responding to intentional and unintentional violation of allocation rules, e.g. appropriating more than the allocated resource. Their purpose is to ‘convince’ (through the use of sanctioning mechanisms) the agents to behave when appropriating or make them leave the cluster. Again, the more principles the better, but from several experiments (see Section 4.4.4) we conclude that the parameters and methods that are used have to be carefully adjusted to the local environment and prevailing agent behaviour. Principles \textit{P7} and \textit{P8} are used to keep the membership high in cases that could otherwise lead to unsatisfied agents leaving the cluster governed by the institution. In summary, as discussed in Section 7.1.1, the implementation of principles to govern the resource-allocation process (and accompanying issues) is indeed sufficient to create enduring open systems.

\begin{enumerate}
\item Can we equip the agents with mechanisms to evaluate the self-organisation?
\end{enumerate}

In order to evaluate this question, we endowed the agents with mechanisms that made them aware of their surroundings and themselves. The agents can use this mechanism of self-awareness to evaluate the institutional processes from an internal perspective and so evaluate the failure or success of the self-organisation, see Section 5.2.3. In this work, the agents exhibit predictive self-awareness for judging procedural fairness from a subjective perspective, a qualitative measure of self-organisation.

\begin{enumerate}
\item Does a qualitative evaluation of processes enable the agents to make more informed choices with respect to self-organisation?
\end{enumerate}

Self-awareness enables an agent to make informed choices, considering the environment, consequences of own actions, and other agents’ behaviour. In Section 6.5.3, the agents’ individual choices influence the system operation in such a way that it leads to greater (overall) satisfaction as opposed to choices taken by an external decision maker. There are many decision-making processes in the open systems testbed that can benefit from additional knowledge through self-awareness, but already at this stage we conclude that a qualitative evaluation by self-aware agents leads to better choices and a more successful self-organisation.

\section{Limitations}

There are several limitations to this work. These are connected to the parameters and principles, information, institutionalised power, scalability, autonomy, synchrony, self-
7.2. Limitations

awareness and cost. Ultimately, all of these limiting factors in combination (and quite possibly some more) will decide on the success or failure of an open system.

Parameters How we choose the parameters has a strong influence on the outcome of the simulation, two examples are presented here. The sequence of events for the CUSTOM replenishment scheme, for example, was chosen so that there are times of abundant and times of scarce resources available to the agents. Alterations on this sequence lead to different outcomes, however the ‘pure’ refill schemes (HIGH, MODERATE and LOW) show how the system performs for a steady rate of replenishment.

Further influence on the performance has the choice of numCheat, the number of noncompliant agents. The system supports 50% of agents that intentionally violate the rules, but from past experimentation we know numbers much higher than that cannot be supported and lead to a guaranteed depletion of the resource at an early stage. If nearly no agent has the sustainability of the resource as its goal, there remain little choices (compare P3) that this minority has impact on. If there are less than 50% noncompliant agents, we do not know how the system will behave. It could be the case that, depending on the environment, a low number of intentional violations leads to a higher efficiency not only for a high, but also a medium replenishment rate, compare Figure 4.14.

Principles The system’s operation is clearly influenced by the choice of principles we make and how we implement those, and we mentioned a few alternatives to P1–P6 throughout this work. We used P7 ‘for free’ in the first round of experimentation as we did not implement the system to be challenged by an external authority, but we did not exploit the full capacity of P8 (in the second round). Here, the agents evaluate the fairness of the allocation procedure on a subcluster level, the procedure on the cluster level that defines how to distribute the resource to the subclusters is not challenged. We defined equality as the distribution norm to be used, on the grounds that the distributor on the cluster level does not differentiate between the subclusters other than by member count. A choice that could be challenged by the subclusters, given they can derive additional information about the profile of other subclusters for evaluation. This represents a nested fairness judgement on two nested levels of system operation.

Information Throughout the system operation, agents have to deal with incomplete information. The first example is the sanctioning procedure, where accurate information on rule violations is dependent on the level of monitoring. The second example is the evaluation of the head’s fairness. The accuracy depends on the size of sampled agents. Both variables are set externally and for the first case, the effect of alternative levels has been shown, for the second case, we can merely compute a lower bound to estimate the
accuracy of fairness estimation. Although for some cases a more accurate information enhances the performance, for other cases it imposes a too high cost that cannot be borne by the system. There is no mechanism available to the agents of the testbed to decide what accuracy of information they require for a successful operation.

**Institutionalised power** The rules implemented in this testbed make only implicit use of the powers, permissions and obligations that are expressed through the Java classes. A role is only empowered to perform certain actions if this functionality is supported by the corresponding class and permitted to perform an action if the conditions for that function are met. Obligation to perform some action arises when a corresponding rule is fired. The implicit use of institutionalised power however, precludes a query that lists the specific powers, permissions and obligations for any single role.

In order to explicitly express these we could insert the corresponding facts into the working memory and conditions into the production memory of Drools. It would then be possible to query the state of the system and retrieve information about what agents occupy what roles at a certain point in time and what are the powers, permissions and obligations associated with that role.

**Scalability** In the scope of the simulation testbed, we believe scalability is not a problem. The complexity of \( \text{raMethod} \) is linear with \(|A|\), the most expensive method is \( \text{feMethod} \) whose complexity is at most squared \((|A|^2)\). This is due to the relatively simple (in comparison with alternatives) implementation of methods. Moreover, increasing the population by a large factor does typically not yield much more information on the system performance in simulation.

For real world applications however, scalability is of greater concern. Depending on the type of open system (e.g. wireless sensor network, vehicular ad hoc network or virtual organisation), the complexity of certain tasks can be an inhibiting factor for a successful system operation. The space complexity (e.g. memory requirements for a queue or sanctioning facts) grows linearly with the amount of agents, as does the computational complexity per agent (even for \( \text{feMethod} \)). Significant factors are the time constraint and network load. The frequency with which the rounds have to occur for a meaningful operation could cause high bandwidth requirements and be unfeasible for certain type of networks (e.g. WSN). These factors also depend on the specific method implementations, meaning when the lightweight methods in this work are substituted by computationally more intensive approaches.

**Autonomy** The autonomy of agents is limited by several factors. In the step of principled operationalisation, we make several assumptions on the behaviour of agents and
the environment that are reflected in the computer model and affect the observed performance. Moreover, we predefine how the agents solve a certain issue. Some of them are solved using self-aware techniques, whereas others are decided at random. Ideally, the agents decide for themselves to what detail they need to analyse a certain situation, depending on how much utility they can gain in contrast to random decisions.

**Synchrony**  The simulation operates in time slices (rounds), and every issue (voting, appeals, etc.) is resolved in the same round. In reality however, the issues that occur on different levels of analysis do not have to be solved simultaneously and within one round, thus can be implemented asynchronously in so called ‘action situations’. These action situations then occur on different levels of the protocol stack and can take varying times to be resolved. For example, whilst two parties are engaged in dispute resolution on the collective-choice level, they can still perform actions at the operational choice level, such as monitoring or appropriating.

The simultaneous turn taking has additional effects on the agent strategies. If some agent does or does not know another agent’s move before the next ruleflow group becomes active, the games that can be played differ from one another. For example, in the first case an auction turns into an open price auction, whereas in the second case it is a sealed bid auction [31].

**Self-awareness**  As mentioned before, the principles have to be implemented carefully according to the prevailing environment. There is no one-size-fits-all strategy so that a trade-off between parameters will not optimise the results. Adding self-awareness as a mechanism for the agents to learn the environmental and agent states and how to respond to them, proved to improve the outcome but so far self-awareness has only been used for a limited amount of decisions. There remains a wide range of variables to be analysed for the system to autonomously choose the parameters of the principles and ideally create further rules themselves.

**Cost**  One of the factors that are influenced by choices is the cost factor. We did not make extensive use of costs in this implementation, but in addition to monitoring costs, further costs for dispute resolution, applying for membership, voting for a new raMethod or head, for example, can be introduced. Some of them are one-off transaction costs (new head) and some are recurring costs (monitoring procedure). When deciding on a different specification instance $l \in \mathcal{L}$ (i.e. a different set of parameters), these costs, that are reflected by the distance $d$, have to be taken into account.
7. Summary, Conclusion and Future Work

7.3. Future Work

There are several lines of research we consider for future work. These include enhancing the testbed with alternative and additional institutional and brute facts, or improving the agents’ fairness perception with an evaluation of applicable fairness components using six rules. Further ideas concern the judgement of agent profiles that has so far been predefined, but could be obtained by social networking techniques. The last line is to instantiate the abstract model presented in this work for developing and deploying an autonomic power system for the management of energy distribution.

7.3.1. Principles and methods: Institutional facts

Ostrom’s eight principles have proved to be sufficient for enduring CPR management, but there has been work done to extend these with over 30 factors that influence endurance [1] or subdivide some principles into further cases [34]. One point of critique is that the principles do not address the social mechanisms (such as trust or transparency) as an important factor for failure or success. So far, the testbed includes fairness as a social mechanism that influences action selection, but this represents only one cause that affects human behaviour [70]. Possible extensions to the testbed can include mechanisms for trust, following up on our previous publication on trusted communities [15].

We only chose a relatively simple implementation of the different methods, whereas there are more complicated methods available from the literature to be integrated into the computer model. There are role-based access control methods [101] for distributed environments or floor-control protocols for open systems [8] that can be used as alternative acMethods. On the issue of winner determination (wdMethod), there are various alternative protocols that can be used, as presented in [91] or [115], for example, bearing in mind that each wdMethod has a different level of robustness to strategic manipulation. Depending on the nature of dispute, different adrMethods can be used, compare for example [62] and [117].

In future work, we could extend the preformal theory with further principles, social mechanisms and alternative implementations of methods. The additional principles will cover a broader range of reactions to a changing environment and agent behaviour, and the different methods can be used to tailor the testbed to the capabilities of specific types of open systems. The use of alternative methods is likely to come at varying costs and complexities that can also depend on the choice of other degrees of freedom. Our ultimate goal is for the agents to choose which specification instance is the most appropriate, considering the cost involved and prospects of the system to endure.
7.3. Future Work

7.3.2. Environment: Brute facts

The prevailing environment in this work included exogenous replenishment of a divisible resource. We can modify the testbed to include different replenishment types, such as endogenous replenishment or a combination of both, and different resource types, such indivisible or reusable resources. One possibility for an endogenously provided and reusable resource is a knowledge commons, with a popular example being Wikipedia [45]. This type of CPR has been analysed from the perspective of institutions [54] and presents the benefits and drawbacks of digital knowledge and its impact on intellectual property, overpricing and preservation.

An interesting line of research is to model policies at the design stage before they are put into effect by the legislation, and analyse their impact the evolution of norms and the behaviour of the society. [21] presents an approach for modelling environmental policies to understand the interaction of agents, actions and norms in socially and environmentally coupled systems. We can utilise their findings to optimise the policy-making process for knowledge commons.

7.3.3. Fairness perception of procedures

In the scope of this work, we judged fairness using counterfactual thinking\(^6\) applied to a subset of seven procedural components. An alternative to this approach are the six justice rules as formulated by Leventhal [70]. They describe factors that, when found present or absent upon evaluation, increase or decrease the perceived procedural justice by the evaluating agent.

- Consistency rule—The allocation procedure should be consistent over time and across actors.
- Bias-suppression rule—The procedure should not be dictated by self-interest or preconceptions, and preferably, judicial and adversary roles are separated.
- Accuracy rule—Record keeping and accountability are important factors to ensure the integrity of the allocation process. It should be based on reliable information and informed opinions.
- Correctability rule—The possibility to modify and reverse decisions during the allocation procedure has to be offered.
- Representativeness rule—The values, concerns and outlooks of important subgroups of individuals affected by the allocation process must be considered. This

\(^6\)Counterfactual thinking concerns the questions of what would/could/should have happened.
rule is relevant for participatory decision making.

- Ethicality rule—This rule states that the procedures have to be compatible with the moral and ethical values by the evaluating individual.

For a fairness judgement, an agent does not have to perform a ‘full’ evaluation, but can evaluate any of the seven procedural components using any subset of the six justice rules, and then aggregate these partial results to a final judgement.

In future work, we could redesign the fairness evaluation applying the concept just described. In total there are 42 evaluations that can be made when judging procedural justice, though not each of them might be applicable to open systems. Applicable judgements can be aggregated according to an agent’s individual preference, for example with a weighted sum.

Human actors apply such rules selectively and follow different rules at different times or attach different importance to them. Several factors are responsible for determining to what extent an actor is concerned about fair procedures. The actor can occupy a certain role that requires maintaining fairness, or can be occupied with goals of greater concern than fairness. Typically, an actor is more likely to evaluate the fairness of a situation when the suspicion arises that fair procedures have been violated or there are sudden changes of operation.

This last set of criteria can be leveraged for judging fairness in a security context with constrained resources. A possible attack will only be investigated in detail, if there is reason to suspect unfair behaviour (such as a battery exhaustion or other denial of service attacks [121], for example). Furthermore, it is important to distinguish between ‘true fair’ behaviour from ‘quasi-fair’ behaviour [70]. This latter type resembles the former type of behaviour and arises when agents set up a strategy to manipulate other agents or exploit the system. Quasi-fair behaviour will be abandoned as soon as it has served its purpose or proves ineffective in achieving the final goal (e.g. a security breach). However, it might be much harder to game a system where the agents evaluate fairness (even without the distinction of quasi-fairness), so that this mechanism can serve as an primary disincentive for potential attackers.

### 7.3.4. Social networking

When a head performed an allocation according to its internal distribution norm, the resource was allocated according to information on the agent profile (meritorious or needy). This information was predefined for each agent at start and did not change.

---

7This was realised in the testbed with agents having different norm preferences at different times, i.e. the groups ‘equityChange’ and ‘equalityChange’.
7.3. Future Work

throughout the course of a single simulation.

In future work, this information does not have to be predefined but can be gained from (social) interactions between agents. We can interleave a cycle of opinion formation and belief revision within each agent that allows them to form internal profiles of other agents, which can be updated in each time slice, as presented in [92]. The process of opinion formation does not have to lead to the same opinion on an issue within every agent, it is more likely that for diverse initial opinions, clusters of similar opinions will evolve, as presented in [53], for example.

The same mechanism can be used to form an opinion about the prevailing norm that is used by other agents when it is their turn to perform a resource allocation. From this, an agent can revise its own interpretation of norms. The consequences are twofold. Firstly, an agent can adjust its behaviour to match the perceived norm when allocating resources. Secondly, an agent can adjust its behaviour when resource is allocated according to a norm different from his. If that different norm is perceived to be unfair\(^8\), an agent is less motivated to comply with the rules of the institution (e.g. on appropriation), as stated in [32] or [69].

It would be interesting to investigate the effect of fairness perceptions on agent compliance (instead of ‘automatic’ behaviour revision, see A.4) and how this affects the sustainability of the resource and endurance of the open system.

7.3.5. Assisted resource allocation

We presented three types of organisational justice: distributive justice which is related to fair allocations, procedural justice which is related to fair procedures, and interactional justice which is related to fair interpersonal treatment. The first two types can be analysed through simulations of open systems, such as in Chapter 4 and 6, where we considered automated resource allocation. For the third type, simulations are not suitable. The idea of assisted resource allocation involves an open system where the agents contain an interface to human actors that influence the agents’ behaviour. An example where such an artificial system (a big step up from simulation [112]) can be used is demand-side management of infrastructure, such as energy, water or transport.

Take a local neighbourhood in a smart grid, for example. There are human actors that both provide (with the use of solar panels, for example) and consume energy. The challenge for the electricity grid is to balance out the need and generation of energy. This means that the agents cannot supply and appropriate resources to their liking but have to make arrangements with each other. The decision making takes place in an

\(^8\)That is most likely the case, otherwise they would be in the same ‘norm-cluster’.
automated fashion, but reflects the preferences of the individual user.

We would like to use electronic institutions to manage electricity as a common-pool resource. For first experimentation, the behaviour of the agents can be simulated following the (noncooperative) generalised Nash equilibrium problem\textsuperscript{9} [6] which has applications to electricity markets, or by using divisible good auctions \textsuperscript{65}. Interesting aspects include a system of local neighbourhoods, where entire neighbourhoods are competing with each other (viewed as a single entity) when offering flat-rate, on-demand, or spot-market resource access.

Simulating smart grids as CPRs furthermore presents unique challenges and opportunities for policy-makers. They can test the effect of policies that are meant to incentivise micro-level behaviour, aiming for macro-level improvement with respect to minimising energy consumption, reducing carbon emissions and encouraging sustainable living. This requires understanding how old and new policies interact with each other, how new policies impact on the user behaviour and perception, and how this behaviour affects the selection of new policies. (Self-)awareness is an important component when trying to understand the interconnectedness of policies and human behaviour when it comes to instantiating the real-world application. When humans are taking part in the design of the institution, self-awareness is used to judge interactive justice during conflicts which in turn can help to promote trust \textsuperscript{30} in the interaction of humans and technology (here the smart meter, for example).

\textsuperscript{9}Also known as ‘social equilibrium problem’ or ‘abstract economy’, the GNEP represents the problem of finding a Nash equilibrium in a game where a player’s action has impact on the payoff and possible actions by other players, as presented in [7].
Bibliography


2009, pp. 83–93. 167


[34] Michael Cox, Gwen Arnold, and Sergio Villamayor Tomás, A review of design principles for community-based natural resource management, Ecology and Society
Bibliography

15 (2010), no. 4, 38. 166


[38] Bruce Edmonds, Nigel Gilbert, Steven Gustafson, David Hales, and Natalio Krasnogor (eds.), *Proceedings of the socially inspired computing joint symposium (AISB’05 convention)*, London, UK, AISB, 2005. 47, 174, 181


[46] Ian Foster, Carl Kesselman, and Steven Tuecke, *The anatomy of the grid: En


[59] Andrew J. I. Jones and Marek Sergot, *A formal characterisation of institutionalised


[74] Elizabeth A. Mannix, Margaret A. Neale, and Gregory B. Northcraft, Equity, equality, or need? The effects of organizational culture on the allocation of benefits and burdens, Organizational Behavior and Human Decision Processes 63 (1995), no. 3, 276–286. 118, 121, 126


Bibliography


[87] Elinor Ostrom and Charlotte Hess, *A framework for analyzing the knowledge commons*, in Hess and Ostrom [54], pp. 41–82. 32, 36


Bibliography


[120] Adam Wierzbicki, *Trust and fairness in open, distributed systems*, Springer,
Berlin/Heidelberg, Germany, 2010. 123


[123] Stephen M. Younger, Behavioural norms and the evolution of societies, in Edmonds et al. [38], pp. 103–108. 51


[125] Jennifer Zelmer, Linear public goods experiments: A meta-analysis, Experimental Economics 6 (2003), 299–310. 33


Note: The numbers listed behind the full stop of each bibliography entry refer to the pages on which the entry is cited.
A. Class methods and rules

A.1. Member :: demand

```java
public double demand(Institution i, CommonPool pool) {
  if (active && i.isPrinciple2() && Random.randomDouble() > noRequestPercentage) {
    switch (i.getAllocationMethod()) {
    case QUEUE:
      demand = preferredRequest;
      break;
    case RATION:
      if (pool.getResourceLevel() * compliancyDegree / i.getActiveMemberCount() < preferredRequest) {
        if (!i.isPrinciple4()) {// no monitoring
          demand = pool.getResourceLevel() * compliancyDegree / i.getActiveMemberCount();
        } else {// with monitoring
          demand = (pool.getResourceLevel() - i.getActiveMemberCount() * i.getMonitoringCost() * i.getMonitoringLevel()) * compliancyDegree / i.getActiveMemberCount();
        }
      } else {
        demand = preferredRequest;
      }
      break;
    } // end switch
  } else demand = 0;
  if (demand < 0) {
    demand = 0;
  }
  return demand;
}
```
A. Class methods and rules

A.2. Member :: vote

```java
public Vote vote(Institution i, CommonPool pool, String ballot) {
    if (ballot.equals("raMethod") && i.isVoteRaMethod()) {
        RaMethod vote;
        if (pool.getResourceLevel() < 0.75 * i.getMaxLevel() / compliancyDegree) {
            vote = RaMethod.RATION;
        } else {
            vote = RaMethod.QUEUE;
        }
        return Vote.voteRaMethod(vote);
    }
    return null;
}
```

A.3. Member :: appropriate

```java
public double appropriate(Institution i, CommonPool pool, Allocation all) {
    if (active) {
        double appropriateAmount = 0;
        if (i.isPrinciple2()) {
            double amount = (all == null ? 0 : all.getQuantity());
            if (compliancyDegree > 1) { // wrongful appropriation
                if (amount + (preferredRequest - standardRequest) < demand){
                    appropriateAmount = amount + (preferredRequest - standardRequest);
                } else {
                    appropriateAmount = demand;
                }
            } else { // appropriate allocation
                appropriateAmount = amount;
            }
        } else if (Random.randomDouble() > noRequestPercentage) { // Principle 2 disabled
            appropriateAmount = preferredRequest;
        }
    } // not every agent subject to noise
```
if (pool.isUnintentionalError() && Random.randomDouble() < i.getNoisePercentage()){
    double share = standardRequest;
    if (i.isPrinciple2() && i.getAllocationMethod() == RaMethod.RATION ){
        share = i.getFairshare();
    }
    if (Random.randomDouble() < 0.5) {
        appropriateAmount += share*i.getNoiseLevel()*Random.randomDouble();
    } else {
        appropriateAmount -= share*i.getNoiseLevel()*Random.randomDouble();
    }
}
if (appropriateAmount < 0){
    appropriateAmount = 0;
}
return appropriateAmount;
} else { //inactive agents
    return 0;
}
A. Class methods and rules

A.5. Member :: appeal / Head :: uphold / Member :: apply / Gatekeeper :: include

```
1 rule "Members appeal against sanction"
2 ruleflow-group "appeal"
3 when
4   | $i: Institution($iid:id, $r: round, pr6=true)
5   | $m: Member($n:name, instId==$iid, active=false)
6   | $san: Sanctioned(agent==$n, round==$r, inst==$iid)
7   | not (Appealed(agent==$n, $aRd:round, inst==$iid, $r-$aRd < $i.appealtime))
8   | not (Sanctioned(agent==$n, $sRd:round, $l: level, $sRd<$r, $r - ($sRd + $l*$i.excludetime) <= $i.appealtime, inst==$iid))
9 then
10 | insert(new Appealed($name, $round, $iid));
11 | retract($san);
12 | modify ($m){
13 |   | setActive(true)
14 | }
15 end
```

A.6. NonMember :: appropriate

```
public double appropriate(CommonPool pool) {
    double appropriateAmount = 0;
    if (initialCompliancyDegree > 1 && active && Random.randomDouble() < pool.getOutAppropriationFrequency()) {
        appropriateAmount = standardRequest * initialCompliancyDegree;
    }
    return appropriateAmount;
}
```

A.7. Gatekeeper :: assign

```
rule "Assign new head"
```
A.8. Gatekeeper :: include()

```plaintext
<table>
<thead>
<tr>
<th>ruleflow-group &quot;exclude&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>when Institution($iid:id, $r:round)</td>
</tr>
<tr>
<td>not ( exists(Head(instId == $iid)) )</td>
</tr>
<tr>
<td>$mL: List( size &gt; 0 ) from collect(Member(instId==$iid))</td>
</tr>
<tr>
<td>then</td>
</tr>
<tr>
<td>Member m = (Member) ← mL.get(Random.randomInt($mL.size()));</td>
</tr>
<tr>
<td>retract(m);</td>
</tr>
<tr>
<td>insert(new Head(m));</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>
```

A.8. Gatekeeper :: include()

```plaintext
| rule "Include member after sanction" |
| ruleflow-group "exclude" |
| no-loop |
| when |
| $i: Institution($iid:id, $r:round, pr5==true) |
| $m: Member($n:name, institutionId==$iid, active==false) |
| $s : Sanctioned(agent==$n, $sanRd:round, $level:level, ← ($sanRd + $level*i.excludetime < $r), $level <= ← i.maxSanctionLevel) |
| not Sanctioned(agent==$n, $rd:round, $rd > $sanRd) |
| then |
| if(Random.randomDouble() < 0.1){ |
| $m.changeBehaviour($i.getMaxSanctionLevel(), ← $s.getLevel()); |
| modify ($m){ |
| setActive(true) |
| } |
| } |
| end |
```

A.9. Gatekeeper :: exclude

```plaintext
| rule "Member exclusion" |
| ruleflow-group "exclude" |
| when |
| Institution($iid:id, $r:round, pr5==false) |
```
A. Class methods and rules

```java
$m$: Member($n:name, instId==$iid, active==false)
Sanctioned(agent==$n, round==$r)
then
retract($m);
insert(new NonMember($m));
end

rule "Member exclusion with graduated sanctions"
ruleflow-group "exclude"
when
$i$: Institution($iid:id, $r:round, pr5==true)
$m$: Member($n:name, instId==$iid, active==false)
Sanctioned(agent==$n, round==$r, level > $i.maxSanctionLevel)
then
retract($m);
insert(new NonMember($m));
end

A.10. Monitor :: monitor / Monitor :: monitor_out
```

```java
public Set<String> monitor(Institution i, CommonPool pool, Set<Member> members, Set<Agent> nonMembers) {
    Set<String> toMonitor = new HashSet<String>();
    monitoring = 0; // counter
    outMonitoring = 0;
    if (i.isPrinciple4()) { // member monitoring
        for (Member ag : members) {
            if (ag.active && Random.randomDouble() < i.getMonitoringLevel()) {
                toMonitor.add(ag.getName());
                monitoring ++;
            }
        }
    } else {
        if (i.isPrinciple1()) { // non-member monitoring
            for (Agent ag : nonMembers) {
                if (ag.active && Random.randomDouble() < i.getOutMonitoringLevel()) {
                    toMonitor.add(ag.getName());
                    outMonitoring ++;
                }
            }
        }
    }
    return toMonitor;
}
```
A.11. Head :: call for vote

```java
public CallForVote callForVotes(Institution i) {
    if (i.isPrinciple3()) { // on (head, raMethod)
        return new CallForVote(false, true);
    } else {
        return null;
    }
}

rule "Call for votes"
ruleflow-group "cfv"
no-loop
when
    $i: Institution($iid:id, pr3==true)
    $hd: Head(instId==$iid)
then
    CallForVote cfv = $hd.callForVotes($i);
    if(cfv != null) {
        modify($i) {
            setVoteHead( cfv.isHead() ),
            setVoteRaMethod( cfv.isRaMethod() )
        }  
    }
```

A.11. Head :: call for vote

```java
rule "Agent monitoring list"
ruleflow-group "allocate"
when
    $i: Institution($iid:id, $p:pool, $r:round)
    $mon: Monitor(instId==$iid, pool==$p.id)
    $mL: Set() from collect( Member(instId==$iid) )
    $nmL: Set() from collect( NonMember(pool==$p.id) )
    not TaskExecuted( $i, "monitor", $round ;)
then
    Set<String> monitored = $mon.monitor($i, $p, $mL, $nmL);
    for( String agent : monitored ) {
        insert( new Monitored( agent , $r) );
    }
    insert( new TaskExecuted( $i, "monitor", $round ) );
end
```
A. Class methods and rules

A.12. Head :: declare

```java
rule "Count votes"
  ruleflow-group "vote"
  when
  | Institution($iid:id, $r:round, pr3==true)
  | Member($ag:name, instId==$iid)
  | Vote($voter==$ag, round==$r, $b:ballot)
  | not VoteCount(inst==$iid, ballot==$b, round==$r)
  | $votes: List(size > 0) from accumulate($v:
    | Vote($voter:$voter, round==$r, ballot==$b) and
    | Member(name==$voter, instId==$iid), collectList($v))
  then
  | HashMap<Integer, Integer> tally = new HashMap<Integer, Integer>()
  | for(Object o : $votes) {
    | Vote v = (Vote) o; //(ballot.value)
    | if(tally.containsKey(v.getValue())) {
    |     tally.put(v.getValue(), tally.get(v.getValue()) + 1);
    | } else {
    |     tally.put(v.getValue(), 1);
    | }
  }.
  | insert(new VoteCount($iid, $b, $r, tally));
end

rule "Declare winner and update raMethod"
  ruleflow-group "vote"
  when
  | $i: Institution($iid:id, $r:round, pr3==true)
  | $vc: VoteCount(ballot=="raMethod", round==$r, inst==$iid)
  then
  | Integer forQueue = $vc.result.get(RaMethod.QUEUE.ordinal());
  | Integer forRation = $vc.result.get(RaMethod.RATION.ordinal());
  | if(forQueue==null) forQueue = 0;
  | if(forRation==null) forRation = 0;
  | if(forQueue > forRation) {
```
A.13. Head :: allocate

```java
public Set<Allocation> allocate(Institution i, CommonPool pool, List<Demand> demands) {
    Set<Allocation> allocations = new HashSet<Allocation>();
    double level = pool.getResourceLevel() - monitoring * i.getMonitoringCost() - outMonitoring * i.getOutMonitoringCost();
    if (level < 0){
        return allocations;
    }
    switch (i.getAllocationMethod()) {
    case QUEUE:
        Collections.shuffle(demands);
        Queue<String> demandQueue = i.getDemandQueue();
        Set<String> alreadyDemanded = new HashSet<String>();
        for (Demand d : demandQueue) {
            alreadyDemanded.add(d.getAgent());
        }
        for (Demand d : demands) {
            if (!alreadyDemanded.contains(d.getAgent())) {
            demandQueue.add(d);  
            }
        }
        while (!demandQueue.isEmpty()) {
            if (level >= demandQueue.peek().getQuantity()) {
                Demand d = demandQueue.poll();
                allocations.add(new Allocation(d.getAgent(), i.getRound(), d.getQuantity(), i.getPlook()));
                level -= d.getQuantity();
            } else {  
                break;  
            }
        }
        break;
    }
```
A. Class methods and rules

```java
30 case RATION:
31    double fairshare = level / demands.size();
32    i.setFairshare(fairshare);
33    Collections.shuffle(demands);
34 for (Demand d : demands) {
35        if (level >= d.getQuantity() || level >= fairshare){
36            if (d.getQuantity() > fairshare){
37                allocations.add(new Allocation(d.getAgent(),
38                    i.getRound(), fairshare, i.getPool()));
39                level -= fairshare;
40            } else {
41                allocations.add(new Allocation(d.getAgent(),
42                    i.getRound(), d.getQuantity(), i.getPool()));
43                level -= d.getQuantity();
44            }
45            } else {
46                break;
47            }  
48        }  
49        break;
50    }// end switch
51 return allocations;
```

A.14. Head :: sanction

```sql
1 rule "Sanctioning monitored members (first offence)"
2 ruleflow-group "report"
3 no-loop
4 when
5     Institution($iid:id, $p:pool, $r: round)
6     $m: Member($n:name, instId==$iid, pool==$p.id)
7     Monitored(agent==$n, round==$r)
8     not Sanctioned(agent==$n, round==$r)
9     (  
10        (Allocation(agent==$n, round==$r, $alloc:quantity, pool==$p) and
11            Appropriated(agent==$n, round==$r, $approp:quantity, pool==$p, $approp > $alloc ) )
12        or
13        (not Allocation(agent==$n, round==$r, pool==$p) and
14            Appropriated(agent==$n, round==$r, $approp:quantity, pool==$pool, $approp > 0 ) )
192```
A.14. Head :: sanction

rule "Sanctioning monitored members (repeatedly)"
ruleflow-group "report"
no-loop
when
| Institution($iid:id, $p:pool, $r:round, pr5==true)
| $m: Member($n:name, instId==$iid, pool==$p.id )
| Monitored(agent==$n, round==$r)
| not Sanctioned(agent==$n, round==$r)
| ( Allocation(agent==$n, round==$r, $alloc: quantity, ← pool==$pool) and
| Appropriated(agent==$n, round==$r, $approp: quantity, ← pool==$pool, $approp > $alloc) )
| or
| ( not Allocation(agent==$n, round==$r, pool==$p) and
| Appropriated(agent==$n, round==$r, $approp: quantity, ← pool==$p, $approp > 0) )
| )
| Sanctioned(agent==$n, $level:level)
| not Sanctioned(agent==$n, $l:level, $l > $level)
then
| insert(new Sanctioned($n, $r, $l+1, $iid));
| modify ($m){
| setActive(false)
| }
end

rule "Sanction monitored non-members"
ruleflow-group "report"
no-loop
when
| Institution($iid:id, $p:pool, $r:round, pr1 == true)
| $nm : NonMember($n:name, pool==$p.id)
| Monitored(agent==$n, round==$r)
| not Sanctioned(agent==$n, round==$r, inst==$iid)
| Appropriated(agent==$n, round==$r, $approp: quantity, ← pool==$p, $approp > 0)
A. Class methods and rules

```java
then
| insert(new Sanctioned($n, $r, 1, $iid));
| modify ($nm){
|  | setActive(false)
| }  
| modify($p){
|  | setOutAppropriationFrequency(
|  |    getOutAppropriationFrequency() -
|  |    (getOutAppropriationFrequency() * getOutImproveFrequency()))
| }  
end
```

A.15. CommonPool :: refill

```java
rule "Refill pool"
salience -10
ruleflow-group "init"
when
| $t: IntegerTime() // same as round
| $pool: CommonPool($t.intValue()>getLastFilled(),
|    getRefScheme()==RefillScheme.CUSTOM)
then
| double fillAmount = $pool.getMaxLevel() / 2;
| int time = $t.intValue();
| if(time < 50) {
|  | fillAmount *= 1;
| } else if(time < 100) {
|  | fillAmount *= 0.95;
| } else if(time < 150) {
|  | fillAmount *= 0.87;
| } else if(time < 200) {
|  | fillAmount *= 0.52;
| } else if(time < 250) {
|  | fillAmount *= 0.92;
| } else if(time < 300) {
|  | fillAmount *= 0.97;
| } else if(time < 350) {
|  | fillAmount *= 0.62;
| } else if(time < 400) {
|  | fillAmount *= 0.90;
| } else if(time < 450) {
|  | fillAmount *= 0.50;
```
A.16. Termination criteria

```java
// simulation loop
while (t.intValue() < sim.finishTime) {
    logger.info("Round " + t.intValue());
    session.startProcess("allocation.Simulation");
    session.fireAllRules();
    t.increment();
    session.update(session.getFactHandle(t), t);
}
```

```java
A.16. Termination criteria
```

```java
1 // simulation loop
2 while (t.intValue() < sim.finishTime) {
3     logger.info("Round " + t.intValue());
4     session.startProcess("allocation.Simulation");
5     session.fireAllRules();
6     t.increment();
7     session.update(session.getFactHandle(t), t);
8 
9     rule "Refill pool - HIGH/MODERATE/LOW scheme"
10     salience -10
11     ruleflow-group "init"
12     when
13         $t : IntegerTime()
14         $p: CommonPool($t.intValue() > getLastFilled(), ←
15             getRefScheme() != RefillScheme.CUSTOM)
16     then
17         double fillAmount = $p.getMaxLevel()/2;
18         int time = $t.intValue();
19         if(time < 50) {
20             fillAmount *= 1;
21         } else if ($pool.getRefScheme() == RefillScheme.HIGH){
22             fillAmount *= 0.95;
23         } else if ($pool.getRefScheme() == RefillScheme.MODERATE){
24             fillAmount *= 0.80;
25         } else( //RefillScheme.LOW
26             fillAmount *= 0.50;
27         }
28     modify($p) {
29         setResourceLevel($pool.getResourceLevel() + fillAmount),
30         setLastFilled( time )
31     }
32     end
33 
34     // simulation loop
35     while (t.intValue() < sim.finishTime) {
36         logger.info("Round " + t.intValue());
37         session.startProcess("allocation.Simulation");
38         session.fireAllRules();
39         t.increment();
40         session.update(session.getFactHandle(t), t);
41 
42     
43     
44     
45     
46     
47     
48     
49     
50     
51     
52     
53     
54     
55     
56     
57     
58     
59     
```
A. Class methods and rules

```
8 }
10 rule "Common pool depleted"
11   ruleflow-group "init"
12 when
13   | $p: CommonPool(resourceLevel < 0)
14   | Institution($iid : id, pool==p)
15   | $mL : List() from collect(Member(instId==$iid))
16 then
17   | retract($pool);
18   | for(Object m : $members) {
19     | retract(m);
20     | insert(new NonMember((Member)m));
21   } end
24 rule "End of the institution"
25 ruleflow-group "init"
26 when
27   | $i: Institution($iid:id)
28   | not(exists(Member(instId==$iid)))
29 then
30   | retract($i);
31 end
```

A.17. Control loop for CPR testbed
A.17. Control loop for CPR testbed

Algorithm 4: Control loop for CPR testbed

\[ \exists P \in \epsilon \subseteq E: P \text{ is pool}; \]
\[ \exists I \in L: I \text{ is institution with } P_1 \leftarrow \text{true}; \]
\[ \forall m \in M \subseteq A: \text{ role of } m \text{ is member}; \]
\[ \forall nm \in A \setminus M: \text{ role of } nm \text{ is non-member}; \]
\[ \exists h \in M: \text{ role of } h \text{ is head}; \]
\[ \exists gk \in M: \text{ role of } gk \text{ is gatekeeper}; \]
\[ \exists mon \in M: \text{ role of } mon \text{ is monitor}; \]
\[ r \leftarrow 0; P \leftarrow P_{\max}; \]
\[ \text{while } (M \neq \emptyset) \land (P \geq 0) \land (t < \text{FinishTime}) \text{ do} \]
\[ P \leftarrow \min (P_{\max}, P + P_{\text{refill}}); \quad \text{ // refill common pool} \]
\[ \text{if } P_3 \text{ then} \]
\[ h \text{ calls for vote on } raMethod; \]
\[ \forall m \in M: \text{ } m \text{ votes; } \]
\[ h \text{ counts votes and declares } raMethod; \]
\[ \text{else} \]
\[ \text{ automatic choice of } raMethod; \]
\[ \text{if } P_2 \text{ then} \]
\[ \forall m \in M: m \text{ places demand; } \]
\[ h \text{ allocates resource to all } m \text{ according to } raMethod; \]
\[ \forall nm \in A \setminus M: \text{ } nm \text{ appropriates resource } R_{nm}; \quad \text{ // only noncompliant nm} \]
\[ P \leftarrow P - \sum_m R_m - \sum_{nm} R_{nm}; \]
\[ \text{mon reports offences by non-members at cost } P_{nonm}; \]
\[ h \text{ sanctions reported non-members: activity} \leftarrow \text{false}; \]
\[ P \leftarrow P - P_{nonm}; \]
\[ \text{if } P_4 \text{ then} \]
\[ mon \text{ reports offences by members at cost } P_{mon}; \]
\[ P \leftarrow P - P_{mon}; \]
\[ \text{if } P_5 \text{ then} \]
\[ h \text{ sanctions reported members at level } S \text{ (activity} \leftarrow \text{false)}; \]
\[ gk \text{ includes members with served sentence (activity} \leftarrow \text{true)}; \]
\[ \text{else} \]
\[ h \text{ sanctions reported members at level } 1; \]
\[ \text{if } P_6 \text{ then} \]
\[ \text{sanctioned members appeal against sanction;} \]
\[ h \text{ upholds sanction and gk includes members;} \quad \text{ // if applicable} \]
\[ gk \text{ excludes sanctioned members} \leftarrow \text{active non-members;} \quad \text{ // if } P_5 \text{ dep. on } S \]
\[ \text{assign missing roles;} \quad \text{ // at random} \]
\[ r \leftarrow r + 1; \quad \text{ // increment round} \]
B. Class field values and experimental parameters

These are the initialisation values and the field values of the classes used in the implementation of the testbed, see page 84, tables describing the setup for individual runs with corresponding figures are given after. The roles of head, monitor and gatekeeper extend the member class and have no additional fields associated with them. Facts that get inserted into the working memory at start are (missing fields see below):

- Time t: counter for the time/round
- Pool(id==0,...): one common pool
- Institution(id==0,...): one institution associated with that pool
- Agent(name=="elf"+i,...): all agents $a_i \in \mathcal{M}$
- Agent(name=="outelf"+i,...): all agents $a_i \in \mathcal{A} \setminus \mathcal{M}$

Table B.1. Initialisation These are the values that are used to initialise the first round of a testbed run.

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int pools</td>
<td>1</td>
</tr>
<tr>
<td>int institutions</td>
<td>1</td>
</tr>
<tr>
<td>int agents</td>
<td>100</td>
</tr>
<tr>
<td>int numCheat</td>
<td>0</td>
</tr>
<tr>
<td>int outAgents</td>
<td>20</td>
</tr>
<tr>
<td>int outNumCheat</td>
<td>0</td>
</tr>
<tr>
<td>double greedMax</td>
<td>0.2</td>
</tr>
<tr>
<td>int finishTime</td>
<td>500</td>
</tr>
</tbody>
</table>
B. Class field values and experimental parameters

Table B.2.: **Common Pool** These values initialise a common-pool object.

<table>
<thead>
<tr>
<th>field name</th>
<th>field value</th>
</tr>
</thead>
<tbody>
<tr>
<td>final int id</td>
<td>0</td>
</tr>
<tr>
<td>double initialLevel</td>
<td>2<em>standardRequest</em>agents</td>
</tr>
<tr>
<td>final double maxLevel</td>
<td>2<em>standardRequest</em>agents</td>
</tr>
<tr>
<td>double outAppropriationFrequency</td>
<td>0.1</td>
</tr>
<tr>
<td>double outImproveFrequency</td>
<td>0.1</td>
</tr>
<tr>
<td>RefillScheme refScheme</td>
<td>CUSTOM</td>
</tr>
<tr>
<td>boolean unintentionalError</td>
<td>false</td>
</tr>
<tr>
<td>final double noisePercentage</td>
<td>0.05</td>
</tr>
<tr>
<td>final double noiseLevel</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Table B.3.: **Institution** These values initialise an institution object.

<table>
<thead>
<tr>
<th>field name</th>
<th>field value</th>
</tr>
</thead>
<tbody>
<tr>
<td>final int id</td>
<td>0</td>
</tr>
<tr>
<td>final int initialAgents</td>
<td>agents</td>
</tr>
<tr>
<td>CommonPool pool</td>
<td>pool0</td>
</tr>
<tr>
<td>final boolean principle1</td>
<td>true</td>
</tr>
<tr>
<td>boolean principle2</td>
<td>false</td>
</tr>
<tr>
<td>boolean principle3</td>
<td>false</td>
</tr>
<tr>
<td>boolean principle4</td>
<td>false</td>
</tr>
<tr>
<td>boolean principle5</td>
<td>false</td>
</tr>
<tr>
<td>boolean principle6</td>
<td>false</td>
</tr>
<tr>
<td>final double monitoringLevel</td>
<td>0.1</td>
</tr>
<tr>
<td>final double monitoringCost</td>
<td>standardRequest</td>
</tr>
<tr>
<td>final double outMonitoringLevel</td>
<td>0.1</td>
</tr>
<tr>
<td>final double outMonitoringCost</td>
<td>0.1*standardRequest</td>
</tr>
<tr>
<td>final int appealtime</td>
<td>30</td>
</tr>
<tr>
<td>int samplingrate</td>
<td>50</td>
</tr>
<tr>
<td>int maxSanctionLevel</td>
<td>3</td>
</tr>
<tr>
<td>int excludeTime</td>
<td>5</td>
</tr>
<tr>
<td>double applyPercentage</td>
<td>0.1</td>
</tr>
<tr>
<td>boolean voteHead</td>
<td>false</td>
</tr>
<tr>
<td>boolean voteRaMethod</td>
<td>false</td>
</tr>
<tr>
<td>Phase state</td>
<td>Phase.CFV</td>
</tr>
<tr>
<td>int round</td>
<td>0</td>
</tr>
</tbody>
</table>
Table B.4: **Member** Note that either of the two fields `initialCompliancyDegree` is chosen, depending on whether \( i < \text{numCheat}(2) \) holds for the member counter \( i \in \{0, \ldots, \text{agents} - 1 \} \), or not (1).

<table>
<thead>
<tr>
<th>field name</th>
<th>field value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>final String name</code></td>
<td>&quot;elf&quot;+i</td>
</tr>
<tr>
<td><code>final int pool</code></td>
<td>0</td>
</tr>
<tr>
<td><code>final double initialCompliancyDegree (1)</code></td>
<td>1.0</td>
</tr>
<tr>
<td><code>final double initialCompliancyDegree (2)</code></td>
<td>1.0 + sim.greedMax * rand()</td>
</tr>
<tr>
<td><code>double compliancyDegree</code></td>
<td>initialCompliancyDegree</td>
</tr>
<tr>
<td><code>final double standardRequest</code></td>
<td>50.0</td>
</tr>
<tr>
<td><code>boolean active</code></td>
<td>true</td>
</tr>
<tr>
<td><code>int institutionId</code></td>
<td>0</td>
</tr>
<tr>
<td><code>double noRequestPercentage</code></td>
<td>0.1</td>
</tr>
<tr>
<td><code>int sanctionLevel</code></td>
<td>0</td>
</tr>
<tr>
<td><code>double changeBehaviourPercentage</code></td>
<td>0.3</td>
</tr>
<tr>
<td><code>double improveBehaviour</code></td>
<td>0.5</td>
</tr>
<tr>
<td><code>double preferredRequest</code></td>
<td><code>standardRequest*compliancyDegree * (1 + (0.2 * rand() - 0.1))</code></td>
</tr>
</tbody>
</table>

Table B.5: **NonMember** Depending on whether \( i < \text{outNumCheat}(2) \) holds for the non-member counter \( i \in \{0, \ldots, \text{outAgents} - 1 \} \), or not (1), either of the two fields `initialCompliancyDegree` is chosen.

<table>
<thead>
<tr>
<th>field name</th>
<th>field value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>final String name</code></td>
<td>&quot;outelf&quot;+i</td>
</tr>
<tr>
<td><code>final int pool</code></td>
<td>0</td>
</tr>
<tr>
<td><code>final double initialCompliancyDegree (1)</code></td>
<td>1.0</td>
</tr>
<tr>
<td><code>final double initialCompliancyDegree (2)</code></td>
<td>1.0 + sim.greedMax * rand()</td>
</tr>
<tr>
<td><code>double compliancyDegree</code></td>
<td>initialCompliancyDegree</td>
</tr>
<tr>
<td><code>final double standardRequest</code></td>
<td>50.0</td>
</tr>
<tr>
<td><code>boolean active</code></td>
<td>true</td>
</tr>
</tbody>
</table>
B. Class field values and experimental parameters

Table B.6 mentions how the parameters of individual runs are set up for experimentation, referred to by the label they have in the corresponding figure, see Section 4.4 and Appendix C. Only parameters that deviate from the default are given, and \( x \) refers to a particular refill scheme mentioned in Table B.7. The number in the first column each refers to the set of experiments:

1. Existence and Management Principles
2. Protection Principles
3. Alternatives (‘good’ population)
4. Alternatives (‘bad’ population)

For each refill scheme, the same measurements are taken for all runs, which are resource level per active cluster, active agents per active cluster, and active clusters. When comparing the refill schemes the measurements taken are remaining agents per remaining cluster, remaining clusters and appropriated resource per cluster.

### Table B.6.: Experimental parameters of individual runs

<table>
<thead>
<tr>
<th>Brute Facts</th>
<th>Institutional Facts</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>outNumCheat=10, refScheme=X</td>
<td>pr2=true, outMonitoringLevel=0.01</td>
</tr>
<tr>
<td>2</td>
<td>numCheat=50, outNumCheat=10, unintentError=true, refScheme=X</td>
<td>pr2&amp;pr3=true</td>
</tr>
<tr>
<td>3</td>
<td>outNumCheat=10, refScheme=X</td>
<td>pr2&amp;pr3=true, monitoringLevel=0.01</td>
</tr>
<tr>
<td>4</td>
<td>numCheat=50, outNumCheat=10, refScheme=X</td>
<td>pr2&amp;pr3=true</td>
</tr>
</tbody>
</table>

### Table B.7.: Figures for sets of runs according to refill scheme

<table>
<thead>
<tr>
<th>CUSTOM</th>
<th>HIGH</th>
<th>MODERATE</th>
<th>LOW</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.6</td>
<td>C.1</td>
<td>C.5</td>
<td>C.9</td>
</tr>
<tr>
<td>2</td>
<td>4.7</td>
<td>C.2</td>
<td>C.6</td>
<td>C.9</td>
</tr>
<tr>
<td>3</td>
<td>4.8</td>
<td>C.3</td>
<td>C.7</td>
<td>C.11</td>
</tr>
<tr>
<td>4</td>
<td>4.9</td>
<td>C.4</td>
<td>C.8</td>
<td>C.12</td>
</tr>
</tbody>
</table>
C. Supporting figures

C.1. Refill scheme: HIGH

Figure C.1.: $P_1$, $P_2$ and $P_3$, refill scheme HIGH: all *members* compliant, 50% of *non-members* noncompliant, no unintentional violation.

Figure C.2.: $P_4$, $P_5$ and $P_6$, refill scheme HIGH: 50% noncompliant *members* and *non-members*, with unintentional violations.
C. Supporting figures

Figure C.3.: P4 ‘good’ population, refill scheme HIGH: all member compliant and 50% noncompliant non-members, no unintentional violation.

Figure C.4.: P4 ‘bad’ population, refill scheme HIGH: 50% noncompliant members and non-members, no unintentional violation.

C.2. Refill scheme: MODERATE
C.2. Refill scheme: MODERATE

Figure C.5.: P1, P2 and P3, refill scheme MODERATE: all members compliant, 50% of non-members noncompliant, no unintentional violation.

Figure C.6.: P4, P5 and P6, refill scheme MODERATE: 50% noncompliant members and non-members, with unintentional violations.

Figure C.7.: P4 ‘good’ population, refill scheme MODERATE: all member compliant and 50% noncompliant non-members, no unintentional violation.
C. Supporting figures

Figure C.8.: \( P_4 \) ‘bad’ population, refill scheme MODERATE: 50% noncompliant members and non-members, no unintentional violation.

C.3. Refill scheme: LOW

Figure C.9.: \( P_1, P_2 \) and \( P_3 \), refill scheme LOW: all members compliant, 50% of non-members noncompliant, no unintentional violation.
C.3. Refill scheme: LOW

Figure C.10.: $P_4$, $P_5$ and $P_6$, refill scheme LOW: 50% noncompliant members and non-members, with unintentional violations.

Figure C.11.: $P_4$ ‘good’ population, refill scheme LOW: all members compliant and 50% noncompliant non-members, no unintentional violation.

Figure C.12.: $P_4$ ‘bad’ population, refill scheme LOW: 50% noncompliant members and non-members, no unintentional violation.
D. Testbed extension

D.1. Member :: vote

```java
public Vote vote(Institution i, CommonPool pool, String ballot) {
    if (ballot.equals("head")) {
        Member vote;
        List<Member> helplist = new ArrayList<Member>();
        if (!droppedHeads.contains(i.getInstHead().getName())){
            vote = i.getInstHead();
        } else {
            for (Object o : i.getInstMembers()){
                Member m = (Member) o;
                if (!droppedHeads.contains(m.getName())){
                    helplist.add(m);
                }
            }
            if (helplist.isEmpty()){
                vote = null;
            } else {
                vote = (Member) helplist.get(Random.randomInt(helplist.size()));
            }
        }
    return Vote.voteHead(vote);
    } else {
        return null;
    }
}
```

D.2. Member :: sat evaluation

```java
rule "Update satisfaction if head appointed externally"
```
D. Testbed extension

```plaintext
ruleflow-group "vote"
no-loop
when
    Institution($iid:id, $r:round, pr7==false)
    HeadChange(inst==$iid, round==$r)
    $m: Member(instId == $iid)
then
    modify($m){
    | setSatisfaction( $m.getInitialSat() );
}
end

rule "Update satisfaction if head elected"
ruleflow-group "vote"
no-loop
when
    Institution($iid:id, $r:round, pr7==true)
    HeadChange(inst==$iid, round==$r)
    Declared(inst==$iid, ballo =="head", round==$r, $res:result)
    $m: Member(instId == $iid)
    Vote(voter==$m.name, ballot=="head", round==$r, $v:value)
then
    double sat = $m.getInitialSat();
    if ($v!=null && $v==$res){
        sat = 1.0;
    }
    modify($m){
    | setSatisfaction(sat);
}
end
```

D.3. Member :: sample

```plaintext
rule "Collect members for sample"
ruleflow-group "appropriate"
when
    $i: Institution($iid:id, $r:round)
    $hd: Head(instId==$iid)
    $m: Member(instId==$iid, $n:name)
    $acNames: List () from accumulate( Member(instId==$iid, $n:name, active==true), collectList($n))
    not SampleList(agent==$n, round==$r, inst==$iid)
```
D.4. Member :: judge

```java
then
  insert( new SampleList($n, $r, $m.sample($acNames), $iid));
end

public List<String> sample(List<String> acNames){
  List<String> copyNames = new ArrayList<String>();
  for(String s : acNames){
    copyNames.add(s);
  }
  if(copyNames.size() > judgeSize){
    Collections.shuffle(copyNames);
    copyNames = copyNames.subList(0, judgeSize);
  }
  return copyNames;
}
```

D.4. Member :: judge

```java
rule "Judge head allocation"
ruleflow-group "appropriate"
when
  $i: Institution($iid:id, $p:pool, $r: round)
  $hd: Head(instId==$iid)
  $m: Member(instId==$iid, $n: name)
  $jList: SampleList(agent==$n, round==$r, $acNames:list)
  $demands: List() from collect( Demand(pool==$p.id, round==$r, agent memberOf $acNames) )
  $allocations: List() from collect( Allocation(pool==$p.id, round==$r, agent memberOf $acNames) )
  not TaskExecuted(inst==$i, task="judgeHead", agent==$n, round==$r)
then
  modify($m) {
    judgeHead($i, $p, $hd, $demands, $allocations);
  }
  insert( new TaskExecuted($i, "judgeHead", $n, $r) );
end

public void judgeHead(Institution i, CommonPool p, Head $hd, List<Demand> demands, List<Allocation> allocs){
  Norm justiceNorm;
  if (p.getResourceLevel() < 0.75*i.getMaxLevel()/compliancyDegree) {
```
D. Testbed extension

```java
justiceNorm = crisisNorm;
}
else {
  justiceNorm = regularNorm;
}
int helpalloc = 0; // possible allocations
if (!i.isPrinciple8()) {
  helpalloc = (int) (p.getStartResourceLevel() / standardRequest) * ((double) judgeSize) / i.getInitialAgents();
} else if (i.getActiveMemberCount() != 0) {
  helpalloc = (int) (p.getStartResourceLevel() / standardRequest) * ((double) judgeSize) / i.getActiveMemberCount();
}
int meritoriousDem = 0;
int needyDem = 0;
int meritoriousAll = 0;
int needyAll = 0;
for (Demand d : demands) {
  if (d.getProfile() == Profile.NEEDY) {
    needyDem ++;
    for (Allocation a : allocs) {
      if (d.getAgent() == a.getAgent()) {
        needyAll ++;
      }
    }
  } else { // profile == MERITORIOUS
    meritoriousDem ++;
    for (Allocation a : allocs) {
      if (d.getAgent() == a.getAgent()) {
        meritoriousAll ++;
      }
    }
  }
}
switch (justiceNorm) { // demands become simulated allocations
case EQUITY:
  if (helpalloc > meritoriousDem) {
    if (helpalloc - meritoriousDem < needyDem) {
      needyDem = helpalloc - meritoriousDem;
    } else {
      meritoriousDem = helpalloc;
      needyDem = 0;
    }
```
D.5. Member :: leave

```java
rule "Members leave due to dissatisfaction"
```
D. Testbed extension

```
2  ruleflow-group "exclude"
3  salience 10
4  when
5     | $i: Institution($iid:id, $r: round, pr8==false)
6     | $hd : Head(instId==$iid)
7     | $m: Member(name!=$hd.name, instId==$iid, getSatisfaction() < getLeaveSat())
8     then
9     | retract($m);
10    | insert(new NonMember($m,0));
11 end
```

D.6. Member :: change

```
1  rule "Members change subcluster"
2  ruleflow-group "exclude"
3  salience 10
4  when
5     | $i: Institution($iid:id, $r:round, pr8==true)
6     | $hd : Head(instId==$iid)
7     | $m: Member(name!=$hd.name, instId==$iid, getSatisfaction() < getLeaveSat())
8     | $insts : List() from collect (Institution(round==$r, instId!=$iid))
9     then
10    | Institution ninst = (Institution) $insts.get(Random.randomInt($insts.size()));
11    | modify($m){
12        | setInstitutionId(ninst.getId()),
13        | setPool(ninst.getPool().getId()),
14        | setSatisfaction($m.getInitialSat())
15    } end
```

D.7. Head :: declare

```
1  rule "Declare winner head"
2  ruleflow-group "vote"
3  when
4     | $i: Institution($iid:id, $r:round, pr3==true)
```
D.8. Head :: update

```java
Head($headId:id, instId==$iid)
$vc: VoteCount(ballot="head", round==r, inst==$iid)
not (Declared(inst==$iid, ballot="head", round==r))
then
Map<Integer, Integer> tally = $vc.result;
if(tally.containsKey(null) && tally.get(null) > $i.instMembers.size()/2) {
    retract($i);
} else {
    tally.remove(null);
    if(tally.containsKey($headId) && tally.get($headId)>$i.instMembers.size()/2) {
        Integer hd = (Integer) $headId;
    } else {
        tally.remove($headId);
        Integer largestVal=null;
        List<Entry<Integer, Integer>> largestList = new ArrayList<Entry<Integer, Integer>>();
        for (Entry<Integer, Integer> j : tally.entrySet()) {
            if(largestVal==null || largestVal<j.getValue()) {
                largestVal=j.getValue();
                largestList.clear();
                largestList.add(j);
            } else if(largestVal==j.getValue()) {
                largestList.add(j);
            }
        }
        Integer hd = largestList.get(Random.randomInt(largestList.size())).getKey();
    }
    insert(new Declared($iid, $vc.ballot, $r, hd));
} end
```

D.8. Head :: update

```java
rule "Update winner head"
| ruleflow-group "vote" |
| when |
| Institution($iid:id, $r:round, pr7==true) |
| Declared(inst==$iid, ballot="head", round==r, $res:result) |
| $h: Head(instId==$iid, $hid:id, $hid!=:res) |
```
D. Testbed extension

```plaintext
7 | $m$: Member(instId==$iid, $mid:id, $mid==$res)
8 | not HeadChange(round=$r, inst=$iid)
9 | then
10 | retract($h);
11 | insert(new Member($h));
12 | retract($m);
13 | insert(new Head($m));
14 | insert(new HeadChange($m.getName(), $r, $iid));
end
```

D.9. Head :: allocate

```plaintext
1  public Set<Allocation> allocate(Institution i, CommonPool p, List<Demand> demands) {
2   Set<Allocation> allocations = new HashSet<Allocation>();
3   double level = p.getResourceLevel() - monitoring*i.getMonitoringCost() - outMonitoring*i.getOutMonitoringCost();
4   if (level < 0){
5     return allocations;
6   }
7   Norm justiceNorm;
8   LinkedList<Demand> demandQueue = new LinkedList<Demand>();
9   if (level < 0.75*i.getMaxLevel()/compliancyDegree) {
10      justiceNorm = crisisNorm;
11   } else {
12      justiceNorm = regularNorm;
13   }
14   switch(justiceNorm){
15     case EQUITY:
16        Collections.shuffle(demands);  
17        for (Demand d : demands){
18           if (d.getProfile()==Profile.MERITORIOUS){
19              demandQueue.addFirst(d);
20           } else {
21              demandQueue.addLast(d);
22           }
23        }
24        break;
25     case EQUALITY:
26        Collections.shuffle(demands);  
27        for (Demand d : demands){
28           demandQueue.add(d);
29        }
30   }
31 }
```
D.10. Gatekeeper :: split

```java
29  } } 
30  break;
31  case NEED:
32  Collections.shuffle(demands);
33  for (Demand d : demands){
34      if (d.getProfile()==Profile.NEEDY){
35          demandQueue.addFirst(d);
36      } else {
37          demandQueue.addLast(d);
38      }
39  }
40  } // end switch
41  int qSize = demandQueue.size();
42  int qCounter = 0;
43  while (!demandQueue.isEmpty() && qCounter < qSize ) {
44      qCounter ++;
45      if (level >= demandQueue.peek().getQuantity()) {
46          Demand d = demandQueue.poll();
47          allocations.add(new Allocation(d.getAgent(),
48                          d.getQuantity(), p.getId()));
49          level -= d.getQuantity();
50      } else {
51          break;
52      }
53  }
54  return allocations;
55 }
```

D.10. Gatekeeper :: split

```mermaid
rule "Adjust maximum pool level - any refill scheme"
salience -5
ruleflow-group "init"
when
| $t: IntegerTime() |
| $p: CommonPool($t.intValue()>getLastFilled()) |
| $i: Institution(pool==$p, pr8==true) |
| Gatekeeper($sr:getStandardRequest()) |
| not LevelAdjusted(inst==$i, round==$t) |
then
modify($p){
| setMaxLevel(2*$sr*$i.getActiveMemberCount()) |
```
D. Testbed extension

```java
| 13  | } |
| 14  | insert(new LevelAdjusted($i, $t)); |
| 15  | end |
```

D.11. Agency :: appoint

```java
rule "External appointment of head"
ruleflow-group "vote"
when
| $i: Institution($iid:id, $p:pool, $r:round, ⇧ round% samplingrateHead==0, pr7==false) |
| $h: Head(instId==$iid) |
| $mL: List(size>0) from collect(Member(instId==$iid, ⇧ name!=$h.name)) |
| not (exists HeadChange(inst==$iid, round==$r)) |
then |
| retract($h); |
| insert(new Member($h)); |
| Member m = (Member) $mL.get(Random.randomInt($mL.size())); |
| retract($m); |
| insert(new Head(m)); |
| insert(new HeadChange(m.getName(), $r, $iid)); |
end
```

D.12. Additional or changed field values

Table D.1 shows the changes or additions that have been made to the testbed values and field variables for the extended version. Again, some of the member field values depend on the variables from the initialisation phase. For example, the distribution norms regularNorm and crisisNorm are chosen according to the values of equityPct, equalityPct and normTransition. Two random values rand1 and rand2 are chosen to select one of the three norms equity, equality and need as follows:

- regularNorm = equity, if rand1 ∈ [0, equalityPct)
- regular = equality, if rand1 ∈ [equalityPct, equalityPct+equityPct)
- regularNorm = need, if rand1 ∈ [equalityPct+equityPct, 1]
- crisisNorm = regularNorm, unless rand2 < normTransition, then
D.12. Additional or changed field values

The profile of an agent depends on a third random variable \( rand_3 \):

- profile = meritorious, if \( rand_3 < profilePct \)
- profile = needy, if \( rand_3 \geq profilePct \)

When \( P8 \) is used for experimentation, 3 clusters are initialised and the parameters that change are:

- int pools = 3
- int institutions = 3, where
  - int agents = 100 for Institution(id==0)
  - int agents = 0 for Institution(id==1 or id==2)

Table D.1.: Extension field values

<table>
<thead>
<tr>
<th>Initialisation</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int outAgents</td>
<td>0</td>
</tr>
<tr>
<td>double equityPct</td>
<td>0.5</td>
</tr>
<tr>
<td>double equalityPct</td>
<td>0.33</td>
</tr>
<tr>
<td>double normTransition</td>
<td>0.75</td>
</tr>
<tr>
<td>double profilePct</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Institution</th>
<th>field value</th>
</tr>
</thead>
<tbody>
<tr>
<td>final boolean principle2</td>
<td>true</td>
</tr>
<tr>
<td>final boolean principle3</td>
<td>true</td>
</tr>
<tr>
<td>boolean principle7</td>
<td>false</td>
</tr>
<tr>
<td>boolean principle8</td>
<td>false</td>
</tr>
<tr>
<td>final double outMonitoringLevel</td>
<td>0.0</td>
</tr>
<tr>
<td>final double outMonitoringCost</td>
<td>0.0</td>
</tr>
<tr>
<td>int samplingRate</td>
<td>500</td>
</tr>
</tbody>
</table>
D. Testbed extension

Member

<table>
<thead>
<tr>
<th>field name</th>
<th>field value</th>
</tr>
</thead>
<tbody>
<tr>
<td>final int pool</td>
<td>0</td>
</tr>
<tr>
<td>final double initialComplianceDegree</td>
<td>1.0</td>
</tr>
<tr>
<td>Norm regularNorm</td>
<td>see p. 218</td>
</tr>
<tr>
<td>Norm crisisNorm</td>
<td>see p. 218</td>
</tr>
<tr>
<td>Profile profile</td>
<td>see p. 218</td>
</tr>
<tr>
<td>int judgeSize</td>
<td>10</td>
</tr>
<tr>
<td>int judgeTolerance</td>
<td>2</td>
</tr>
<tr>
<td>double initialSat</td>
<td>0.8</td>
</tr>
<tr>
<td>double leaveSat</td>
<td>0.4</td>
</tr>
<tr>
<td>double increaseFactor</td>
<td>0.1</td>
</tr>
<tr>
<td>double decreaseFactor</td>
<td>0.25</td>
</tr>
<tr>
<td>double satisfaction</td>
<td>initialSat</td>
</tr>
<tr>
<td>List&lt;String&gt; droppedHeads</td>
<td>empty</td>
</tr>
</tbody>
</table>

Agency

<table>
<thead>
<tr>
<th>field name</th>
<th>field value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int samplingRateHead</td>
<td>500</td>
</tr>
</tbody>
</table>

D.13. Control loop for CPR testbed extension
D.13. Control loop for CPR testbed extension

Algorithm 5: Control loop for CPR testbed extension

\[ \exists P \in \epsilon \subseteq E: P \text{ is pool}; \]
\[ \exists I \in L: I \text{ is institution with } P_1 \leftarrow \text{true} \land P_2 \leftarrow \text{true} \land P_3 \leftarrow \text{true}; \]
\[ \forall m \in M \subseteq A: \text{role of } m \text{ is member } \land \text{satisfaction } s_m = 0.8; \]
\[ \exists h \in \mathcal{M}: \text{role of } h \text{ is head}; \quad \exists g_k \in \mathcal{M}: \text{role of } g_k \text{ is gatekeeper}; \]
\[ \text{if } P_8: \exists 3 \text{ subclusters } C_1, C_2 \text{ and } C_3 \text{ with members } M_1 \leftarrow M, M_2 \leftarrow \emptyset, M_3 \leftarrow \emptyset; \]
\[ r \leftarrow 0; \quad P \leftarrow P_{\text{max}}; \]
\[ \textbf{while } (M \neq \emptyset) \land (P \geq 0) \land (t < \text{FinishTime}) \land (\exists h) \textbf{ do} \]
\[ P \leftarrow \min (P_{\text{max}}, P + P_{\text{refill}}); \quad // \text{refill common pool} \]
\[ \text{assign missing roles;} \quad // \text{i.e. gatekeeper} \]
\[ \text{if } P_8 \textbf{ then} \]
\[ g_k \text{ splits resource for subclusters } C^i: P^i \leftarrow |M^i| / |M| \cdot P; \quad // i = 1, 2, 3 \]
\[ h \text{ calls for vote on head;} \quad // \forall C^i \text{ if applicable} \]
\[ \forall m \in M: m \text{ votes;} \]
\[ h \text{ counts votes and declares head;} \quad // \forall C^i \text{ if applicable} \]
\[ \text{if } P_7 \textbf{ then} \]
\[ h \leftarrow \text{head;} \quad // \text{updated according to vote, } \forall C^i \text{ if applicable} \]
\[ \text{else if } r \mod \text{rate} \equiv 0 \textbf{ then} \]
\[ h \leftarrow \text{external appointment;} \quad // \forall C^i \text{ if applicable} \]
\[ \forall m \in M: m \text{ updates } s_m \text{ according to } h; \]
\[ \forall m \in M: m \text{ places demand;} \]
\[ h \text{ allocates resource to all } m: \quad // \text{using internal distribution norm, } \forall C^i \text{ if applicable} \]
\[ \forall m \in M: m \text{ appropriates resource } R_m; \]
\[ P \leftarrow P - \sum_m R_m; \]
\[ \text{for } m \in M \textbf{ do} \]
\[ m \text{ samples allocations;} \quad // \text{demand–allocation pairs} \]
\[ m \text{ judges fairness of head’s allocation procedure;} \]
\[ \text{if } m \text{ judges head unfair } \textbf{ then} \]
\[ s_m \leftarrow s_m - s_m \cdot 0.25; \]
\[ \text{else} \]
\[ s_m \leftarrow s_m + (1 - s_m) \cdot 0.1; \]
\[ \text{if } P_8 \land s_m < 0.4 \textbf{ then} \]
\[ m \text{ joins other subcluster;} \]
\[ \text{else if } !P_8 \land s_m < 0.4 \land ‘leave’ \text{ case then} \]
\[ m \text{ leaves;} \]
\[ \text{else} \]
\[ m \text{ stays;} \quad // ‘no leave’ \text{ case } \lor s_m \geq 0.4 \]
\[ r \leftarrow r + 1; \quad // \text{increment round} \]
E. Supporting material (extension)

E.1. End distribution of single trials

Figure E.1.: Subcluster change of capitalist population ($Pr7$ and $Pr8$, 27 heads)

Table E.1.: End distribution of capitalist population ($Pr7$ and $Pr8$)

<table>
<thead>
<tr>
<th>Subcluster</th>
<th>equityStay</th>
<th>equityChange</th>
<th>equalityStay</th>
<th>equalityChange</th>
<th>need</th>
</tr>
</thead>
<tbody>
<tr>
<td>subcluster 1</td>
<td></td>
<td></td>
<td>1</td>
<td>27</td>
<td>19</td>
</tr>
<tr>
<td>subcluster 2</td>
<td></td>
<td>26</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subcluster 3</td>
<td>16</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>16</td>
<td>28</td>
<td>10</td>
<td>27</td>
<td>19</td>
</tr>
</tbody>
</table>
E. Supporting material (extension)

Figure E.2.: Subcluster change of capitalist population \((rate = 15 \text{ and } Pr8, 82 \text{ head changes})\)

Table E.2.: End distribution of capitalist population \((rate = 15 \text{ and } Pr8)\)

<table>
<thead>
<tr>
<th>subcluster 1</th>
<th>equityStay</th>
<th>equityChange</th>
<th>equalityStay</th>
<th>equalityChange</th>
<th>need</th>
</tr>
</thead>
<tbody>
<tr>
<td>subcluster 2</td>
<td>1</td>
<td>20</td>
<td>7</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>subcluster 3</td>
<td>.</td>
<td>5</td>
<td>1</td>
<td>19</td>
<td>11</td>
</tr>
<tr>
<td>total</td>
<td>19</td>
<td>28</td>
<td>8</td>
<td>30</td>
<td>15</td>
</tr>
</tbody>
</table>

E.2. Average end distribution

(a) End distribution --- Socialist (HIGH)

(b) End distribution --- Capitalist (HIGH)
E.3. Purity on average and for individual trial

Figure E.3.: Distribution of remaining agents by norm for high refill scheme

![Distribution of remaining agents by norm for high refill scheme](image1)

Figure E.4.: Distribution of remaining agents by norm for moderate refill scheme

![Distribution of remaining agents by norm for moderate refill scheme](image2)

E.3. Purity on average and for individual trial

Figure E.5.: Purity by distribution norm for high refill scheme

![Purity by distribution norm for high refill scheme](image3)
Figure E.6.: Purity by distribution norm for moderate refill scheme

Table E.3.: End distribution of socialist population (Pr7 and Pr8), moderate/high refill scheme

<table>
<thead>
<tr>
<th></th>
<th>equityStay</th>
<th>equityChange</th>
<th>equalityStay</th>
<th>equalityChange</th>
<th>need</th>
</tr>
</thead>
<tbody>
<tr>
<td>subcluster 1</td>
<td>8</td>
<td>9</td>
<td>15</td>
<td>17</td>
<td>51</td>
</tr>
<tr>
<td>subcluster 2</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>subcluster 3</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>total</td>
<td>8</td>
<td>9</td>
<td>15</td>
<td>17</td>
<td>51</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>equityStay</th>
<th>equityChange</th>
<th>equalityStay</th>
<th>equalityChange</th>
<th>need</th>
</tr>
</thead>
<tbody>
<tr>
<td>subcluster 1</td>
<td>5</td>
<td>7</td>
<td>15</td>
<td>16</td>
<td>44</td>
</tr>
<tr>
<td>subcluster 2</td>
<td>6</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>subcluster 3</td>
<td>5</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>2</td>
</tr>
<tr>
<td>total</td>
<td>16</td>
<td>7</td>
<td>15</td>
<td>16</td>
<td>46</td>
</tr>
</tbody>
</table>

Table E.4.: End distribution of socialist population (rate = 15 and Pr8), low refill scheme

<table>
<thead>
<tr>
<th></th>
<th>equityStay</th>
<th>equityChange</th>
<th>equalityStay</th>
<th>equalityChange</th>
<th>need</th>
</tr>
</thead>
<tbody>
<tr>
<td>subcluster 1</td>
<td>4</td>
<td>4</td>
<td>12</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>subcluster 2</td>
<td>.</td>
<td>1</td>
<td>1</td>
<td>19</td>
<td>45</td>
</tr>
<tr>
<td>subcluster 3</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>total</td>
<td>9</td>
<td>9</td>
<td>18</td>
<td>19</td>
<td>45</td>
</tr>
</tbody>
</table>