Federal University of Santa Catarina (UFSC)

Graduate Program in Automation and Systems Engineering (PosAutomação)

Cleber Jorge Amaral

GoOrg: A model to automatically design organisations for Multi-Agent Systems

Florianópolis - SC - Brasil

January 2023

Cleber Jorge Amaral

GoOrg: A model to automatically design organisations for Multi-Agent Systems

Doctoral Thesis submitted to the Graduate Program in Automation and Systems Engineering (PosAutomação) of Federal University of Santa Catarina (UFSC) in partial fulfillment of the requirements for obtaining the doctoral degree (Doctor in Automation and Systems Engineering).
Supervisor: Jomi Fred Hübner
Co-supervisor: Stephen Cranefield

Florianópolis - SC - Brasil

January 2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Amaral, Cleber Jorge
GoOrg: A model to automatically design organisations for Multi-Agent Systems / Cleber Jorge Amaral ; orientador, Jomi Fred Hübner, coorientador, Stephen Cranefield, 2023.
140 p.

Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2023.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Design Organizacional. 3. Estrutura Organizacional. 4. Sistemas MultiAgentes Abertos. 5. Design Automático. I. Hübner, Jomi Fred. II. Cranefield, Stephen . III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. IV. Título.

Cleber Jorge Amaral

GoOrg: A model to automatically design organisations for Multi-Agent Systems

This work in the doctoral degree was evaluated and approved by a thesis defence committee composed of the following members:

Prof. Ricardo Azambuja Silveira, Dr.
INE/CTC/UFSC

Prof. Jaime Simão Sichman, Dr.
PCS/USP

Prof. Rafael Cauê Cardoso, Dr.
DCS/University of Aberdeen

We hereby affirm that this is the original and final version of the thesis which was evaluated and approved for obtaining the title of doctor in Automation and Systems Engineering.

Documento assinado digitalmente
JULIO ELIAS NORMEY RICO
Data: 29/01/2023 21:09:36-0300
CPF: ***.840.859-**
Verifique as assinaturas em https://v.ufsc.br

Coordinator of the Graduate Program in Automation and Systems Engineering (PosAutomação)

Documento assinado digitalmente
Jomi Fred Hubner
Data: 28/01/2023 06:29:47-0300
CPF: ***.334.769-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Jomi Fred Hübner, Dr.
Supervidor

Florianópolis - SC - Brasil
January 2023

This work is dedicated to my mother and father, Laci and Jorge who gave me the very first and most important lessons of my life. This work is also dedicated to my wife Sharlene, my daughter Sarah and my son Nathan who unconditionally supported me throughout a long journey in this research.

Acknowledgements

A special feeling of gratitude to Jomi, who from the very beginning of this research said the exact thing that I needed to hear. There can be no greater example of a kind and humane conduct that transcends the topics covered in this PhD thesis.

I also want to express my gratitude to Stephen for his once again excellent advice. I also want to acknowledge the help, inspiration, and motivation I received from my friends Timotheus Kampik, Cristina Helena, Michael Silva, Thomás Teixeira, Mozart Gonçalves, and Olivier Boissier.

Thank you to the professors, staffs and students of the PosAutomação program of UFSC and of the Information Science Department of the University of Otago for all the good lessons, talks and support. Also, many thanks to the publications' reviewers for their valuable contributions, and to Ricardo Silveira, Jaime Sichman and Rafael Cardoso for their insightful criticism in reviewing this thesis.

In closing, I would like to thank IFSC for approving and supporting this research, and the program CAPES-UFSC "Automação 4.0", the project AG-BR of Petrobras and the post-graduate program PosAutomação of UFSC for partially funding this research.

"I would rather have questions that cannot be answered
than answers that cannot be questioned."
(Richard Feynman)

Resumo

O design de organizações é uma tarefa complexa e trabalhosa. Isto é tema de estudos recentes que definem modelos para executar esta tarefa automaticamente. No entanto, os modelos existentes restringem o espaço de possíveis soluções requisitando definições prévias dos papéis organizacionais e geralmente não são adequados para o planejamento de recursos. Esta tese de doutorado apresenta o GoOrg, um modelo que utiliza como entrada um conjunto de objetivos e um conjunto de agentes disponíveis para gerar diferentes arranjos de estruturas organizacionais construídas a partir de posições organizacionais sintetizadas. As características mais distintas do GoOrg é o uso de posições organizacionais ao invés de papéis e que as posições são sintetizadas automaticamente no lugar de requisitar que o usuário as defina. Estas características facilitam a parametrização, a utilização no planejamento de recursos e as chances do modelo de encontrar soluções viáveis. Para avaliar o GoOrg, esta tese introduz duas especializações que estendem o modelo. Estas extensões definem processos e restrições, ilustrando como o GoOrg pode ser adequado para diferentes domínios. Entre os aspectos associados ao design de organizações, este trabalho apresenta uma comparação entre modelos de design organizacional e discute entradas, abstrações de agentes e procedimentos para adaptação de organizações durante seu ciclo de vida.

Palavras-chave: Design Organizacional. Estrutura Organizacional. Estrutura Social. Sistemas MultiAgentes Abertos. Design Automático.

Resumo Expandido

Introdução

Agentes são entidades autônomas de software que normalmente cooperam com outros, formando Sistemas Multi-Agentes (SMAs). Para que seus objetivos comuns sejam alcançados, os agentes fazem parte de organizações que restringem ou incentivam certos comportamentos de agentes. Organizações são entidades independentes dos agentes, portanto, projetadas utilizando modelos, técnicas e ferramentas próprias. Uma organização é fundamentalmente definida por sua estrutura, ou seja, por posições organizacionais, suas relações e responsabilidades. Apesar da importância da estrutura e o quão trabalhoso é projetá-la, há poucos e limitados modelos de projeto automático de estruturas organizacionais. De fato, é desafiador elaborar um modelo de projeto automático de organizações que seja efetivo, adaptável e simples de parametrizar. O modelo GoOrg apresentado nesta tese busca vencer estas limitações. As entradas do modelo são os objetivos e os agentes disponíveis. As saídas são estruturas organizacionais cuja viabilidade é calculada além de outros atributos que ajudam a selecionar a organização mais adequada para um dado cenário. O GoOrg não utiliza o conceito de papéis organizacionais, como em outros modelos, ao invés disso utiliza posições que possuem relações uma-para-um com agentes. Esta característica facilita a verificação de viabilidade de uma organização, ou seja, se uma dada organização gerada poderá ser preenchida pelos agentes disponíveis. Além disso, a demanda por agentes pode ser verificada em tempo de projeto e a distribuição de objetivos pode ser estabelecida com maiores critérios pela ferramenta de projeto organizacional. Para demonstrar a aplicabilidade do modelo, esta tese apresenta também duas extensões do GoOrg, uma desenvolvida para um cenário industrial de produção de uma fábrica e outra para um cenário de sensores distribuídos para rastreamento de objetos.

Objetivos

O objetivo geral deste trabalho é desenvolver um modelo para geração automática de estruturas organizacionais para SMAs que seja independente de domínio. Os objetivos específicos são: (i) que o modelo seja extensível para diferentes domínios; (ii) que considere os agentes disponíveis verificando a viabilidade das estruturas organizacionais geradas e (iii) que permita que as estruturas geradas sejam ordenadas utilizando diferentes critérios para que a "melhor" estrutura, conforme as preferências do usuário, possa ser automaticamente selecionada.

Metodologia

Esta pesquisa utiliza a seguinte metodologia: (i) realização de revisão bibliográfica sobre geradores de organizações, destes as clássicas do campo de pesquisa da administração até as pesquisas de geradores automáticos de organizações de SMAs; (ii) definição do problema de pesquisa; (iii) proposição de uma solução; (iv) implementação da solução; e (v) avaliação da solução.

Resultados e Discussão

O GoOrg utiliza representações impessoais de agentes, desacoplando a organização dos agentes, facilitando a geração de mais candidatos e tornando o projeto de organizações mais flexível, uma vez que não se restringe às especificidades de agentes nomeados. A utilização de posições organizacionais ao invés de papéis organizacionais faz com que os recursos possam ser estimados em tempo de projeto

e eleva o controle do projeto sobre a distribuição de atividades a determinados agentes, elevando a relevância desta etapa. Apesar das vantagens que o modelo de posições com relações um-para-um traz, há a desvantagem de reduzir a flexibilidade de combinações em tempo de execução, o que pode elevar a necessidade de refazer o projeto organizacional quando o cenário é alterado. O GoOrg se destaca por sintetizar posições ao invés de requerer que os papéis organizacionais sejam definidos pelo usuário, reduzindo a complexidade da parametrização do modelo e reduzindo a influência dos vieses do usuário. No entanto, isso traz maior complexidade computacional. No que se refere as entradas do modelo, destaca-se que o GoOrg utiliza principalmente objetivos organizacionais e características que podem ser associadas a estes, sendo relativamente mais simples de se conceber comparado com outros modelos que requerem a definição dos comportamentos dos agentes. Como saída, o GoOrg gera um conjunto de estruturas quantificadas por atributos, facilitando a seleção automática da estrutura mais adequada para o cenário. Por fim, conforme demonstrado pelas extensões apresentadas, o GoOrg pode facilitar adaptações organizacionais durante a execução do SMA, suportando realocações simples de agentes, trocas de estrutura por outras previamente criadas ou, se necessário, reprojetando completamente as estruturas organizacionais. Para cada procedimento de adaptação foram discutidos custos associados como o curso de aquisição de novos agentes e o custo de sobrequalificação de agentes.

Considerações finais

O projeto organizacional tem sido refinado ao longo do tempo, desde os estudos no campo da administração aos modelos de projeto automático de organizações para SMAs. Os modelos de geração automática para SMAs utilizam papéis organizacionais que, apesar da flexibilidade em tempo de execução, dificultam a estimação de recursos e reduzem a relevância do projeto organizacional. Este estudo adotou posições organizacionais que compartilham as principais vantagens dos papéis, como o desacoplamento da organização de agentes nomeados, porém refletindo numericamente demandas, o que permite estimá-las e verificar antecipadamente a viabilidade de organizações. Este modelo foi avaliado do ponto de vista de duas extensões desenvolvidas para diferentes domínios. As extensões puderam gerar conjuntos de estruturas com atributos quantificados, realizando a seleção automática do "melhor" candidato conforme as preferências do usuário definidas no momento do projeto. Os processos de geração e seleção de organizações foram implementados como módulos distintos, o que permite dividir a complexidade, além de acelerar os resultados quando apenas um dos processos é requisitado. Porém, na realização do projeto organizacional o tempo de processamento não é o único aspecto a se considerar. Quando se planeja optar por procedimentos de adaptação mais leves como realocações e trocas de estruturas por outras previamente geradas, há também de se verificar outros custos como de aquisição e sobrequalificação de agentes. Como trabalhos futuros, planeja-se testar os procedimentos de adaptação em diferentes cenários, substituir o processo de busca de soluções por um algoritmo mais rápido, substituir o processo de vinculação de agentes com posições organizacionais por um algoritmo ótimo, implementar mais extensões para diferentes domínios e formas de estruturas e concluir a integração do modelo com uma ferramenta de desenvolvimento de SMAs.

Palavras-chave: Design Organizacional. Estrutura Organizacional. Estrutura Social. Sistemas MultiAgentes Abertos. Design Automático.

Abstract

The design of organisations is a complex and laborious task. It is the subject of recent studies, which define models to automatically perform this task. However, existing models constrain the space of possible solutions by requiring a priori definitions of organisational roles and usually are not suitable for planning resource use. This doctoral thesis presents GoOrg, a model that uses as input a set of goals and a set of available agents to generate different arrangements of organisational structures made up of synthesised organisational positions. The most distinguishing characteristics of GoOrg are the use of organisational positions instead of roles and that positions are automatically synthesised rather than required as a user-defined input. These characteristics facilitate the parametrisation, the use for resource planning and the chance of finding feasible solutions. To evaluate GoOrg, this thesis introduces two specialisations that extend the model. These extensions define processes and constraints, illustrating how GoOrg suits different domains. Among aspects that surround an organisation's design, this work presents a comparison of design models and discusses models input, agents' abstractions and procedures for adapting the organisation during its life cycle.

Keywords: Organisational Design. Organisational Structure. Social Structure. Open Multi-Agent Systems. Automated Design.

List of Figures

Figure 1 – Automated design of a *PCB Production* scenario. 27
Figure 2 – Some other candidate solutions for the *PCB Production* scenario. . . . 28

Figure 3 – Design model categories. 32

Figure 4 – The generic design process of GoOrg-based models. 43
Figure 5 – GoOrg model. 44

Figure 6 – GoOrg4Prod model. 47
Figure 7 – A given set of goals with associated *workloads* and *dataloads*. 48
Figure 8 – *GoOrg4Prod* organisational structure attributes in three dimensions. . . 50
Figure 9 – The four processes of *GoOrg4Prod*. 53
Figure 10 –The set G' of split goals for $\phi_g = 4$ and $\delta_g = 1000$. 53
Figure 11 –Supported transformations. 55
Figure 12 –Step by step of state search with all possible solutions for the given G. 58
Figure 13 –The available agents. 62
Figure 14 –The generated candidates quantified according to the user's preferences. 62
Figure 15 –The two best structures, which are not 100% feasible. 63
Figure 16 –Two feasible structures for the given example. 63
Figure 17 –Candidate #1646 64

Figure 18 –A MAOP approach for the DSN domain. 68
Figure 19 –GoOrg4DSN model. 70
Figure 20 –The three processes of *GoOrg4Prod*. 72
Figure 21 –A motivating scenario with four sectors, each one with 5 sensors. 75
Figure 22 –The set of goals in which no target is being detected. 76
Figure 23 –Candidate #1 (unique) when there is no target being detected. 76
Figure 24 –The set of goals in which one target is being detected. 76
Figure 25 –Candidate #1 for the scenario with 1 target being detected. 77
Figure 26 –Candidate #1 for the scenario in which three targets are being detected. 78

Figure 27 –GoOrg simplified class diagram. 81

Figure 28 –Feed Production Scenario. 88
Figure 29 –Some of the candidates for the Feed Production Scenario. 88
Figure 30 –A reallocation by replacing an agent by another. 90
Figure 31 –Matching kinds of agents and kinds of positions 91

Figure 32 – A required structure-switching due to a change in the user's preferences. 93
Figure 33 – Comparing the generality of candidates of the motivating scenario. . . 94
Figure 34 – Acquisition cost when switching between structures of the motivating scenario. 95
Figure 35 – A complete redesign after a change to the set of goals. 96
Figure 36 – Different forms of assigning goals to organisational members. 97
Figure 37 – Comparing a structure of roles and a structure of positions. 100
Figure 38 – A structure of instances of roles. 100
Figure 39 – Comparing examples of user-defined roles and synthesised positions structures. 102
Figure 40 – Comparing structures for the given scenario regarding all attributes. . . 103
Figure 41 – Comparing generators with roles and goals as input. 105

Figure 42 – Comparing the height of structures for the given scenario. 119
Figure 43 – Comparing the efficiency of structures for the given scenario. 120

Figure 44 – Synthesing roles, relationships, missions and norms from a GoOrg's output. 133

Figure 45 – The identification of the roles associated with positions. 137

List of Tables

Table 1 – Comparison among Automated Organisational Structure Generators. . 39

Table 2 – All the candidates for the given G containing just two goals. 59

Table 3 – Organisational structures generated for Feed Production with three goals. 85

Table 4 – Candidates #2, #3 and #4 for Feed Production scenario with three goals. 103

List of Abbreviations and Acronyms

CPU	Central Processing Unit
DF	Directory Facilitator
DSN	Distributed Sensor Networks
FIPA	Foundation for Intelligent Physical Agents
IDE	Integrated Development Environment
JVM	Java Virtual Machine
MAOP	Multi-Agent Oriented Programming
MAPC	Multi-Agent Programming Contest
MAS	Multi-Agent System
MaSE	Multiagent Systems Engineering
OSD	Organisation Self-Design
PCB	Printed Circuit Board
SADDE	Social Agents Design Driven by Equations
OMACS	Organization Model for Adaptive Computational Systems
DOMAP	Decentralised On-line Multi-Agent Planning
ODML	Organizational Design Modeling Language
KB-ORG	Knowledge-Based Organization Designer

Contents

1 INTRODUCTION 25
1.1 MOTIVATION 25
1.2 PROBLEM AND RESEARCH QUESTIONS 27
1.3 OBJECTIVES 29
1.4 CONTRIBUTION AND RELEVANCE 29
1.5 DOCUMENT STRUCTURE 30
2 ORGANISATION DESIGN MODELS 31
2.1 AUTOMATED ORGANISATIONAL DESIGN BY TASK PLANNING . . . 33
2.2 SELF-ORGANISATION APPROACHES 33
2.3 AUTOMATED ORGANISATIONAL STRUCTURE GENERATORS 35
2.3.1 Structure Generators' Background 35
2.3.2 State of the Art 37
2.3.3 Comparing Structure Generators 38
3 GOORG MODEL 43
3.1 GOORG ELEMENTS 44
3.2 ATTRIBUTES OF AN ORGANISATIONAL STRUCTURE 46
3.3 GOORG HIGHLIGHTED CHARACTERISTICS 46
4 GOORG4PROD: A SPECIALISATION FOR A FACTORY PRODUCTION LINE DOMAIN 47
4.1 GOORG4PROD ELEMENTS 48
4.2 GOORG4PROD ADDED ATTRIBUTES 50
4.3 GOORG4PROD PROCESSES 52
4.3.1 Preparing goals for assignments 53
4.3.2 Generating organisations 54
4.3.3 Binding agents and positions 59
4.3.4 Choosing organisations 60
4.3.5 Computational complexity 61
4.4 GOORG4PROD RESULTS 61
5 GOORG4DSN: A SPECIALISATION FOR THE DISTRIBUTED SENSORS NETWORK DOMAIN 67
5.1 GOORG4DSN ELEMENTS 69
5.2 GOORG4DSN ADDED ATTRIBUTES 71
5.3 GOORG4DSN PROCESSES 71
5.3.1 Generating organisations 72
5.3.2 Binding agents and positions 74

5.3.3 Choosing organisations . . . 74
5.3.4 Computational complexity . . . 74
5.4 GOORG4DSN RESULTS . . . 75
6 GOORG: IMPLEMENTATION . . . 79
6.1 TOOLS AND PROGRAMMING LANGUAGES . . . 79
6.2 GOORG IMPLEMENTATION ARCHITECTURE . . . 80
6.2.1 Executing GoOrg Implementation . . . 82
6.2.2 GoOrg Implementation Inputs . . . 83
6.2.3 GoOrg Implementation Outputs . . . 83
6.3 EXTENDING GOORG . . . 84
7 DISCUSSION . . . 87
7.1 ORGANISATIONAL ADAPTATION . . . 87
7.1.1 Reallocation . . . 90
7.1.2 Structure-switching . . . 92
7.1.3 Redesign . . . 96
7.2 ASSIGNING GOALS TO NAMED AGENTS, ROLES OR POSITIONS . . 96
7.3 PLANNING RESOURCES OF ORGANISATIONS . . . 98
7.4 SYNTHESISING POSITIONS INSTEAD OF REQUIRING USER-DEFINED ROLES . . . 100
7.5 USING GOALS AS INPUT INSTEAD OF ROLES AND BEHAVIOURS . 104
7.6 SUMMARY OF THIS DISCUSSION . . . 105
8 CONCLUSION . . . 107
BIBLIOGRAPHY . . . 109
APPENDIX A Comparing organisational attributes among candidates . . 119
APPENDIX B XML specification of Feed Production with four goals . . 121
APPENDIX C XML specification of DSN with 4x5 sensors and 3 targets 123
APPENDIX D XML specification and outputs for Feed Production with three goals . . . 127
APPENDIX E Improving GoOrg . . . 131
APPENDIX F Synthesizing organisational roles . . . 137
APPENDIX G Works developed during the PhD . . . 139

1 Introduction

In this chapter it is presented the motivation for this work, the problem and research questions that guided the investigations done, and the objectives and contributions of this thesis.

1.1 Motivation

A software entity known as an agent is one that has certain beliefs, goals, and capabilities and demonstrates some autonomy by choosing how it will carry out its goals (Bordini et al., 2007; Rahwan et al., 2015). A typical agent is a member of a Multi-Agent System (MAS) and is not operating in isolation in the environment in which it is located. To achieve the system's goals, it is necessary some mechanism to organise and coordinate such autonomous entities in a MAS (Hübner et al., 2002). Organisations are used to overcome this problem by promoting a coherent mechanism, which constrains and upholds acceptable behaviour of agents (Boissier et al., 2016; Hübner et al., 2002; Sierra et al., 2004). The organisation is an entity in which members adhere to a set of rules, have similar beliefs, and cooperate in achieving common goals. Because organisations are independent of agents, they are typically designed using distinct models, techniques and tools (Gasser, 2001). Additionally, the design of organisations is arguably as important as the design of the agents (Boissier et al., 2013; Cardoso and Ferrando, 2021).

Among the organisation's aspects to define, the organisational structure is crucial. It represents positions, displaying the hierarchy, relationships and responsibilities (Daft, 2009). An explicit structure allows members to know their position related to others, their authority relationships, and their commitment to organisational goals (Hatch, 1997). It supports agents' entrances and exits. It also promotes a method of task assignment (DeLoach, 2002).

Despite its significance and how time-consuming organisational design is, only a small number of research have focused on automatic generators of explicit organisational structures (DeLoach and Matson, 2004; Horling and Lesser, 2008; Sierra et al., 2004; Sims et al., 2008; So and Durfee, 1998). Although seminal, these works still have limitations to overcome. Indeed, determining a design model that is effective, adaptable, and simple to parametrise is challenging. Due to the specificity of each domain and the high number of variables that surround the design process, no existing model can be considered a decisive solution. This thesis presents GoOrg, a MAS organisational design model for au-

tomatically generating organisations, which addresses these issues. GoOrg considers that organisational structures are composed of organisational positions, which can be arranged into many shapes. It also considers that agents can occupy organisational positions and commit to their assigned goals. As input, GoOrg expects a set of goals and a set of available agents. Goals are used to synthesise organisational positions. Structures are created from the synthesised positions. The feasibility of each structure is evaluated using a given set of available agents. GoOrg outputs a list of organisational structures sorted according to the user's preferences.

The concept of roles is used in existing models for generating organisational structures for MAS (DeLoach and Matson, 2004; Horling and Lesser, 2008; Sierra et al., 2004; Sims et al., 2008; So and Durfee, 1998). Agents and roles have many-to-many relationships. Although intuitive, the concept of roles does not ease planning resource use and checking the organisation's feasibility. GoOrg uses organisational positions instead, which have one-to-one relationships with agents (Slade, 2018). Therefore, a position-based organisational structure reflects resource demands.

Existing models expect roles as input, which means that roles should be defined a priori by the user. This restricts the set of solutions, possibly making it infeasible to find a structure to be filled by the available agents. Indeed, many situations may require roles that the user cannot foresee, which for those models means that they cannot produce feasible organisations for the given inputs. In contrast to related works, in GoOrg, organisational positions are synthesised, relieving the user from the task of defining roles. Synthesized positions increase the likelihood of finding structures that can be filled by the available agents by generating a wider range of potential solutions. Besides, existing models require agent behaviour definitions. Behaviours are complex and time-consuming to define. Instead, GoOrg employs goals as inputs, which are typically simpler to define.

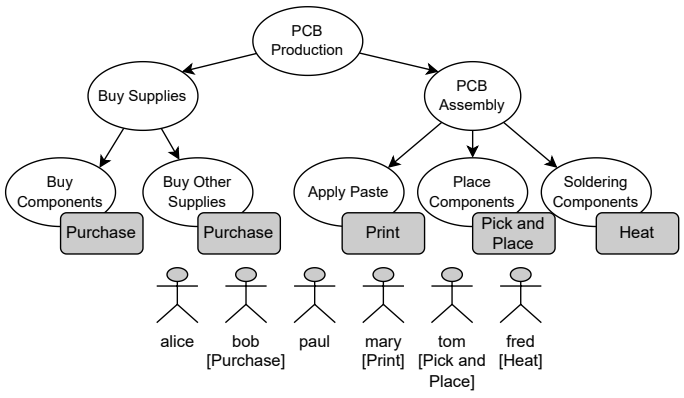
From the contributions of the administration research field to the contributions of the computing science research field, this work covers the state-of-the-art in organisational design. It depicts the computational models, presenting them into different classes, and compares GoOrg with other automated organisational structure generators.

This work also introduces *GoOrg4Prod* and *GoOrg4DSN*, specialisations of GoOrg which define particular processes. They illustrate how the model can be customised for different purposes by selecting particular constraints. These specialisations divide the design into subprocesses. A distinction between processes facilitates the encapsulation of parts of the design into modules. For example, the separation of the binding subprocess from the generating subprocess is relevant for making quicker adaptations for running organisations. The implementation of GoOrg is also depicted, and a discussion regarding organisational design aspects is presented.

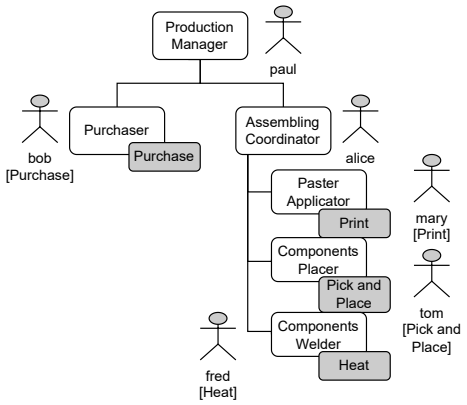
1.2 Problem and Research Questions

The first challenge this research addresses is the automated design of organisational structures. An organisations' generator must consider that a problem domain has a set of goals to be achieved, and possibly these goals cannot be achieved by a single agent. In this sense, it is necessary to organise agents into some structure in which the goals can be distributed across organisational positions. The agents thereby assume organisational positions and are in charge of completing their assigned goals. There must be a way to constrain the generator on finding solutions, which depends on each goal and on the problem domain. For instance, in some domains, it makes sense to associate the goals and the agents with some skills. Therefore, a goal can indicate the required skills that an agent should have to achieve it.

To illustrate the problem, it is considered the set of goals for the *Printed Circuit Board (PCB) Production* scenario presented in Figure 1a. In the mentioned figure, each circle represents a goal, each shaded box refers to the needed skills to achieve the goal, and the bodies on the bottom represent available agents that can be part of the organisation. It is intended to generate organisational structures like the one shown in Figure 1b. In this figure, each unfilled box represents an organisational position of a structure, each shaded box refers to the needed skills to achieve the goal and the bodies closer to each organisational position represent the agent that is occupying each organisational position.



(a) Inputs: A set of goals with required skills.

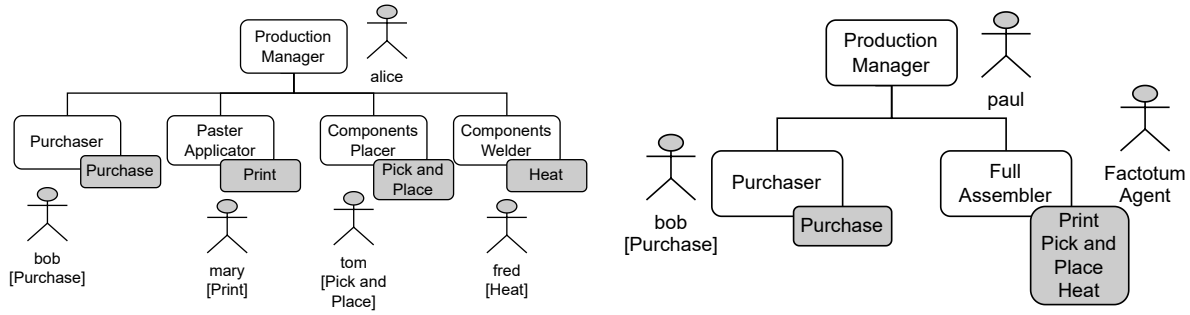


(b) Output: An organisational structure.

Figure 1 – Automated design of a *PCB Production* scenario.

However, from the given set of goals, the structure illustrated in Figure 1b may not be the only possible solution. Indeed, there are numerous locations in the hierarchy where the positions shown in Figure 1b can be organised. They may also be a part of various hierarchies or other kinds of structures, and they can be in charge of a variety of goals. The goals can also be split, and their parts can be assigned to different positions. Besides, the structure must be feasible according to the available resources (agents) for this domain problem. Figure 2 illustrates some of the many candidates that can be generated. In this

sense, the second challenge this research addresses is how to choose an organisational structure candidate among others.



(a) A flat structure for specialised agents.

(b) A structure with a factotum agent.

Figure 2 – Some other candidate solutions for the *PCB Production* scenario.

For choosing a candidate, according to relevant characteristics for the problem domain, it is necessary to somehow quantify the potential solutions (candidates). Indeed, for the *PCB Production* scenario illustrated, there are some characteristics of the organisational structures that can be inferred. For instance, the solution illustrated by Figure 1b has many *specialist* positions, i.e., each agent achieves just one goal, leaving other goals to other agents of the organisation and that structure is also very hierarchical (tall). The solution illustrated by Figure 2a has also many *specialist* positions, but it is flatter. Still, the solution shown in Figure 2b is more compact than the others as it takes the presence of a factotum agent into account. As demonstrated, two attributes that may be inferred from hierarchies are the generality (contrasting with speciality) of positions and the height of structures. Such attributes can be quantified and then utilised to sort and choose organisational structures in accordance with the user's preferences.

Furthermore, generating and choosing organisational structures is usually a computationally expensive task and it might be required to repeat it several times. Indeed, it might be necessary to make adaptations as the system's conditions change over the organisation's lifecycle. Fortunately, not every change requires a complete design procedure. For instance, the available agents that were informed in the design time may change while the system is running. Some changes in the availability of agents may be solved with simple *reallocations* of agents, i.e., the structure is still the same, just the bindings between agents and positions change. However, there are situations when maintaining the same structure is impractical, such as when agent availability has altered so that no agent meets the requirements to fill an existing position. In these circumstances, a *structure-switching* is appropriate because one of the generated solutions might be better suited for the new condition. Still, there are cases in which there is no solution to any extent, for instance, when the set of goals has changed. In such cases, it is necessary a *redesign*. It is important to provide different ways to adapt the organisation, and to provide information to calculate the impact of a change.

In summary, this research intends to answer the following questions:

- How to automatically generate domain-independent organisational structures for MAS?
- How to choose an organisational structure candidate among others?
- How to make the organisation generator facilitate organisational adaptations in a running (online) MAS?

1.3 Objectives

The general and specific objectives of this research are:

General Objective

Develop a model to automatically design domain-independent organisational structures for Multi-Agent Systems.

Specific Objectives

- Develop a model to automatically generate and choose organisational structures;
- Make the model extendable to be specialised for different domains;
- Consider a set of available agents to allow resources planning and to check the organisational structure feasibility; and
- Allow the generated structures to be sorted according to different criteria.

1.4 Contribution and Relevance

The main contributions of this research are:

- The development of a new model to automatically design and choose organisational structures;
- The development of a novel approach for synthesising organisational positions, which besides facilitating the model parametrisations also helps to generate more solutions;
- A discussion and proposal of new perspectives on some organisational design aspects such as the importance of using impersonal representations of agents like organisational positions or roles, the importance of the use of positions instead of roles for planning resources and different adaptation procedures for organisations; and

- To make available a free and open organisational structure generator tool that can be used, extended and improved by the community.

1.5 Document Structure

The rest of this document is structured as follows: Section 2 presents the state of the art of organisational design research area, ranging from contributions of the Administration Research Field to organisational design models proposed by the MAS community, in which GoOrg is compared to other approaches; Section 3 presents *GoOrg*, a novel extensible model for generating and choosing organisational structures; Section 4 presents *GoOrg4Prod*, a specialisation of GoOrg for a factory domain; Section 5 presents *GoOrg4DSN*, a specialisation of GoOrg for the Distributed Sensor Networks (DSN) domain; Section 6 presents details of the implementation of GoOrg; Section 7 presents a discussion on aspects of organisational design, including procedures for adapting organisations, model inputs, and planning resources; Finally, Section 8 presents conclusions and recommendations for future research.

2 Organisation Design Models

This work proposes an automated design *model* for generating organisations. It often uses the terms organisational design model and organisation generator interchangeably. In fact, a model is an abstraction of a system under study (Kühne, 2006). This work considers that a generator is a model, since each class of generators abstracts organisations in a distinctive approach by undergoing some kind of transformation process. Although this work is focused on automated design models, it is worth contextualising the broad research area of organisational design models.

Different design models can be placed into categories and classes.[1] The first categorisation is set from the research area and the kind of organisational members. There are studies in the Administration Research Field for designing organisations for humans and in the computing area, which is mainly concerned with designing organisations of software. Focusing on the design of organisations of software, there are the subcategories of automated and non-automated models, and organisations of autonomous and non-autonomous entities. Finally, in the matter of this work, it is considered that automated design models are in different classes. Figure 3 provides an overview of the works that lie under the referred categories and classes. Following this, the categories and classes are detailed.

The design of organisations is a long-standing research topic in the administration research field. This field is concerned with organisations formed by humans, such as companies. Most works propose frameworks that state an organisation model and issues to be solved (often iteratively). According to the kinds of tasks, the job can be split into functions and, for instance, if the organisation is geographically distributed, it may require departments for different regions. Among the spawned models are the iterative method proposed by Stoner and Freeman (1992), the *Star Model* proposed by Galbraith (1995), the step-by-step framework proposed by Burton et al. (2011) and the design process introduced by De Pinho Rebouças De Oliveira (2006).

In the design of organisations of software, there are many studies of *Service Composition*, which can orchestrate applications by defining sequences and managing communication among Web services (Wu et al., 2015). *Service Composition* has no concern about the degree of autonomy of the software artefacts it is orchestrating. Besides, there are many studies of organisations formed by computational agents, i.e., autonomous en-

[1] The term “class” is being used to refer to something more specific than a category.

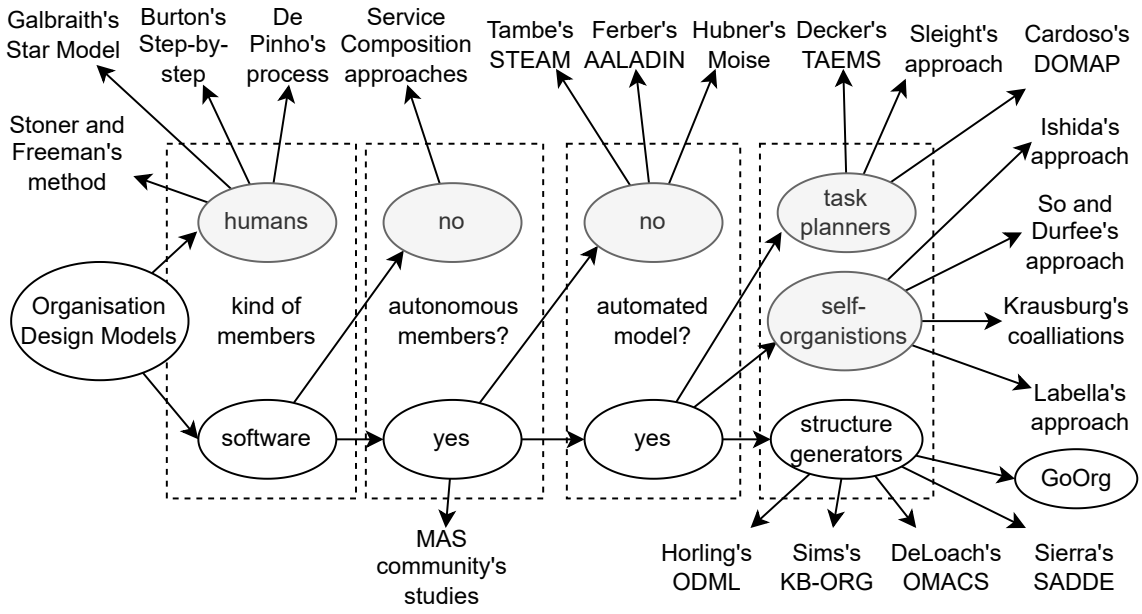


Figure 3 – Design model categories.

tities, which is a research subject of the MAS community.[2] The models for designing organisations for *Service Composition* and for MAS also can be placed into the subcategories of non-automated and automated models. The former models require the definition of the organisation from the user (engineer), and the latter uses artificial intelligence to automatically generate organisations (Amaral and Hübner, 2019, 2020a; Wu et al., 2015).

Considering just models for designing organisations for MAS, some examples of *non-automated organisation generators* are STEAM (Tambe, 1997), AALADIN (Ferber and Gutknecht, 1998) and $\mathcal{M}oise^{+}$ (Hübner and Sichman, 2003). These models allow explicit organisational design in a wide variety of structures and other aspects such as norms, roles, relationships, organisational goals, and ontologies. They are problem-driven approaches, and the organisation's design is specified by a human (the user/engineer). However, this study is focused on *automated computational organisation generators*, i.e., those that automatically generate organisations through computational processes.

To dive into the particular subcategory of automated models, this chapter presents the current state of the art of *automated computational organisation generators*. It is structured as follows: the next sections present three classes of automated organisation generators: Section 2.1 presents *automated organisational design by task planning*; Section 2.2 introduces *self-organisation* approaches; and Section 2.3 presents *automated organisational structure generators*. This work is situated in the third class of organisational structure generators, which is the only one that focuses on generating explicitly modelled organisations (Hatch, 1997; Sims et al., 2004).

[2] Since both humans and agents have some degree of autonomy, there is an intersection between studies of organisations of MAS and studies of organisations of humans.

2.1 Automated Organisational Design by Task Planning

The *automated organisational design by task planning* is the first class to be introduced. These generators usually create *problem-driven* organisations, for specific and often short-term purposes. The organisational structure, when it exists, is not explicit, and it is frequently a non-intended result of a task allocation process. Such generators are focused on solving a given problem by decomposing tasks, allocating them and sending plans to available agents. Usually, there are no roles in this context; the agents are already named and have received their responsibilities somehow. They typically cooperate by fulfilling their tasks. In this sense, a common goal is achieved when a number of tasks are achieved by the organisational members (agents). For instance, a marketplace organisation that has the goal *do business* achieves it when a member achieves the goal *sell product* and another member achieves the goal *buy product*. In this particular class, the automated planning community has produced many contributions to MAS design.

An earlier study on planners able to generate organisations is TÆMS (Decker, 1995). This domain-independent framework provides a way to quantitatively describe individual tasks that are performed in shared environments. It does not use the concept of roles. Tasks, which are slightly similar to goals, are allocated directly to agents. This approach proposes mixing perspectives from traditional task planners, i.e., problem-driven, and self-organisation, which are *experience-driven* approaches. Sleight (2014) presents an agent-driven planner from a similar perspective, but using the Decentralised Markov Decision Process model. It considers the organisation as a first-class object with a dynamic response to environmental changes. There are no goals in this domain-independent approach; it uses rewards in stochastic environments instead. The concepts of roles and explicit structures are also absent.

Cardoso and Bordini (2019) has proposed a domain-independent model called Decentralised On-line Multi-Agent Planning (DOMAP). This task planner first creates factored representations for each agent, based on their limited vision. The second step is to assign goals to agents according to estimations. Following that, agents effectively plan individual actions without sharing private details. Redesign processes may occur if any individual plan fails. Finally, the allocated agents execute their plans. Although the algorithm does not use explicit organisation in the allocation process itself, it can use an organisational structure as input to fill the roles with available agents.

2.2 Self-organisation Approaches

The second class uses *self-organisation* approaches, which is often referred to as Organisation Self-Design (OSD). In this class, the organisations emerge from the dynamics of the scenario, usually defined by the agents' common interests and interactions (Fink

et al., 1983). The resulting organisations are flexible, may operate continuously, have overlapping tasks, formed by named agents, have no external or central control, no hierarchy, and information flows among agents in many directions (Ye et al., 2016). The organisational structure is usually a non-intended ephemeral outcome of this bottom-up process, i.e., it is a result of arbitrary and temporary situations. For instance, the agent that first achieves a goal leads a team to achieve the next goal. The target of this method is to solve some problem and not precisely design an organisation (Sims et al., 2008). *Self-organisation* approaches are more concerned about coordination policies and less concerned about the generated organisational structure itself (Sleight et al., 2015).

In one of the earlier studies, Ishida et al. (1992) has presented an adaptative self-design approach which creates and destroys agents to allocate tasks according to resources and environmental changes and needs. Decker et al. (1997) proposed an approach for adaptation of MAS on organisational, planning, scheduling and execution levels. It uses the *cloning* technique in the execution phase, which is the action of creating a clone agent to partially or totally transfer tasks.[3] A study presented by So and Durfee (1998) encompasses the characterisation of different organisational designs and includes self-organisations and the reconfiguration process for stable organisations. This study also proposes a way to evaluate the organisation's design. In another work, Kamboj and Decker (2007) extended the study performed by Decker et al. (1997), adding a task representation framework, enabling this new method to become domain-independent and to reason about quantitative aspects of tasks. There are several studies on self-organised swarms based on computationally limited agents, which often do not even know about the presence of other agents and that are coordinated by simple mechanisms (Labella et al., 2007; Ye et al., 2016).

More recently, Kota et al. (2012) study presents a decentralised approach in which agents can reason about adaptations to modify their structural relationships when there are opportunities for improving the system performance. Ye et al. (2014) joined cloning and spawning functions on their self-design method. In this sense, besides cloning, when overhead is detected, an agent is also able to delegate a task when it is detected that the agent cannot perform the task at a certain time.

Ohta et al. (2006), Rahwan et al. (2015), Krausburg et al. (2021) and other studies on coalitions are also included in this class of organisation generators. These approaches differ in some characteristics compared to other *self-organisation* approaches as they usually have centralised information and algorithmic process that could not be considered a bottom-up sequence. However, they do share other characteristics of *self-organisation* approaches as they usually generate organisations of named agents, the organisations have

[3] The *cloning* mechanism is used by agents when predicting or perceiving overload (Shehory et al., 1998).

little relationship definitions fostering to handle overlapping tasks and having information flowing in many directions. Besides, the concerns with the algorithmic performance show that these approaches are mainly sensitive to the dynamics of the scenario.

2.3 Automated Organisational Structure Generators

Finally, the third class is the *automated organisational structure generators*. It is focused "on a specification of desired outcomes and the course of actions for achieving them, analysis of the organisational environment and available resources, allocation of those resources and development of organisational structures and control system" (Hatch, 1997). It considers inputs such as organisational goals, available agents, resources and performance targets, producing explicit organisation definitions, which may include roles, constraints, assignments of responsibilities, hierarchy levels, and other relationships. In recent years, this class received little attention. In fact, studies on task planning have attracted interest for a while, and in recent years, studies on *self-organisations* have gained even more attention. Nevertherless, the *automated organisational structure generators* are the only ones that carefully designs organisations as first-class abstractions (Sims et al., 2008). This paradigm of conceiving about organisations treats them as separate from the environment and the agents. This has a number of benefits, one of which is that it supports the separation of concerns of a MAS which is a strategy to deal with complex systems.

This work is in this particular class. The adopted definitions for organisation and organisational design, as follows, are adherent to this particular class of organisation generators.

2.3.1 Structure Generators' Background

According to Pattison et al. (1987, p. 88), "**organisation** design is the problem of choosing the **best** organisation class — from a set of class **descriptions** — given knowledge about the organisation's **purpose** (goal, task, and constraints on the goal) and the **environment** in which the organisation is to operate". This definition assumes that the design is rational and built as a *top-down* process. This approach is commonly used to create explicit organisations. When organisations are explicit entities, agents and designers can reason about the organisation itself, facilitating its improvement. In the following, this work shed light on the emphasized words.

As stated by McAuley et al. (2007, p. 12) **organisations** are "collectivities of people whose activities are consciously designed, coordinated and directed by their members to pursue explicit purposes and attain particular common objectives or goals". Pattison et al. (1987, p. 64) define an organisation as "a group of one or more individuals whose

purpose is to perform some set of tasks in an attempt to achieve a set of goals while observing a set of constraints". For Katz and Kahn (1987), an organisation is "a system of roles" and "the psychological basis of organisational function are seen in terms of motivation to fulfil the roles". For this class, an organisation is represented by a structure of organisational roles or positions and their relationships, and agents occupy these positions cooperating to achieve the organisational goals. In this sense, the organisation and the agents are separate entities. Consequently, agents can reason about the organisation, they can enter or leave it, they can change or adapt it, and they can obey or disobey its rules (Hübner et al., 2010).

Notably, the definition given by Katz and Kahn (1987) mentions organisational roles, which are impersonal representations used as interfaces between the organisation and agents. A role is defined as "an abstract representation of an agent function, service, or identification within a group" (Ferber and Gutknecht, 1998, p. 130), and roles "can be seen as *place-holders* for agents and represent the behaviour expected from agents by the society design" (Dastani et al., 2003, p. 2). A role refers to a set of responsibilities, often materialised as one or more organisational positions, to be occupied by agents.

Many studies of what is known as *contingency theory* point out that there is no one **best** way to design organisations and no general principles for all situations (So and Durfee, 1998). One may even say that organisations, as instances of design models, cannot be considered absolutely right or wrong because it depends on the attribute in focus. An organisation may exist for many purposes and can be inserted into different environments and contexts. For instance, companies positioned in competitive markets have to achieve some set of goals using fewer resources as much as possible, delivering some specified quality as soon as possible. Other organisations exist for other purposes such as common safety, knowledge sharing, technological improvements, social assistance and health care, and so on. Indeed, the concept of "best" is subjective, so it is supposed to be defined by the user (engineer).

In this class, the organisational structure (social structure or simply "structure") is the most essential element of an organisation. As stated by Mintzberg (1983, p. 2), a structure can be defined as "the sum total of the ways in which its labour is divided into distinct tasks and then its coordination is achieved among these tasks". It represents the existing positions of an organisation, showing the hierarchy, relationships, and responsibilities (Daft, 2009). It refers to an administrative instrument resultant of identification, analysis, ordering and grouping of activities and resources of companies, including span degrees and decision process to achieve the expected goals (Fink et al., 1983). As seen, the structure is intrinsically linked to many other organisational aspects. Besides, as pointed out by Durfee et al. (1987, p.1280), "an organisational structure specifies a set of long-term responsibilities and interaction patterns", and it "provides guidance without

dictating local decisions". Notably, the structure is a staple of organisational design (Kilmann et al., 2010). In fact, "all organisations develop some deliberate structure" (Robbins and Coulter, 2012, p. 6), even when it is not explicit. The inseparability of organisation and structure concepts is observed in different studies in which correlated categorisations are presented (Burns and Stalker, 1994; Hatch, 1997; Pettigrew and Fenton, 2000; Stoner and Freeman, 1992). Some researchers do not even put boundaries between these concepts, often treating organisation and structure as the same thing (Pettigrew and Fenton, 2000). In this class, organisations are described by their structures. From organisational structure **descriptions**, organisations can be instantiated to be occupied by agents in a running system.

Approaches based on organisational roles or positions tend to create formal organisations in a top-down manner on the basis of organisational **purposes**, which are typically stated as a set of goals. For many authors (Newman, 1973; Robbins and Coulter, 2012), goals provide the first pillar of an organisational design, representing the organisation's strategy. A goal is a desired state of the world (Boissier et al., 2016), and thus can be used to define the system's overall behaviour (Uez and Hübner, 2014).

An organisation can be seen as a subsystem embedded in a *supersystem*: the **environment**. The environment provides inputs to its subsystems and consumes their outputs (Hatch, 1997). Organisations are diverse in kind and form according to their purposes and environments. Arguably, it is practically impossible to address specificities of all sorts of organisational purposes and environments. For this reason, domain-independent models in this class allow the user adapting the model for specific domains.

2.3.2 State of the Art

An earlier study in this class is Social Agents Design Driven by Equations (SADDE) (Sierra et al., 2004). It uses as input mathematical models to predict efforts and create an organisational structure. It is a comprehensive method for designing a MAS that starts from a manual process for creating domain-specific equations. Then, it establishes the organisation, which is a semi-automatic process. The last two procedures are the definition of agent models and the creation of a MAS. All these phases are connected by defined transitions, including feedback from the MAS to each earlier phase.

DeLoach and Matson (2004) proposed another approach called Organization Model for Adaptive Computational Systems (OMACS) (see also Deloach et al. (2008); Matson and Deloach (2005)). It is an extension of the work Multiagent Systems Engineering (MaSE), a methodology that among its functions defines a way to identify roles from a given set of goals, in this case, aided by an a priori definition of use cases. However, MaSE is not an automated model like its extension. OMACS proposes a mathematical process in which agents are allocated into roles based on the capabilities that an agent possesses,

and what a role requires. To design an organisation, it needs goals, roles, capabilities, and agent types as input. The model requires a priori defined roles. It does not set hierarchy relationships directly but defines a function for setting relationships in a generic way. Agents can also have a special kind of relationship to define a coordination level.

Sims et al. (2008) have proposed the model Knowledge-Based Organization Designer (KB-ORG) to generate organisations for MAS. Their approach does a combinatorial search over the space of candidate organisations describing both hierarchical and peer-to-peer elements. The major contribution regards efficiency by the use of segmentation of application-level and coordination-level functions in the planning process, reducing computational efforts considerably. When an application-level role is split among agents, the algorithm synthesises a coordination role. The whole process allocates agents to roles, and resources to specific tasks, and creates organisational coordination roles. As inputs, the algorithm has environmental conditions, goals, performance requirements, role characterisations and agents' capabilities. The outputs are the allocation of agents to application-level and coordination-level roles. KB-ORG uses quantitative models to define roles.

In another study, Horling and Lesser (2008) introduced Organizational Design Modeling Language (ODML). Their approach allows quantifying organisation models, which can be used to predict performance and as a heuristic method to choose designs. They argue that with this, it is possible to deduce *how* and *why* a design can be chosen over others for a given context. An algorithm template produces a range of possible organisation instances to be searched by the automated process. The organisation search space is defined by decision points specified by variables and *has-a* relationships. The template is similar to a structure of roles, showing their hierarchy and relationships. The algorithm creates instances for all possible structures foreseen by the templates and, after that, searches for the best one. ODML is considered both as a language and as a search-space algorithm that creates and chooses organisation instances. The input includes organisation characteristics and node definitions. For instance, a role can be defined as a node in which the desired behaviour of an agent that enacts such a role should follow. Other parameters that should be defined are scenario constants, the cardinality of each node, relationships among nodes, and constraints. The authors acknowledged that the approach's drawbacks are the level of effort necessary to build the models and the complexity of the algorithm response.

2.3.3 Comparing Structure Generators

Table 1 gives an overview of *Automated Organisational Structure Generators*, the class of generators that GoOrg belongs to. The models are being compared by their inputs, by characteristics of their organisational generation process, and their outputs.

In this particular class of generators, it is expected to start the organisational

design by the organisation strategy, i.e., the goals. In the first column, it is assessed whether *Goals are inputs*. It can be considered that all the generators in this class have organisational goals as their primary concern. Even the models SADDE and ODML, which require agents' behaviours instead of goals, can be considered as having goals as inputs since the agents' behaviours are usually defined to accomplish goals. The *No need roles as inputs* column indicates if the generator needs a priori defined roles. The definition of roles can be a complex task since it requires knowledge about the domain and available agents to define which sets of responsibilities should be joined together. Even in a known domain, such as a school, in which one may expect the roles of *teacher*, *secretary* and *director*, it is possible to have roles less obvious, such as *tutor* and *discipline coordinator*. Indeed, it is hard to know which roles a MAS should present. GoOrg is the only model that does not require role definitions, easing the user's parametrising job (this statement is further discussed in Section 7.2 and Section 7.5).

Table 1 – Comparison among Automated Organisational Structure Generators.

Organisation Generator	Goals are inputs	No need roles as inputs	Has quantitative analysis	Organisations are explicit	Is domain-independent	Synthesises Roles/Positions	Synthesises Coord. Levels	Synthesises Org. Norms	Synthesises departments	Binds agents and roles/positions	Does resource planning	Does reallocations	Does structure-switchings
GoOrg	Y	Y	Y	Y	Y	Y	Y*	F	Y*	Y	Y	Y*	Y*
SADDE	Y*	-	Y	Y	Y	-	-	-	-	Y	-	Y*	-
OMACS	Y	-	Y	Y	Y	-	-	-	-	Y	-	Y*	Y*
KB-ORG	Y	-	Y	Y	Y	-	Y	-	-	Y	-	Y*	-
ODML	Y*	-	Y	Y	Y	-	-	-	-	Y	-	Y*	Y*

Legend: (Y)es, (-)No, on the (F)uture work and (*) see comments.

The column *Has quantitative analysis* describes the capability of generators to create structures that take into account quantitative parameters. For instance, a model may be parametrised with the expected effort to accomplish a goal, which helps the model generate more accurate organisations considering the production scenario. *Organisations are explicit* refers to models that use explicit organisation representations. Top-down design approaches usually generate explicit organisations as entities, which agents and humans can reason about. An explicit organisation is also a way for entrants to know their responsibilities in the system, easing their cooperation in achieving organisational goals. *Is domain-independent* relates to models that are not restricted to any particular

problem domain. All the assessed models have these three mentioned features.

The next columns are related to the outputs of the generators. *Synthesises Roles/-Positions* refers to the ability to automatically synthesise roles/positions. GoOrg is the only model that is able to synthesise positions, which enlarges the search space, making it possible to find more solutions to a given problem. For example, in the school domain among the solutions generated by GoOrg, some positions have a set of responsibilities that are usually delegated to what is known as a *teacher*, others to a *secretary*, *tutor* and so on. In this sense, GoOrg synthesises positions that can be recognised as usual roles, and it may also synthesise positions that could not be foreseen by the user (engineer). The *Synthesises coordination levels* column represents the ability of the model to synthesise coordination roles. KB-ORG specifically synthesises coordination roles/positions using quantitative data to infer the need for coordination agents. GoOrg synthesises positions, placing them into many combinations regarding their levels in hierarchies, producing both superordinate and subordinate positions. In this sense, GoOrg can synthesise coordination levels because some of the generated structures have coordination goals associated with superordinate positions. *Synthesise organisational norms* indicates whether generators automatically create organisational norms, such as permissions and obligations, for each role of a MAS. None of the works generate organisational norms. *Synthesises departments* refers to the specific ability of the generator to create organisational departments automatically. GoOrg can synthesise multiple hierarchies, which can form multiple organisations or departments of an organisation.

Binds agents and roles column tells whether the model is doing agent allocations into roles/positions. This feature shows that the model can suggest an allocation of the available agents throughout the generated organisational structure. Whether all the roles/positions are filled, an organisation can be considered feasible. Besides, all the works have quantitative analysis, which allows one to set up the generator to create fillable and meaningful organisations. Although all the assessed models are able to check the organisation's feasibility, they use roles. Roles do not ease the planning of resources, since the dynamism of a role-based system (with many-to-many relationships between agents and roles) makes the allocation of resources a very complex and hard-to-determine task. These models can create instances of roles to register that a role is being used a number of times. It is close to the concept of the organisational position, but these models have no functionality to register the number of agents that is necessary to fill the structure (see Section 7.3, for more details). In other words, other models estimate (plan) the number of roles, not the number of agents that are necessary to a system. GoOrg, *Does resource planning* because it uses positions instead of roles, positions reflect the need for agents.

The following columns are related to the capability of the generators to deal with reorganisations. *Does reallocation* refers to the ability to move agents from some posi-

tion/role to another without needing to redesign the organisation. It is considered that all the assessed models can respond to a change in the parametrisation and produce different agents' allocations. Besides, the models are for the design phase of a system, i.e., it is assumed that other coordination mechanisms handle the allocation process during the organisation's lifecycle. *Does structure-switchings* refers to a switch from one structure to another existing one. GoOrg, ODML and OMACS present mechanisms to quantify organisations by attributes, so they can be chosen. Additionally, GoOrg can generate all candidates for a given problem at once, and as the system conditions or user's preferences change, GoOrg can pick a different structure to be used (see Section 7.1, for more details). Since GoOrg synthesises positions and generates structures with no concern about the available agents, the chance of finding an already generated structure that suits a new system's conditions is higher than in other models (see Section 7.4, for more details). However, this feature also has its drawbacks, which are the increased algorithm complexity, memory usage, and time consumed on generating all the candidates.

In summary, the data presented in Table 1 indicates that the assessed models have in common: (i) the (direct or indirect) use of goals as input; (ii) they generate explicit organisations; (iii) they are domain-independent; (iv) they bind agents and roles/positions; (v), they generate feasible organisations; and (vi) they do (or allow) reallocations. However, most of the models are missing other features in which GoOrg stands out: (a) GoOrg does not require roles as inputs; (b) it automatically synthesises positions, which can be placed in different hierarchy levels; (c) GoOrg is the only model that uses the concept of positions, which facilitates resources planning; (d) GoOrg produces multiple hierarchies such as organisations or departments of an organisation; and (e) GoOrg facilitates structure-switching, generating and quantifying organisations by attributes.

Finally, it is essential to point out that there is no single type of organisation suitable for all situations (Horling and Lesser, 2004). It is also true that there is no individual approach ideal for creating all organisations (Daft, 2009). Each technique offers some advantages that the others may lack, especially regarding different organisation generator classes.

3 GoOrg Model

This section presents GoOrg, a model for automatically designing organisations, expressed as structures composed of organisational positions. As stated by Seidewitz (2003, p. 2), "a model is a set of statements about some system under study". GoOrg is a generic model. Its applicability to specific domains lies with the addition of elements and constraints as required by the domain. Thus, GoOrg needs to be extended (specialised) to the domain it is being applied.

In fact, there is a diversity of domains in which organisations are used. For instance, one may want to design organisations of agents for the production of a factory, or for tracking objects of interest, or for rescuing victims of a calamity. Each domain may have particular requirements and indicators of interest. For instance, a factory is concerned with moving, assembling and efficiency; tracking is concerned with identification and vision coverage; and rescue work is concerned with finding, supporting and minimising the impact of a calamity.

For any domain, GoOrg-based models use goals and agents to generate organisations. According to the user's most preferred attributes, the generated structures are sorted, and the best candidate is chosen. GoOrg does not specify any particular technology to be used or how the generation and choosing processes are carried out. Instead, it only defines the expected inputs and outputs for the design, as shown in Figure 4.

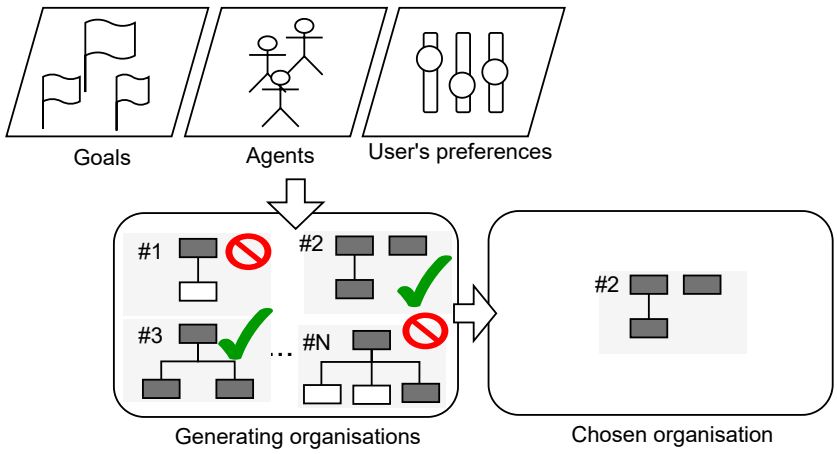


Figure 4 – The generic design process of GoOrg-based models.

Figure 5 illustrates the elements and attributes of the general model, GoOrg. In the following, the model is presented in detail. Section 3.1 describes GoOrg from the

perspective of the model's elements; and Section 3.2 describes GoOrg from the perspective of organisational structure attributes.

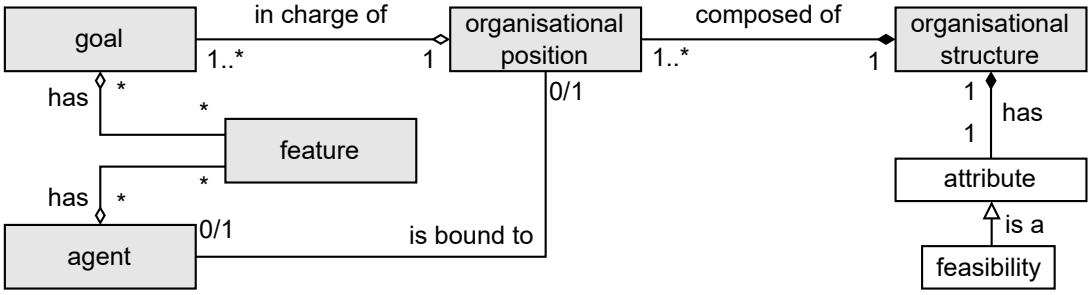


Figure 5 – GoOrg model.

3.1 GoOrg Elements

The GoOrg model considers only essential elements for an organisation's design: goals, agents, organisational positions, features and the organisational structure. Formally, each element is described as follows.

A goal is a desired state of the world to be achieved by the organisation. Goals are to be fulfilled by agents when they occupy organisational positions, so they become performers in these positions. It is assumed that the agent occupying a position is in charge of achieving the position's assigned goals.

Definition 1 (goal)**.** *A goal g is represented as a symbol, and the set of all goals is denoted by G.*

$$g : symbol,\ g \in G$$

An agent occupies an organisational position to achieve its associated goals. An example of an agent is a computer that has a microprocessor able to execute a number of instructions per second, or an application that solves factorials. Formally, GoOrg defines an agent as follows.

Definition 2 (agent)**.** *An agent a is represented as a symbol, and the set of all agents is denoted by A.*

$$a : symbol,\ a \in A$$

Positions are *place-holders* for agents in an organisation. They reflect the necessity of agents for an organisation to function. If every organisational position has an agent to occupy it, the organisation is considered feasible. The agent that occupies a position is in charge of achieve the goals assigned with that position. Each position can only have one agent in it at a time, and each agent can only occupy one position at a time. Formally, an organisational position is defined as follows.

Definition 3 (position)**.** *A position p is represented as a symbol, and the set of all positions is denoted by P. The goals assigned to p are specified by the function gp, considering that p must have at least one goal associated. The function ap specifies the agent occupying the position p, considering that p is a "free position" when $ap(p) = \epsilon$, and that an agent cannot be bound to more than one position.*

$$p : symbol,\ p \in P$$

$$gp : P \rightarrow 2^G$$

$$\forall p \in P,\ gp(p) \neq \{\}$$

$$ap : P \rightarrow A \cup \{\epsilon\}$$

$$\forall p, p' \in P,\ p \neq p' \wedge ap(p) \neq \epsilon \wedge ap(p') \neq \epsilon \Rightarrow ap(p) \neq ap(p')$$

GoOrg considers that a feasible organisation has all positions fillable by the available agents. To check if an agent can occupy a position, it compares the *features* that an agent has to the features that the goals assigned to a position have. For instance, the goal *solve combinatorics* can be associated with the feature *solve factorials*, representing a required *skill* to fulfil the goal. Similarly, the agent *calculator* may have the feature *solve factorials* representing a *skill* it has. In this case, the agent *calculator* is able to fulfil the goal *solve combinatorics* since it has the required *skill*. In this regard, a feature is defined as follows.

Definition 4 (feature)**.** *A feature f is an n-tuple, in which the first element is a symbol. Besides the first element, optionally, a feature may have other elements $(e_2, ..., e_n)$. The set of all features is denoted by F. The function fg specifies the features required by a goal. The function fa specifies the features an agent has.*

$$f : \langle symbol,\ e_2, \ldots, e_n \rangle,\ f \in F$$

$$fg : G \rightarrow 2^F$$

$$fa : A \rightarrow 2^F$$

GoOrg considers that each organisational structure is a particular description of an organisation. GoOrg defines an organisational structure as follows.

Definition 5 (structure)**.** *An organisational structure o is represented as a tuple. It is composed of the already presented sets and functions G, A, P, F, gp, fg, fa and ap.*

$$o : \langle G, A, P, F, gp, fg, fa, ap \rangle$$

3.2 Attributes of an Organisational Structure

Each generated organisation has attributes that quantify it. The model only defines the attribute *feasibility*. The feasibility of an organisational structure is the ratio between positions bound to agents and the total number of positions. It represents how viable it is to fill the structure using the available agents.

Definition 6 (feasibility)**.** *The feasibility of the organisational structure o is represented as $\kappa(o)$, a real number in the range [0,1]. It is the ratio of the number of bound positions and the number of all organisational positions of the organisation o (Eq. 3.1). The set B contains the agents that are bound to positions in P (Eq. 3.2). The organisation is entirely feasible ($\kappa(o) = 1$) when every position is bound to an agent.*

$$\kappa(o) = \frac{|B|}{|P|} \tag{3.1}$$

$$B = \{ap(p) \,|\, ap(p) \neq \epsilon,\ p \in P\} \tag{3.2}$$

3.3 GoOrg Highlighted Characteristics

The proposed model for automatically generating organisations considers that an organisation is represented by its structure of positions. The organisational goals are assigned to positions. Each position should by occupied by an agent. An agent can occupy a position when it has the features required by that position

As a generic model, GoOrg does not specify any kind of relationship between positions. The kinds of relationships (such as "is superior of"), features and other specificities of a particular domain should be defined in an extension of GoOrg.

An organisational structure might have any form (shape). For instance, it can be single positions with no relationships with each other, a group of positions with clear relationships between them, or groupings of positions with relationships within their own groups but none between them.

Besides, it is worth mentioning that the model does not specify that the generation of structures depends on the set of available agents. However, the set of available agents is used to check the structure's feasibility. If there is no available agent, no generated structure is feasible. If the set of available agents changes over time, a new assessment of the organisation's feasibility is made.

4 GoOrg4Prod: A Specialisation for a Factory Production Line Domain

GoOrg4Prod illustrates how GoOrg can be used in a particular domain.[1] It intends to generate structures of agents responsible for production activities in a factory. It is assumed that an external mechanism will pick the chosen organisational structure that is indicated by *GoOrg4Prod*. In this work, the term *GoOrg4Prod* refers to both a model extension and an implementation that can generate organisation descriptions for a specific domain.

The generation of hierarchical structures is considered in this domain. It is assumed that the hierarchical levels of each organisational member are relevant and used somehow in the MAS. Organizational charts, despite their limitations, can be used to represent hierarchies because they are focused on representing superior-subordinate relationships.[2]

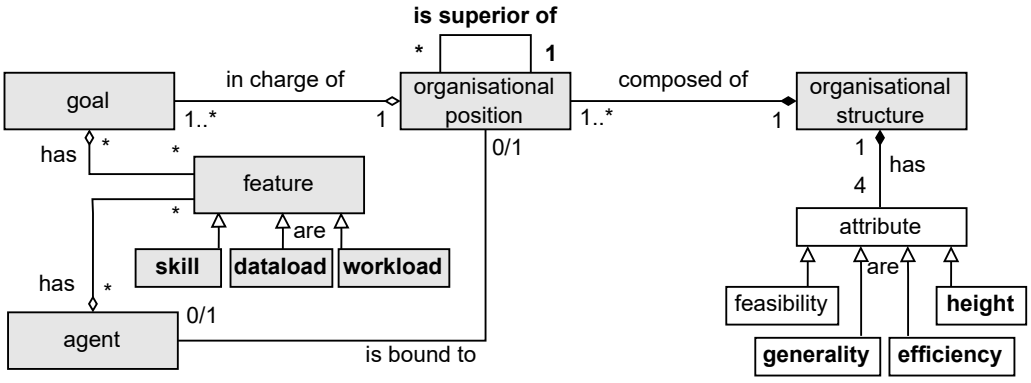


Figure 6 – GoOrg4Prod model.

GoOrg4Prod **synthesises** positions **generating** organisations in which goals should be achieved routinely by executing some *workloads*. Notice that this particular approach is using a baseline of 24 hours. It is considered that effort repeats at every baseline, as cycles, such as a day in a factory. To be able to execute *workloads*, agents should have some *skills*. *GoOrg4Prod* **matches** agents and positions using *skills*. As it is generating hierarchies, the structures present the attributes *height* and *generality*, as later explained. Besides, *GoOrg4Prod* uses *workloads* to calculate the organisation's efficiency, which among other attributes can be used to **choose** organisations based on the user's preferences.

[1] An implementation of *GoOrg4Prod* is available at `https://github.com/cleberjamaral/GoOrg4Prod`.

[2] There are criticisms arguing that organisational charts miss crucial aspects of organisational structures (Mintzberg and Van der Heyden, 1999).

Sometimes, it is also required that the agent that performs a goal communicates something to the performer of another goal, by sending *dataloads*. A maximum capacity of handling *dataloads* and *workloads* can be set on *GoOrg4Prod* to avoid exceeding agents' capacities. Figure 6 highlights in bold font the organisational attributes and features added by *GoOrg4Prod* on extending GoOrg model.

As an illustrating application, it is considered a production line scenario in which the head of a conveyor belt must be fed with items that are inside boxes, which need to be moved from shelves. An external database must be accessed to get orders for items. The system has to get boxes from shelves, move them to near the conveyor belt, and finally pick items from boxes to place on the head of the conveyor belt. In this motivating scenario, it is required to move a certain quantity of boxes a day. It is predicted to spend a certain time on each activity. Some *skills* are required to achieve these goals, they are: *db access*, *lift*, *move* and *pnp (pick and place)*. While achieving some goals, it is necessary to send data to the agent(s) in charge of other goals. Hence, the bandwidth is measured in Kbps to represent communication usage. In this example, it is necessary to consult a database for getting orders which can trigger a series of communications regarding the stage of the movement of the item. The types of *dataload* messages are: *requested box*, *box ready* and *items ready*. Figure 7 illustrates how this scenario is modelled.

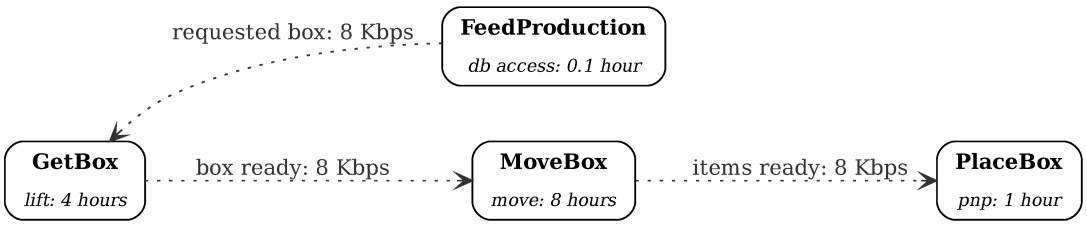


Figure 7 – A given set of goals with associated *workloads* and *dataloads*.

4.1 GoOrg4Prod Elements

GoOrg4Prod extends GoOrg elements by specifying three features: *workloads* and *dataloads*, which are associated with goals, and *skills*, which are associated with agents. These features constrain the organisational design while synthesising positions, arranging hierarchies and binding agents to positions.

From the agent's perspective, it is considered that agents have *skills*. This particular model assumes that every agent has the same time per baseline to work on organisational goals. It also assumes there is no difference in agents' performance, i.e., any agent will spend the same time to achieve a given goal. A *skill* s is defined as a singleton tuple which belongs to the set S of all *skills*, as follows.

$$s : \langle symbol \rangle,\ s \in S$$

According to the aims, *workloads* may be required for achieving goals. A *workload* w represents a demanded effort $e \in \mathbb{R}^+$ which requires a *skill* $s \in S$ to be performed. In the example of Figure 7, there are four *workloads*: (i) the *skill db access* with a predicted effort of 0.1 hours; (ii) the *skill lift* with a predicted effort of 4 hours; (iii) the *skill move* with effort equals to 8 hours; and (iv) the *skill pnp* with a predicted effort of 1 hour. The function wg maps goals to their *workloads*, as follows.

$$w : \langle s, e \rangle,\ s \in S,\ e \in \mathbb{R}^+,\ w \in W$$
$$wg : G \rightarrow 2^W$$

Still according to the aims, it is considered that to achieve a goal it may be necessary to establish communications, which is named *dataloads*. A *dataload* i represents a message, which has an estimated usage of bandwidth $d \in \mathbb{R}^+$ while it is sent to the performer of the recipient goal $r \in G$. The function ig maps goals to their *dataloads*, as follows.

$$i : \langle symbol, r, d \rangle,\ r \in G,\ d \in \mathbb{R}^+,\ i \in I$$
$$ig : G \rightarrow 2^I$$

The set F of features, is composed of *workloads*, *dataloads* and *skills*, as follows. To match positions and agents only *skills* are used (the *skills* that agents have and *skills* that *workloads* require).

$$F = W \cup I \cup S \tag{4.1}$$

Additionally, *GoOrg4Prod* considers that organisational positions may have "is superior of" relationships which stands for superordinate-subordinate relationships. Indeed, a tree representing a hierarchy may have positions belonging to different levels. The function $sp(p)$ records the position p', which is the immediate superordinate of the position p. If p has "no superordinate", $sp(p) = \epsilon$. This function is defined as follows.

$$sp : P \rightarrow P \cup \{\epsilon\}$$

Since the relationship "is superior of" and the function wg are used by the generator, they are being added as elements of the organisational structure. In this sense, the stated definition of the structure (Definition 5) is replaced by the following formula.

$$o : \langle G, A, P, F, gp, fg, fa, ap, sp, wg \rangle$$

An organisational structure is formed by one tree (hierarchy) or more, as a forest of hierarchies. A tree may be composed of only one position (having no superordinate and no subordinates). A forest with all trees composed of only one position has all these positions in the same hierarchy level, which is the flattest structure. Among the generated structures, there may exist trees that are composed of similar positions but in different places in the hierarchies. For instance, in a factory hierarchy, the position *assembler* is superordinate of the *packer*; in another hierarchy, *packer* is superordinate of *assembler*; and in another, both are on the same level.

Finally, *GoOrg4Prod* has a few more parameters. In practice, agents have limited capacity for performing *workloads* and sending/receiving *dataloads*. In response to this practical issue, in *GoOrg4Prod*, $\phi_p \in \mathbb{R}^+$ is defined to represent the maximum *workload* allowed on each position. Following the same idea, $\delta_p \in \mathbb{R}^+$ is defined to represent the maximum *data load* allowed to each position, which regards the *dataloads* that address a goal g. To allow splitting goals into smaller ones when necessary, $\phi_g \in \mathbb{R}^+$ and $\delta_g \in \mathbb{R}^+$ are defined to refer to the maximum grain size for *workloads* and *data loads*.

4.2 GoOrg4Prod Added Attributes

GoOrg4Prod defines new attributes of an organisational structure. The *height* of the structure is calculated using superordinate-subordinate relationships. Based on how the goals are distributed across positions, the *generality* of the structure is calculated. From the added feature *workload*, the efficiency of an organisation can be quantified. These attributes are represented in a three-dimensional space. Every generated organisation has a coordinate in this space. Figure 8 illustrates the organisation attributes space.

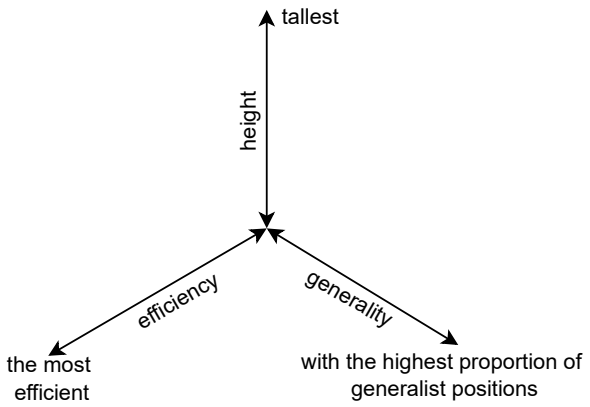


Figure 8 – *GoOrg4Prod* organisational structure attributes in three dimensions.

Height refers to how centralised and bureaucratic a hierarchical organisation is, since a long chain in a tree may imply that the organisation is very centralised, impacting its decision-making model. *Generality* indirectly changes the shape of the structure, in both the vertical and the horizontal directions, since it may impact the organisation's workflow. Besides, one may argue that generalist positions may improve robustness since other agents would be able to take on responsibilities in case of an agent fails. Efficiency indicates how close the combined capacity of the agents, which will occupy the generated positions, is to the expected efforts considering the given goals.

The height of an organisational structure is defined as a ratio between the actual height and the tallest hierarchy that *GoOrg4Prod* can generate from the input. The top level is formed by all top superordinate positions (the positions that have no superordinate, i.e., $sp(p) = \epsilon$). The next level contains all subordinates of the top superordinate positions. The other levels follow the same idea.

Formally, the height of an organisational structure o is represented as $\tau(o)$, a real number in the range [0,1]. It is the ratio between the actual height and the maximum height that the model generates (Eq. 4.3). The function $l(p)$ maps a position p to an integer representing the hierarchical level that the position p is situated at (Eq. 4.2). The function $l(p)$ counts from the position p to its top superordinate position, one level for each relative superior in the organisational structure. The longest chain of hierarchies (trees of the structure) is defined by $max(l(p))$, for all $p \in P$. The cardinality of the set of goals ($|G|$) represents the maximum chain of positions that the model produces. The cardinality $|G|$ should be equal to or greater than 2 to generate different and comparable candidates.

$$l(p) = \begin{cases} 0 & sp(p) = \epsilon \\ 1 + l(sp(p)) & \text{otherwise} \end{cases} \tag{4.2}$$

$$\tau(o) = \begin{cases} \frac{max_{p \in P}(l(p))-1}{|G|-1} & |G| \geqslant 2 \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

The generality of an organisational structure measures the similarity of positions considering their assigned goals. The most generalist organisation has all goals assigned to every position, i.e., all the agents would be able to play any position and perform any goal. *GoOrg4Prod* can split a goal into smaller ones to assign it to multiple positions, as explained in Section 4.3.1. This specialisation assumes that achieving all parts implies the achievement of the original goal.

Formally, the generality of an organisational structure o is represented as $\theta(o)$, a real number in the range [0,1]. It is the ratio between the actual and the maximum possible number of goals assigned to positions (Eq. 4.4). The set GP contains the recorded goals

for all positions (Eq. 4.5), and its cardinality is represented as $|GP|$. The minimal possible number of goals spread across positions is given by the minimal of the cardinality of G and the cardinality of P. The maximum possible number of goals assigned to positions is given by the cardinality of the set G ($|G|$) times the cardinality of the set P ($|P|$). In this sense, the maximum generality ($\theta(o) = 1$) occurs when every position is assigned to every goal. In contrast, the minimum generality ($\theta(o) = 0$), which represents the most specialist organisation, has each goal assigned to only one position.

$$\theta(o) = \frac{|GP| - min(|G|, |P|)}{|G||P| - min(|G|, |P|)} \tag{4.4}$$

$$GP = \bigcup_{p \in P} gp(p) \tag{4.5}$$

In *GoOrg4Prod* the efficiency of a structure o is represented as $\eta(o)$ a real number in the range [0,1]. It is the *capacity utilisation* of the organisation, which is given by the ratio between the *utilisation* and the *capacity* (Eq. 4.6). In the equation, $\pi_i(a)$ refers to the i-th element of the tuple a. The *utilisation* is given by the sum of *workloads*' efforts associated with all given goals. The organisation's *capacity* is the number of positions of the organisational structure times ϕ_p (the maximum *workload* allowed per position).

$$\eta(o) = \frac{\sum_{g \in G} \pi_2(wg(g))}{|P|\phi_p} \tag{4.6}$$

With these new attributes, *GoOrg4Prod* can quantify an organisation by *height*, *generality*, *feasibility* and *efficiency*.

4.3 GoOrg4Prod Processes

GoOrg4Prod generates and chooses organisations in a chain of four processes: (i) the given goals in G are split down into smaller goals according to the granularities (ϕ_g and δ_g) set by the user; (ii) organisational positions are synthesised and structures are generated in a process that searches the space with all possible organisations according to the supported transformations; (iii) the positions of generated structures are bound to the given agents and the feasibility of the organisation is calculated; and (iv) an organisation with positions and bound agents is chosen. It is important to mention that although this work is suggesting a method for organisation generation, it is not claiming it is the only possible one. Figure 9 illustrates the mentioned processes.

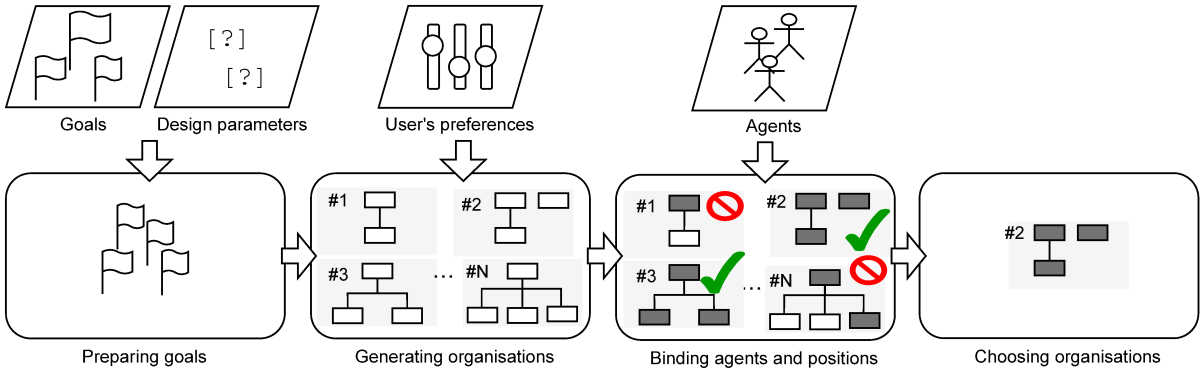


Figure 9 – The four processes of *GoOrg4Prod*.

4.3.1 Preparing goals for assignments

GoOrg4Prod splits the given goals into smaller ones according to a given *workload* grain ϕ_g and the bandwidth grain δ_g parameters. In this sense, the goals of G that are set with more effort or more bandwidth than the respective grains should split into smaller ones, creating the set G'. Splitting goals allows assigning the same goal (with less effort and bandwidth usage) to multiple positions. This process may increase the generality of the final organisation since it potentially creates interchangeable positions.

To exemplify, it is considered the set G illustrated in Figure 7. This set has four goals (**FeedProduction**, **GetBox**, **MoveBox** and **PlaceBox**) which are associated with *workloads* and *dataloads*. Considering that the grain ϕ_g is set as 4 hours and the grain δ_g is set as 1000 Kbps, the *workload* effort $e = 8$ of the goal **MoveBox** is greater than ϕ_g, requiring to split this goal into smaller ones. To fit them to ϕ_g, **MoveBox** is split into two similar goals with half of the original effort. Since the *dataloads* are not surpassing δ_g, it is not necessary to split the goal due to exceeding bandwidth. However, as **MoveBox** is split into two parts (**MoveBox$0** and **MoveBox$1**), the *dataloads* that addresses it, and are originated from it, are also split. The resulting set of goals that suit the given granularities are illustrated in Figure 10.

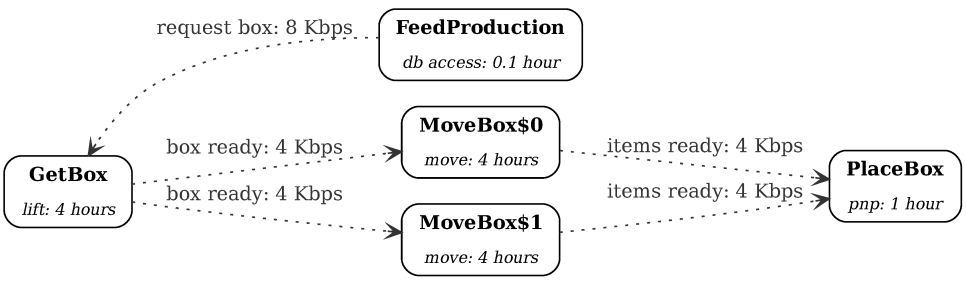


Figure 10 – The set G' of split goals for $\phi_g = 4$ and $\delta_g = 1000$.

The process of splitting down goals by *workloads* is illustrated in Alg. 1. The process of splitting goals by *dataloads* is performed using as input the set G'. This another

process focus on *dataloads* instead of *workloads* using the bandwidth grain size δ_g, creating the set G''.

Algorithm 1: splitGoalsByWorkload

```
Data: G the set of goals, φ_g the max workload grain size
Result: G' the resulting set of split goals
begin
    G' ← {}
    foreach Goal g in G do
        e ← 0
        foreach Workload w in wg(g) do
            e ← e + π2(w)  // sum of efforts of workloads in g
        end
        slices ← max(ceil(e/φ_g), 1)  // number of slices
        if slices = 1 then
            G' ← G' ∪ {g}
        else
            foreach n from 1 to slices do
                g' ← new goal
                g' ← concatenate(g, "$", n)
                foreach Workload w in wg(g) do
                    wg(g') ← wg(g') ∪ {g' ↦ ⟨π1(w), π2(w)/slices⟩}
                end
                foreach Dataload i in ig(g) do
                    ig(g') ← ig(g') ∪ {g' ↦ ⟨π1(i), π2(i), π3(i)/slices⟩}
                end
                G' ← G' ∪ {g}
            end
        end
    end
end
```

4.3.2 Generating organisations

GoOrg4Prod generation process is based on a state-space search algorithm. Each state represents a partial or finished structure of organisational positions. The initial state is a structure with no positions and all $g \in G$ to be assigned to positions. To simplify, it is being considered that $G = G'$, i.e., G is the set of goals after some of them were split.

Every goal assigned to a position is a step towards building an organisational structure. The solution is an organisational structure o with all $g \in G$ assigned. The search algorithm uses a cost function based on the user's preferences. It first explores search states representing the most preferred organisations. After building the most preferred solutions, the algorithm keeps building other structures until there are no unexplored search states.

To generate a variety of structures, *GoOrg4Prod* apply three structure transformations considering every $g \in G$ in two stages. The structure transformations are illustrated in Figure 11, in which the previous states are represented on the top of the figure (identified by *a1*, *b1* and *c1*, respectivelly to each kind of transformation). The states on the bottom of the figure (identified by *a2*, *b2* and *c2*), represent the result of each transformation. On each represented transformation, on the lefthand side, the goals are illustrated, being the unfilled ones not assigned goals and shaded ones the goals that were already assigned to some position. On the righthand side, the structure of positions is illustrated.

In the first stage of the search, a transformation assigns every $g \in G$ to a new top superordinate position, i.e., $|G|$ structures o are created with only the first position of the hierarchies (Figure 11a). In the second stage, every remaining $g \in G$ is: (i) assigned to a new top superordinate position, i.e., it applies the same transformation used in the first stage which creates a new tree in the forest, repeating to each remaining goal the transformation illustrated in Figure 11a; (ii) assigned to new positions that are created to be subordinate of every $p \in P$, repeating to each remaining goal and every existing position the transformation illustrated in Figure 11b; and (iii) assigned to every existing $p \in P$ (no position is created), repeating to each remaining goal and every existing position the transformation illustrated in Figure 11c.

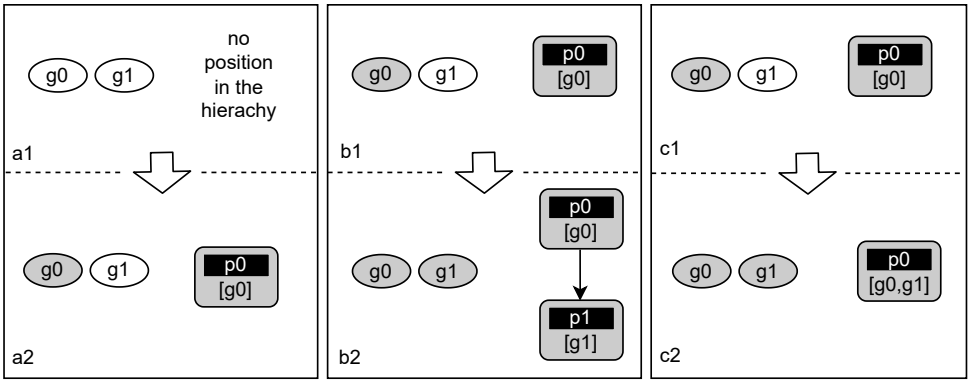


Figure 11 – Supported transformations.

The mentioned transformations are detailed as follows. In the first transformation, called $addSuperiorPosition(g)$, a goal is assigned to a new *superordinate position*. The new position is added to the set of existing positions P. The assigned goal g is recorded by the function gp, the *skills* of the *workloads* of g given by the function wp are recorded by the function fg (set S') and the function sp records that p has no superior. This transformation from the structure o to the structure o' is formalised as follows in which barred arrow notation for elements is used to represent the mappings of a function, i.e., $a \mapsto b$ means b is the image of a, such that a and b are elements of finite sets.

$$o = \langle G, A, P, F, gp, fg, fa, ap, sp, wg \rangle$$
$$p = new\ Position$$
$$addSuperiorPosition(g)$$
$$S' = \{\pi_1(w) \,|\, w \in wg(g)\}$$
$$o' = \langle G, A, P \cup \{p\}, F, gp \cup \{p \mapsto \{g\}\},$$
$$(fg \backslash \{g \mapsto fg(g)\}) \cup \{g \mapsto (fg(g) \cup S')\}, fa, ap, sp \cup \{p \mapsto \epsilon\}, wg \rangle$$

In the second transformation, called $addASubordinate(g, p')$, the goal g is assigned to a new position p that is a subordinate of p'. Thus, the new position is added to the set of existing positions P. The goal g is assigned to the new position p, which is recorded by the function gp. The *skills* of the *workloads* of g given by the function wp are recorded by the function fg (set S'). The function sp records p' as superior of p. This transformation is formalised as follows.

$$o = \langle G, A, P, F, gp, fg, fa, ap, sp, wg \rangle$$
$$p = new\ Position$$
$$addASubordinate(g, p')$$
$$S' = \{\pi_1(w) \,|\, w \in wg(g)\}$$
$$o' = \langle G, A, P \cup \{p\}, F, gp \cup \{p \mapsto \{g\}\},$$
$$(fg \backslash \{g \mapsto fg(g)\}) \cup \{g \mapsto (fg(g) \cup S')\}, fa, ap, sp \cup \{p \mapsto p'\}, wg \rangle$$

In the third transformation, called $joinAPosition(g, p)$, the goal g is assigned to an existing position p. Thus, no new position is created, just g is assigned to the existing p being recorded by $gp(p)$. After assigning g to p, the *skills* of the *workloads* of g given by the function wp are recorded by fg (set S'). Besides, the function fg records the *dataloads* of g recorded by ig (set I'), which excludes *dataloads* of other goals assigned to p that address g, since they are *loopbacks* to p. In other words, it is assumed that an agent occupying the position p would not need to send a message to itself. This transformation is formalised as follows.

$$o = \langle P, G, F, A, gp, fg, fa, ap, sp, wg \rangle$$
$$joinAPosition(g, p)$$
$$S' = \{\pi_1(w) \,|\, w \in wg(g)\}$$
$$I' = ig(g) \backslash \{i \,|\, i \in ig(g) \wedge \pi_3(i) \in gp(p)\}$$
$$o' = \langle P, G, F, A, (gp \backslash \{p \mapsto gp(p)\}) \cup \{p \mapsto (gp(p) \cup \{g\})\},$$
$$((fg \backslash \{g \mapsto fg(g)\}) \cup \{g \mapsto (fg(g) \cup S' \cup I')\}), fa, ap, sp, wg \rangle$$

These structure transformations are applied by the search algorithm on every state exploration. Algorithm 2 specifies the successor states of the search.

Algorithm 2: Successors - creates new states to explore

```
Data: G_na a list of not assigned goals, P the current set of positions
Result: U a list of successor states
begin
    List U                                  // The successors list
    if P = {} then
        foreach Goal g of G_na              // For each not assigned goal
         do
            U.add(addSuperiorPosition(g))   // Add as a superordinate
        end
    else
        foreach Goal g of G_na              // For each not assigned goal
         do
            U.add(addSuperiorPosition(g))   // Add as a superordinate
            foreach Position p of P         // For each existing position
             do
                U.add(addASubordinate(g,p)) // Add as a subordinate of p
                U.add(joinAPosition(g,p))   // Assign g to existing p
            end
        end
    end
end
```

The set G_{na} has all non-assigned goals and the set P has all positions of this state (according to the partial organisational structure o). Each transformation creates a new state to be explored. For instance, $addSuperiorPosition(g)$ creates a state based on the current state (current partial o) assigning g to a new superordinate position in P and updating G_{na}. Each created state will be later explored, and new successors may be created. To generate successors of the created search states, the algorithm use G'_{na} such that $G'_{na} = G_{na} \backslash \{g\}$.

GoOrg4Prod does not stop searching after finding a solution. Instead, it keeps exploring all the search space. Thus, the final output is not only one solution, but a list of all possible solutions.

To illustrate how the algorithm performs the search, a set G with two goals (*g0* and *g1*) is taken into consideration. To simplify, no features are being presented, but it is being considered that each goal has some effort associated, needing to be assigned to some organisational position. Figure 12 shows the generation process while searching in the space of solutions for this particular set of goals. In the initial state, represented at the top of Figure 12, there is no hierarchy yet, the only transformation that can be applied is $addSuperiorPosition(g)$. In one successor state, $addSuperiorPosition(g)$ is applied to $g0$,

and in another state, it is applied to $g1$, both generating hierarchies of a single position. Neither of these states is a target state, since there is still a goal in G to be assigned. At the bottom of the figure, the four target states for this G are presented.

Since each of the intermediary states has one existing position in the hierarchy, the remaining goal of these states is the subject of the three transformations. According to the structure transformation, a different state is created. Some of the states are similar to a previously created state, thus pruned. To save space in this illustration, it is pointed to the similar outcome that was considered to prune the other.

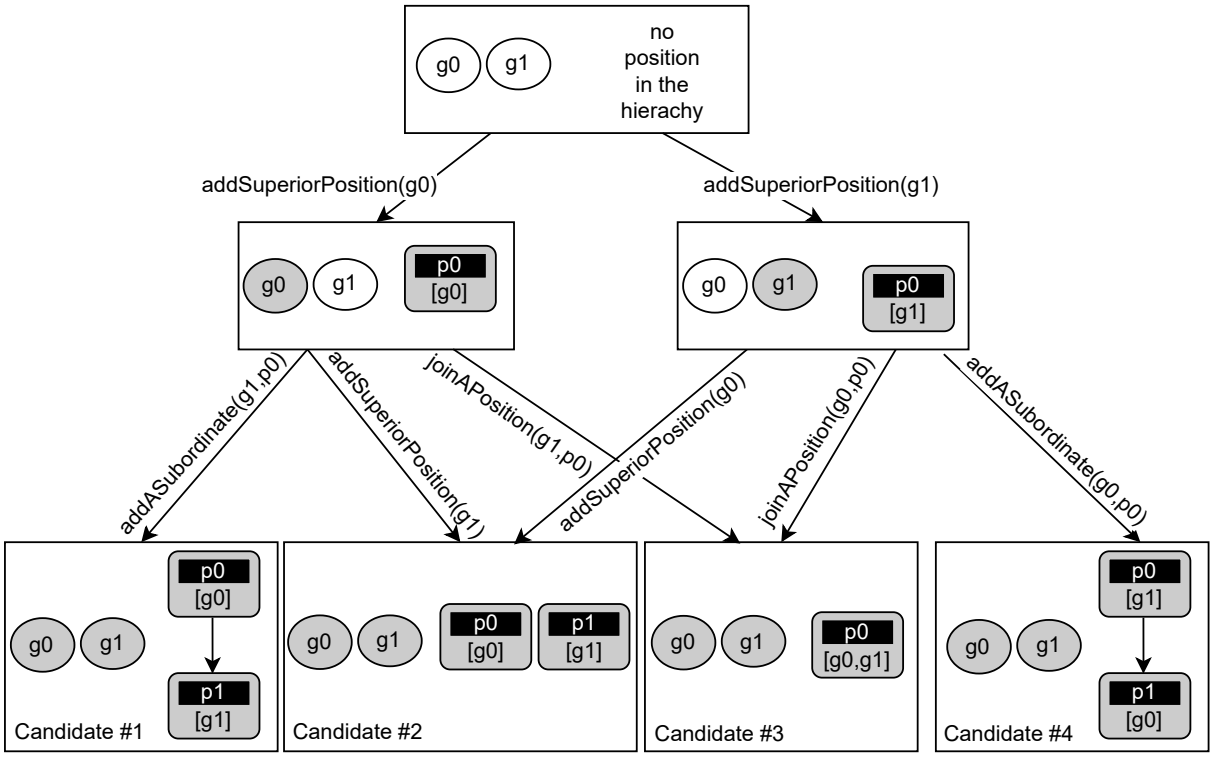
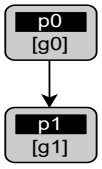
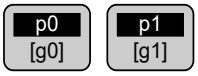
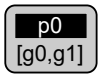
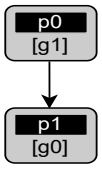


Figure 12 – Step by step of state search with all possible solutions for the given G.

In this example, two states were pruned since they were similar to *Candidate #2* and *Candidate #3*. It has occurred because these candidates are created from more than one intermediary state. For instance, a similar state to *Candidate #2* was pruned because from the righthand side intermediary state another state was created with a position assigned to `g1` and with another position assigned to `g0` on the same hierarchy level. Indeed, the search may produce states with the same distribution of goals and the same relationships between positions which are considered similar states. It should be noted that for pruning, the identification of each position (`p0` and `p1` in this example) is not relevant.

Table 2 shows the generated solutions (the candidates). The order they appear is arbitrary (using the user's preference *UNITARY*).

Table 2 – All the candidates for the given G containing just two goals.

Candidate	Chart	Description
#1		A structure with a superordinate-subordinate relationship. The performer of the goal `g0` is superior of the performer of the goal `g1`. This is a specialised and hierarchical solution for the given set of goals.
#2		A structure with only top superior positions, having no superordinate-subordinate relationships. This is a specialised and flat solution for the given set of goals.
#3		A structure with only top superior positions, having no superordinate-subordinate relationships. This is a generalist and flat solution for the given set of goals.
#4		A structure with a superordinate-subordinate relationship. The performer of the goal `g1` is superior of the performer of the goal `g0`. This is also a specialised and hierarchical solution for the given set of goals.

Considering the organisational attributes, *Candidate #1* and *Candidate #4* have a superordinate-subordinate relationship, presenting hierarchy levels. The other candidates are the flatter structures. As the goals are spread across positions, the structures can also present more specialist organisational positions or more generalist positions. In this example, *Candidate #1*, *Candidate #2* and *Candidate #4*, are equally specialists. *Candidate #3* is more generalist. *Candidate #3* is also the more efficient structure, since it expects only one agent to achieve the goals in G.

4.3.3 Binding agents and positions

GoOrg4Prod binds agents and positions matching their features. It allows the verification of the organisation's feasibility considering the available agents. As presented in Eq. 4.1, the set of features F is formed by the sets of *workloads*, *dataloads* and *skills* (both workloads and agents' *skills*).

To bind agents to positions, *GoOrg4Prod* uses the First Fit algorithm (Algorithm 3). This algorithm is not optimal; it is only being used to demonstrate a possible binding strategy. The available agents in A are settled one-by-one into positions of the set P, using the *skills* in F to determine if the agent a can occupy the position p. As mentioned, in this work, agents and positions are one-to-one relationships. An agent a can be bound to a position p only if a is not bound to another position and if a has all the necessary *skills* that p requires, i.e., $fa(a) \supseteq \{fg(g) \,|\, g \in gp(p) \wedge fg(g) \in S\}$. When the position p is bound with an agent a, then $ap(p) = a$ (Definition 5).

Algorithm 3: FirstFit - binds agents and positions

```
Data: P the set of positions, A the set of available agents
Result: M the set of bound positions and agents
begin
    List M                          // The matching list
    foreach Position p of P         // Foreach position of the structure
     do
        foreach Agent a of A        // Foreach available agent
         do
            if fg(g) ⊆ fa(a)        // 'a' has features of p
             then
                M.add(⟨p, a⟩)       // 'a' matches with p
                A ← A\{a}           // 'a' cannot be used again
                break               // stop inner loop
            end
        end
    end
end
```

4.3.4 Choosing organisations

GoOrg4Prod sorts the generated organisations according to their efficiency ($\eta(o)$), height ($\tau(o)$), generality ($\theta(o)$) and the complementary attributes (the inverse of each attribute). The feasibility attribute ($\kappa(o)$) is not being used to order but to exclude non-feasible organisations.

The complementary attribute is obtained as the complementary percentage. For instance, choosing τ as a criterion means there is a preference for structures with great τ (structures as tall as possible). In this sense, choosing $1-\tau$ means there is a preference for structures with τ near zero (structures as flat as possible).

The user may define multiple criteria which follow a priority order. The organisation in the highest ordering level of the priority criterion is considered the best candidate. If two or more organisations are in the same ordering level, for a priority criterion, then the next priority criterion is used. This process is detailed as follows.

Let c be an ordering criterion ($c \in \{\eta, \tau, \theta, 1-\eta, 1-\tau, 1-\theta\}$) based on the organisational attributes. Let $\gamma \in \Gamma$ be a natural number ($\Gamma \subseteq \mathbb{N}$), representing a priority order for a criterion according to the user's preferences, in which c_1 is the most important criterion for the user. Let $c_\gamma(o)$ be a criterion value for the organisation o according to the priority γ. A partial order relation representing the user's preferences is defined as $\succcurlyeq_\rho$, in which $o \succcurlyeq_\rho o'$ means that o is preferred to o'. If two criteria were set ($\Gamma = \{1, 2\}$), $o \succcurlyeq_\rho o'$ is defined as: $o \succcurlyeq_\rho o' \iff [c_1(o) > c_1(o') \vee (c_1(o) = c_1(o') \wedge c_2(o) > c_2(o'))]$ which can be generalised as stated by the following formula. When two or more organisations share

the same values in all criteria, the first generated candidate is chosen, which would be an arbitrary decision.

$$o \overset{\geq}{\rho} o' \iff \left[\bigvee_{\gamma=1}^{|\Gamma|} c_\gamma(o) > c_\gamma(o') \wedge \left(\bigwedge_{i=1}^{\gamma-1} c_i(o) = c_i(o')\right)\right]$$

For instance, if the most specialist structure ($1-\theta$) is the highest priority criterion (c_1), then three candidates tie in the first criterion when comparing the candidates listed in Table 2. Indeed, candidates #1, #2 and #4 each hold two positions. The first generated candidate (#1) will be arbitrarily chosen if that is the only criterion specified. If there is a second criterion (c_2) and it is the flatter structure ($1-\tau$), then candidate #2 is chosen because it is among the most specialists candidates and it has only one hierarchical level (the flattest).

4.3.5 Computational complexity

GoOrg4Prod uses a blind search technique to generate structures, the breadth-first search algorithm. On the one hand, it is complete and optimal; on the other hand, it is computationally heavy. Considering $n = |G|$, Eq. 4.7 gives the worst estimation of the number of states visited by *GoOrg4Prod* search algorithm. As a worst estimation, the equation does not take into account states that are pruned for being similar to existing ones (with the same goals assigned to similar structures).

$$1 + n + 2^{(n-1)}n!n = O(2^n n!) \tag{4.7}$$

As presented in Algorithm 2, all the possible structures have no positions before the first iteration, so the algorithm does the only suitable transformation for each existing goal: *addSuperiorPosition*. On the next iterations, the algorithm picks the next goal to assign. The algorithm creates one state for each of the three transformations and for each existing position. In this sense, each goal creates a new superordinate (*addSuperiorPosition*), creates a subordinate (*addASubordinate*) of each existing position and joins in each existing position (*joinAPosition*).

As an example, it is considered the case illustrated in Figure 10, in which $|G| = 5$. According to Eq. 4.7, the algorithm may explore 9,606 states as the worst estimation for searching for all possible solutions in this case.

4.4 GoOrg4Prod Results

To illustrate how the best candidate is chosen, the set of goals represented in Figure 10 is considered. It is assumed that the user prefers the most *generalist*, *efficient*

and *flatter* structure, in this priority order. Figure 13 illustrates the three kinds of agents available: (i) the *DB Linkable Elevator*, an agent with the *skills lift* and *database access* for lifting boxes on shelves and to be programmed to access an external database; (ii) the *Box Transporter*, an agent with the *skill move* for moving boxes around the floor; and (iii) the *Pick & Place*, an agent with the *skill pick and place* for picking items from the box and placing them on the conveyor belt (see Appendix B for more details).

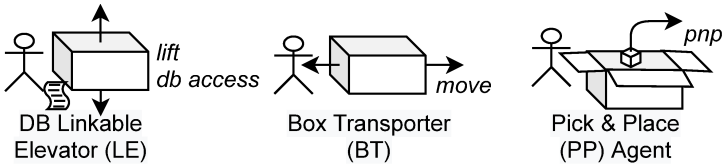


Figure 13 – The available agents.

For the given example, the generation process has produced 1,646 organisational structure candidates.[3] Every organisational structure candidate has all the given goals assigned to positions. These candidates are represented in Figure 14. Each circle represents one or more candidates, since they can be overlapped.

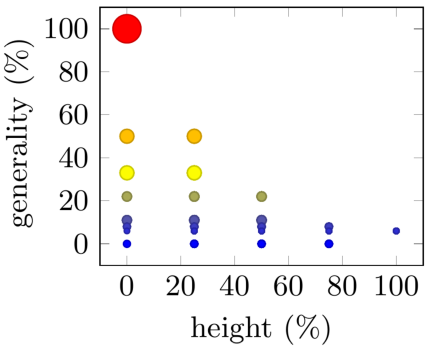


Figure 14 – The generated candidates quantified according to the user's preferences.

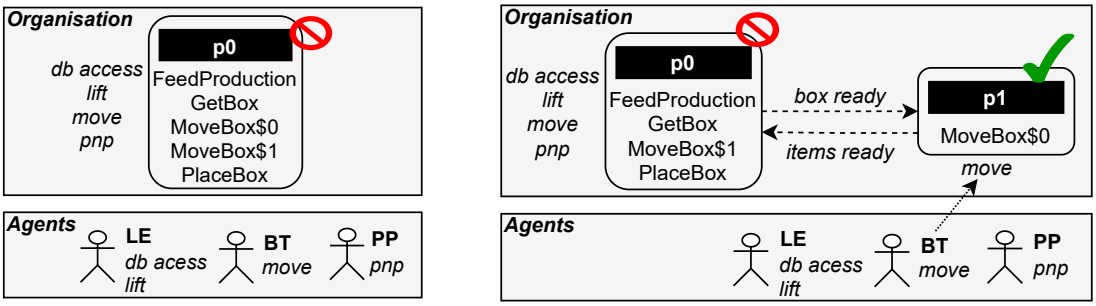
In Figure 14, the *height* axis represents the τ attribute (Eq. 4.3) as a percentage given by 100τ, and the *generality* axis represents the θ attribute (Eq. 4.4) as a percentage given by 100θ. The sizes of the circles represent the efficiency η (Eq. 4.6) as a percentage given by 100η. The biggest circle represents 54.6% of efficiency and the smallest ones represent 10.9% of efficiency.[4] The circle on the top of the *generality* axis and with minimal height represents the best candidate according to the user's preferences. Notice that candidates are ordered by the user's preferences and feasibility does not affect this order. Feasibility is considered only when choosing a candidate, with those that are not feasible being disregarded.

As depicted in Figure 15a, Candidate #1 presents only one organisational position, which would be occupied by one agent that is responsible for achieving all goals. This

[3] This took 1.3 seconds of user time in an Intel® Core™ i7-8550U CPU @ 1.80GHz with 16 GB RAM computer.

[4] The colors also varies according to the size of the circles.

solution has 100% of *generality* since all positions (only one in this case) are assigned to all goals. It also has the minimum height, i.e., one level. Its efficiency is the highest for this problem, 54.6%, which is the total effort (13.1 hours) over the baseline (24 hours). Since all *dataloads* associated to the assigned goals are loopbacks, Candidate #1 has no need for exchanging data among agents. Although it is the best candidate according to the user's preferences, this solution is not feasible since there is no available agent that has all the four *skills* (*db access*, *lift*, *move* and *pnp*) required by the position **p0**.

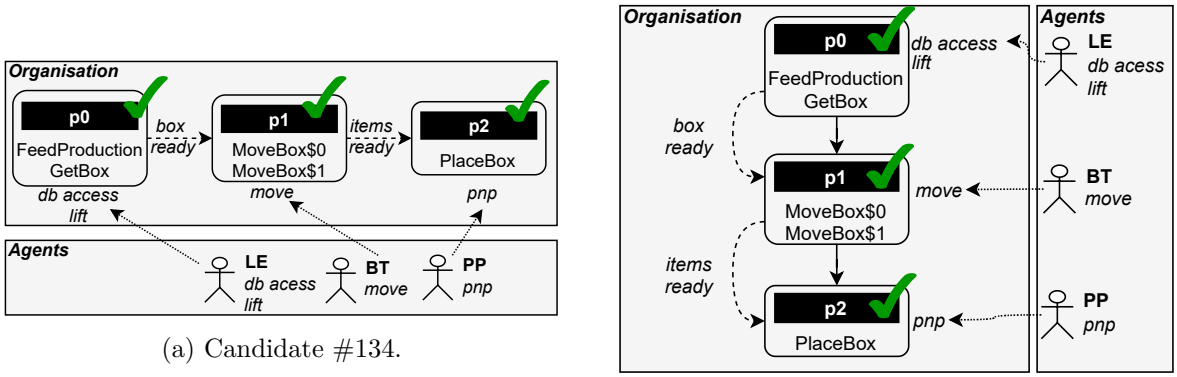


(a) Candidate #1.

(b) Candidate #2.

Figure 15 – The two best structures, which are not 100% feasible.

Figure 15b depicts Candidate #2, the second best solution according to the user's preferences. The two dashed arrows represent *dataloads* that the performer of one goal must send to the performer of another goal. Like Candidate #1, Candidate #2 has the lowest height, since the two generated positions are in the same hierarchy level. However, comparing to Candidate #1 its *generality* was reduced since each position has some degree of specialisation. Additionally, its efficiency is reduced since it has two positions instead of only one, which increases its idleness. This candidate is 50% feasible because only position **p1** has an agent able to perform it. Although candidates #1 and #2 are preferred according to the user's preferences, neither of them is 100% feasible.



(a) Candidate #134.

(b) Candidate #446.

Figure 16 – Two feasible structures for the given example.

Figure 16 depicts two feasible candidates for the given example. Candidate #134 is one of the flattest structures, since it presents just one hierarchy level. However, its

generality is even more reduced compared to the candidates #1 and #2. Besides, compared to the best candidates, it has more positions to be occupied, which drives to an undesirable lower efficiency. Although it is far from the ideal solution (Candidate #1), taking the given available agents and the user's preferences, this candidate is *GoOrg4Prod* first choice since it is the first one that is 100% feasible.

Figure 16 gives another example of a feasible solution: Candidate #446. The *generality*, *efficiency* and *feasibility* attributes are the same as for Candidate #134. However, according to the user's preferences, this solution is not as good as Candidate #134 because there are hierarchical relationships (represented as solid line arrows), which makes this solution less flat. This candidate could be the first choice if the user's preferences were set for the tallest and most general and most efficient solution.

The candidate #1646 (Figure 17) is at the bottom of the list to be chosen, it is the worst candidate for the user's preferences. Its *generality* is very low, since only the positions **p0** and **p2** are interchangeable. Its efficiency is also low due to the high number of positions ($|P| = 5$ for this candidate). Its height is also far from what the user prefers (it has 5 levels). This candidate is 100% feasible assuming that there are 2 units of the agents *Box Transporter* and *DB Linkable Elevator*. In the case of having only one unit of each kind of agent, this candidate is just 60% feasible.

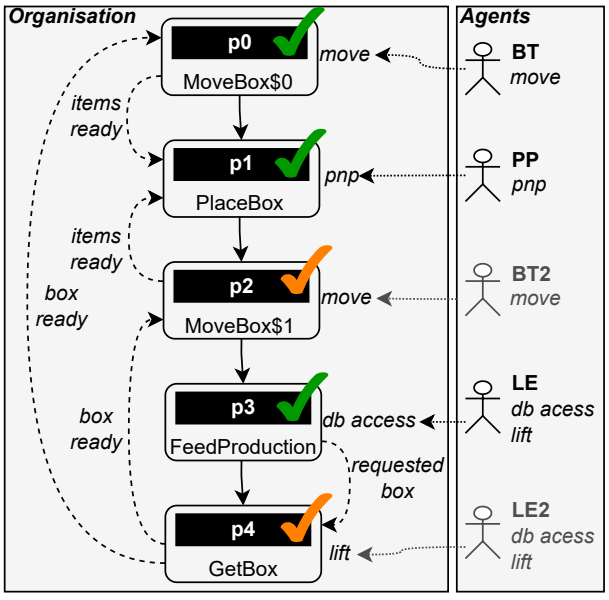


Figure 17 – Candidate #1646

As demonstrated, the three transformations of *GoOrg4Prod* can produce structures with superordinate-subordinate relationships as found in classic organisational charts.[5] A superior (superordinate) is often responsible for some kind of coordination of its subordinates. A superordinate-subordinate relationship may imply, for instance, power and

[5] Hierarchies are one of the existing MAS organisational structure paradigms (Horling and Lesser, 2004).

accountability of a position to another (Kilmann et al., 2010). The *addSuperiorPosition* transformation can create the first position of an organisation, as well as other independent pairs of the first position, potentially producing a forest of hierarchies. A forest may represent multiple departments or even a collection of organisations. Additionally, the transformations *joinAPosition* and *addASubordinate* assign a goal to existing positions and to new organisational positions. Combining these transformations, it is possible to have very hierarchical trees in which there are long chains of superordinate-subordinate relationships (e.g., candidates #446 and #1646). In contrast, it is also possible to have very flat structures with no superordinate-subordinate relationship in an organisational chart (e.g., candidates #1, #2 and #134).

5 GoOrg4DSN: A Specialisation for the Distributed Sensors Network Domain

GoOrg4DSN is another specialisation of GoOrg. In this work, the term *GoOrg4DSN* refers to both a model extension and an implementation that can generate organisation descriptions for the DSN domain. It addresses the problem of Distributed Sensor Networks (DSN), introduced by Lesser et al. (2003). *GoOrg4DSN* is concerned with the generation of organisational structures for tracking one or more moving targets in an area.[1] The objective is to detect targets and follow the target by selecting sensors to get the most accurate coordinates of the target as it moves. A network of sensors that are fixed in geographical positions provides the coordinates of targets as the sensors' signals are triangulated. The resolution track depends on how the sensors used in the triangulations are distributed along the area and their distance from the target. Although restricted, the processing capacity of the sensors can be used to host agents' processes. Other aspects such as the low-speed and unreliable communication, and the need to select communication channels to avoid collisions bring additional challenges.

GoOrg4DSN is based on the approach proposed by Horling and Lesser (2008). In their work, the area covered by the sensors is divided into sectors as groups of sensors. The approach defines three roles: *Sensor Agent*, *Sector Manager* and *Track Manager*. The *Sensor Agent* performs scans for targets and reports detections. The *Sector Manager* performs many tasks: it sends to *Sensor Agents* of the sector a scanning schedule, defines communication channels, combines data from sensors to identify a target and communicates with other *Sector Managers*. It also elects a sensor to play the role *Track Manager* when a new target is identified. The *Track Manager* picks sensors to keep updated about target coordinates, reporting this information to the *Sector Manager*. All the sensors (agents) enact the *Sensor Agent* role. In Horling and Lesser (2008)'s implementation, the geographical area of each sector is arbitrarily defined according to the number of sensors by sector, which varies from 5 to 10 units, and also according to the density of sensors. Each sector has a *Sector Manager* which is user-defined a priori. When a new target is identified, as the *Sector Manager* is usually busy with its duties, it prefers to elect other agents rather than itself to be *Track Manager*.

For this problem, *GoOrg4DSN* uses the Multi-Agent Oriented Programming (MAOP) approach (Boissier et al., 2013). In MAOP, autonomous entities are modelled as agents,

[1] An implementation of *GoOrg4DSN* is available at `https://github.com/cleberjamaral/GoOrg4DSN`

non-autonomous entities such as environmental tools and resources exploited by the agents are modelled as artefacts, and coordination mechanisms are modelled as organisations. Considering the solution proposed by Horling and Lesser (2008), *GoOrg4DSN* has two main conceptual changes which are the absence of explicit roles and the procedural task of scanning the area placed on another level of abstraction. In *GoOrg4DSN*, the scanning task that Horling and Lesser (2008) assigned to *Sensor Agents* are instead executed by non-autonomous entities (artefacts).[2] This MAOP approach is represented in Fig 18.

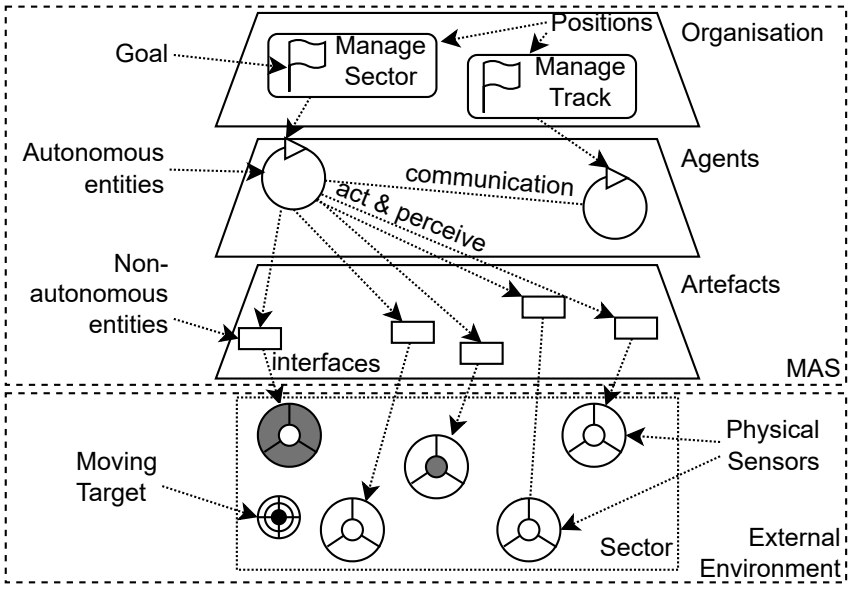


Figure 18 – A MAOP approach for the DSN domain.

In the top dashed rectangle, the MAS is shown in three dimensions: organisation, agents and artefacts. The agents interact among themselves and also act on and perceive the artefacts, which are interfaces to the external environment. In this domain, the external environment is formed by physical sensors situated in a delimited sector which can be visited by moving targets.

The tasks performed by the agents change according to the environment dynamism caused by entry, exit and movement of targets within the monitored area and across sectors. Indeed, such dynamism changes the goals of the organisation. For instance, an agent that manages a sector with no detected targets has to wait for an event to be communicated by the sector artefacts or by agents that manage other sectors. When a target is detected, a goal for tracking that particular target is created. In this sense, a change to the set of organisational goals brings the need to redesign the organisation.

The dashed rectangle at the bottom of Figure 18 represents a sector that contains five geographically distributed sensors. The central sensor with a shaded inner circle

[2] In Horling and Lesser (2008)'s approach, *Sensor Agents* have some autonomy when they negotiate their schedule with the *Sector Manager*. However, they mainly execute scans for targets, which are procedural tasks.

represents the device that was a priori defined to host the agent to manage the sector. Figure 18 is illustrating a situation in which a target was detected. The sensor with a shaded outer circle was elected to be the manager of this tracking. The rest of the sensors are devices that are hosting agents that are not yet part of the organisation. Indeed, the network has distributed processing along with sensors. In *GoOrg4DSN*, each sensor hosts an artefact process and also an agent process, which is often a stand-by agent. In this sense, most of the sensors host an available agent which can be bound to an organisational position, becoming an organisation's member.

Figure 19 illustrates how this specialisation extends GoOrg. To represent that agents become busy while managing a sector and managing a tracking, the goals have a feature of kind *workload*, which have an identification (referring to an *intent*) and an expected effort to execute the workload. The attribute *efficiency* is calculated using *workloads* expected efforts. This implementation specifies two types of efforts: *manage_sector* and *manage_track*. The agents that manage sectors are defined a priori by the user according to signal ranges. Sector managers are usually the agents that can better reach sector sensors, and also reach other sector managers. To represent which sensor is previously defined as a manager, the specified agents have a feature of kind *intents* recording they were set to manage a sector. All agents also have a feature of kind *sector* to record which sector they belong to. This includes the agents that occasionally are not members of the organisation, but are available and may become a tracking manager if necessary. A goal to track a target also has a feature of kind *sector*, allowing the identification of the sector that is handling the tracking. In the running system, the agent in charge of *manage_sector*, in the sector that is handling a tracking, must choose an agent to be assigned to manage this tracking. In this sense, the chosen agent occupies the position that GoOrg4DSN has synthesised for tracking the corresponding target.[3] By the attribute *nearness*, it is possible to check whether an agent and a target are in the same sector or not.

5.1 GoOrg4DSN elements

GoOrg4DSN specifies three features to constrain the organisational design while synthesising positions, arranging hierarchies and binding agents to positions. The added features are: (i) *intents*, which are associated with agents; (ii) *workloads*, which are associated with goals; and (iii) *sectors* which are associated with both goals and agents.

The agents have associated *intents*. These are used to inform *GoOrg4DSN* about the a priori defined usage of the agents, regarding the ones that are chosen to manage a

[3] It is assumed that if the target is simultaneously detected by sensors from different sectors, the managers of the corresponding sectors will negotiate which sector should be associated with the respective *manage_track workload*.

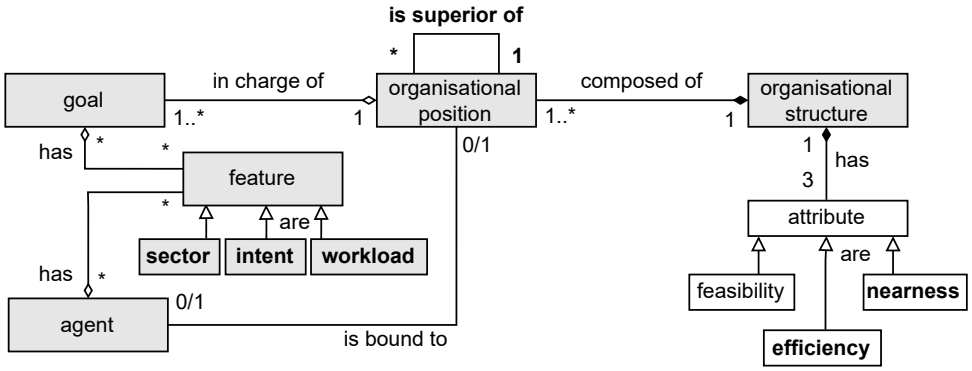


Figure 19 – GoOrg4DSN model.

sector. An *intent* is formally defined as a singleton tuple containing a symbol, as follows.

$$t : \langle symbol \rangle,\ t \in T$$

A goal may have a *workload* required to achieve it. A *workload* w represents a demanded effort $e \in \mathbb{R}^+$ associated with an *intent* $t \in T$. A *workload* is formally defined as a tuple of a symbol and a real positive number. The function wg maps goals to their *workloads*, as follows.

$$w : \langle t, e \rangle,\ t \in T,\ e \in \mathbb{R}^+,\ w \in W$$

$$wg : G \rightarrow 2^W$$

Both agents and goals are associated with *sectors*. A *sector* refers to a group an agent or a target belongs to. The *sector* feature is a priori associated with all available agents (the ones that are defined to host a manager of a sector and the ones that are defined to host available agents). In case of goals, the *sector* feature is associated with the goals *manage_sector* according to the *sector* that is handling the tracking. A *sector* is formally defined as a singleton tuple of a symbol, as follows.

$$c : \langle symbol \rangle,\ c \in C$$

The sets of *workloads* (W), *intents* (T) and *sectors* (C) are subsets of the set of features, as follows.

$$F = W \cup T \cup C \tag{5.1}$$

Like *GoOrg4Prod*, *GoOrg4DSN* considers that organisational positions may have superordinate-subordinate relationships. This only occurs in the case of a superordinate being assigned to the *manage_sector* workload and the subordinate being assigned to

the *manage_track* workload. The function $sp(p)$ that records the superordinate of the position p and the organisational structure o are defined as for *GoOrg4Prod* (Section 4.1).

GoOrg4DSN also adds a design parameter to prevent positions from being assigned to a sum of *workloads* efforts that surpass 100%. It is defined as $\phi_p \in \mathbb{R}^+$, representing the maximum *workload* allowed on each position.

5.2 GoOrg4DSN Added Attributes

GoOrg4DSN defines new attributes of an organisational structure. From the added feature *workload*, the *efficiency* of a structure is calculated, which was already defined in Eq. 4.6. From the feature *sector*, the *nearness* of a structure is calculated.

Considering that a structure is a forest of hierarchies, *nearness* refers to how much similar the positions of every tree in the forest are in terms of their *sectors*. This gives an idea of how geographically near the positions are. For simplicity, this work does not specify distances between sensors. Instead, *GoOrg4DSN* considers that sensors of the same *sector* are close to each other and sensors of different *sectors* are far away from each other. Formally, the *nearness* of an organisational structure o is represented as $\rho(o)$, a real number in the range [0,1] (Eq. 5.2). The nearness reduces when a position has a different *sector* compared to its superordinate. The maximum *nearness* ($\rho(o) = 1$) occurs when every hierarchy (tree) is formed by positions assigned to goals of the same *sector*.

$$n(p) = \begin{cases} 0 & \exists g \in gp(p)\, \exists g' \in gp(sp(p)),\ fg(g) \cap C \neq fg(g') \cap C \\ 1 & \text{otherwise} \end{cases}$$

$$\rho(o) = \frac{\sum_{p \in P} n(p)}{|P|} \tag{5.2}$$

5.3 GoOrg4DSN Processes

GoOrg4DSN generates and chooses organisations in a chain of three processes: (i) organisational positions are synthesised and structures are generated in a process that searches the space with all possible organisations according to the supported transformations; (ii) the positions of generated structures are bound to the given agents and the feasibility of the organisation is calculated; and (iii) an organisation with positions and bound agents is chosen. Compared to *GoOrg4Prod*, the process for preparing goals is not necessary for the DSN domain, since for this domain the goals are not divisible. Figure 20 illustrates the referred processes.

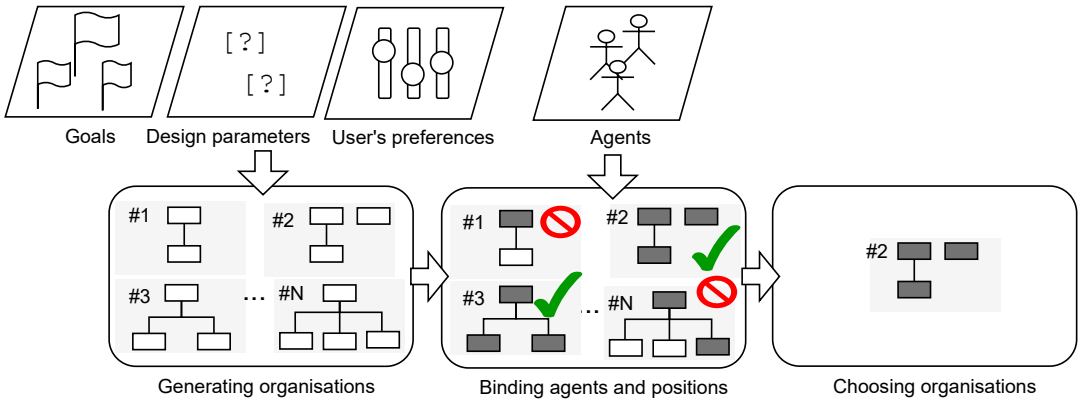


Figure 20 – The three processes of *GoOrg4Prod*.

5.3.1 Generating organisations

GoOrg4DSN uses the same state-space search algorithm that *GoOrg4Prod* uses. It assigns goals to positions step by step, follows a cost function based on the user's preferences, and generates all possible solutions according to its constraints. The algorithm for generating successors states is similar to Algorithm 2, the difference is in the constraints of the transformations.

GoOrg4DSN applies three structure transformations for every $g \in G$ into two stages. In the first stage, a transformation assigns the goals $g \in G$ that are associated with the *workload manage_sector* to new top superordinate positions. In the second stage, each remaining $g \in G$ is the subject of the following structure transformations: (i) if g is associated with a *workload manage_sector*, it is assigned to a new top superordinate position creating a new tree in the forest; (ii) if g is associated with a *workload manage_track*, it is assigned to new subordinate positions (for each position associated with a *manage_sector workload*); and (iii) if g is associated with a *workload manage_track*, it is assigned to every existing position (the ones associated with a *manage_track* or with *manage_sector workloads*). These transformations are presented as follows.

In the $addSuperiorPosition(g)$ transformation, a goal associated with the *workload manage_sector* is assigned to a new *superordinate position*. Indeed, in the running system, an agent that manages a sector performs some actions that imply authority, a *workload* $\pi_1(fg(g)) = manage\_sector$ is only assigned to superordinate positions. The new position is added to the set of existing positions P. The assigned goal g is recorded by the function gp, the first element (π_1) of a *workload* tuple is recorded by the function fg and the function sp records that p has no superior. The element π_1 of a *workload* refers to an *intent*, which can be either *manage_sector* or *manage_track*. The only feature that is recorded by fg is the *intent* inside of the *workload*, since it is the only relevant data for further processes. This transformation from the structure o to the structure o' is formalised as follows.

$$o = \langle G, A, P, F, gp, fg, fa, ap, sp, wg \rangle$$
$$T' = \{\pi_1(w) \,|\, w \in wg(g)\}$$
$$manage\_sector \in T'$$
$$p = new\ Position$$
$$addSuperiorPosition(g) \frac{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}{o' = \langle G, A, P \cup \{p\}, F, gp \cup \{p \mapsto \{g\}\},}$$
$$(fg \backslash \{g \mapsto fg(g)\}) \cup \{g \mapsto (fg(g) \cup T')\}, fa, ap, sp \cup \{p \mapsto \epsilon\}, wg \rangle$$

In the $addASubordinate(g, p')$ transformation, a goal associated with the *workload* *manage_track* is assigned to a new *subordinate position*. The superior position p' must be associated with a *manage_sector* workload. The new position is added to the set of existing positions P. The assigned goal g is recorded by the function gp, the first element of the *workload* tuple is recorded by the function fg, and the function sp records p' as superior of p. This transformation from the structure o to the structure o' is formalised as follows.

$$o = \langle G, A, P, F, gp, fg, fa, ap, sp, wg \rangle$$
$$T' = \{\pi_1(w) \,|\, w \in wg(g)\}$$
$$manage\_track \in T'$$
$$\exists g' \in gp(p'), manage\_sector \in \{\pi_1(w') \,|\, w' \in wg(g')\}$$
$$p = new\ Position$$
$$addASubordinate(g, p') \frac{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}{o' = \langle G, A, P \cup \{p\}, F, gp \cup \{p \mapsto \{g\}\},}$$
$$(fg \backslash \{g \mapsto fg(g)\}) \cup \{g \mapsto (fg(g) \cup T')\}, fa, ap, sp \cup \{p \mapsto p'\}, wg \rangle$$

In the $joinAPosition(g, p)$ transformation, a goal associated with the *workload* *manage_track* is assigned to one of the existing positions. This transformation is applied if g is not associated with a *manage_sector workload* which prevents to have a position responsible to manage more than one *sector*. The assigned goal g is recorded by the function gp, and the first element of the *workload* tuple (π_1) is recorded by the function fg. This transformation from the structure o to the structure o' is formalised as follows.

$$o = \langle G, A, P, F, gp, fg, fa, ap, sp, wg \rangle$$
$$T' = \{\pi_1(w) \,|\, w \in wg(g)\}$$
$$manage\_sector \notin T'$$
$$joinAPosition(g, p)$$
$$o' = \langle G, A, P, F, (gp \backslash \{p \mapsto gp(p)\}) \cup \{p \mapsto (gp(p) \cup \{g\})\},$$
$$(fg \backslash \{g \mapsto fg(g)\}) \cup \{g \mapsto (fg(g) \cup T')\}, fa, ap, sp, wg \rangle$$

5.3.2 Binding agents and positions

GoOrg4DSN also uses the First Fit algorithm (Algorithm 3). The set of features F is formed by the sets of *workloads*, *sectors*, *intents* and the first element of *workload* tuples. The agents and positions are matched by *intents* and the first element of *workload* tuples. It worths to mention that, *GoOrg* extensions implement a binding process just to check the organisation's feasibility. At running conditions, the binding between agents and positions in charge of *manage_track* are defined by positions in charge of *manage_sector* (which are superordinates of their *sectors*).

5.3.3 Choosing organisations

GoOrg4DSN sorts the generated organisations according to their efficiency ($\eta(o)$), nearness ($\rho(o)$) and their complementary attributes (the inverse of each attribute). If the feasibility ($\kappa(o)$) is not 100%, the organisational structure is not considered. The formula used for choosing structures by the user-defined criteria is the same as *GoOrg4Prod* (Section 4.3.4).

5.3.4 Computational complexity

The worst estimation of the number of search states visited is defined in Eq. 4.7, the same as *GoOrg4Prod*. Yet, *GoOrg4DSN* prunes more states, for instance, constraining hierarchies in which a goal associated with *manage_sector workload* is assigned to a subordinate position and when a goal associated with *manage_track workload* is triggering the creation of superordinate positions. For comparison, for a scenario with $|G| = 5$, *GoOrg4Prod* visits 9,606 states and generates 1,646 candidates. For the same number of goals, *GoOrg4DSN* visits only 80 states and generates just 8 candidates.[4] However, the number of states to visit grows exponentially. For more complex scenarios, the number of states and candidates may become too large and not viable for the search approach used

[4] This took 1 second of user time on an Intel® Core™ i7-8550U CPU @ 1.80GHz with a 16 GB RAM computer.

by *GoOrg4DSN*. To exemplify, for $|G| = 7$ *GoOrg4DSN* generates 612 candidates and for $|G| = 8$ it generates 5812 candidates.[5]

5.4 GoOrg4DSN Results

To illustrate how *GoOrg4DSN* generates organisational structures, it is considered an area divided into four sectors identified by the four geographical quadrants (Figure 21). Each sector contains five sensors.

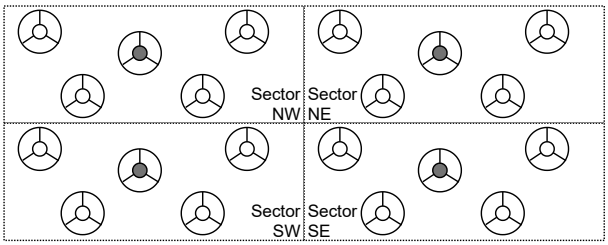


Figure 21 – A motivating scenario with four sectors, each one with 5 sensors.

The user's preferences are based on Horling and Lesser (2008)'s approach. Due to computational and communication restrictions of the sensors that host agents, it is preferred *idle* structures (less *efficient*). With respect to the computational limits of the sensors, in the DSN domain it is important to avoid assigning multiple goals to the same position. Both *efficiency* and *generality* attributes defined for *GoOrg4Prod* domain helps to measure the distribution of goals in a structure, but for *GoOrg4DSN* *efficiency* is better. In the case of *generality*, the preference for more specialist structures could result in structures that avoid assigning the *manage_track* goal to positions that are assigned to *manage_sector*, however, it would not avoid having multiple *manage_track* goals to the same position. On the other hand, the preference for more idle structures using the attribute *efficiency* avoids both of the mentioned situations. In other words, it is preferred to avoid assigning *manage_track* to a position in charge of *manage_sector* or to a position that is already in charge of another *manage_track*. Besides, to optimise the communication among sensors, it is preferred a structure with a high *nearness*. As both *manage_sector* and *manage_track* goals are associated with a *sector* feature, a structure with the higher *nearness* has all superordinate-subordinate relationships between positions assigned to goals associated with the same *sector*. For instance, if there is a target in the sector *se*, according to this criterion, the position assigned to *manage_track* should be a subordinate of the position assigned to manage the sector *se*, as they are physically close to each other.

To show how *GoOrg4DSN* is used in such a dynamic scenario, in this section different situations in terms of targets that should be tracked are presented. First, it is

[5] This respectively took 1.1 seconds and 1.7 seconds on the same mentioned computer.

considered a situation in which no sensor is detecting any target. It is only necessary to manage each sector, as illustrated by Figure 22.

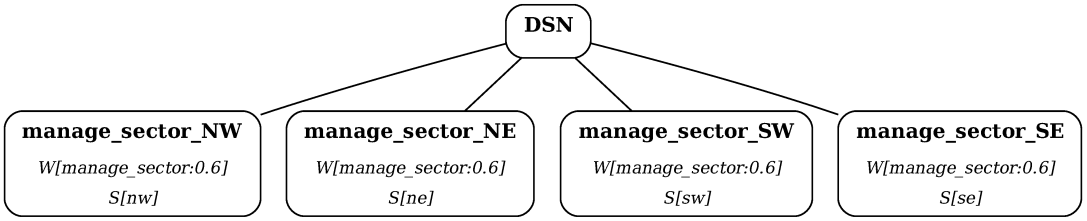


Figure 22 – The set of goals in which no target is being detected.

For this situation, *GoOrg4DSN* generates only one organisational structure, as it is the only solution. Indeed, the goals to be assigned are just the ones associated with *manage_sector workloads*. As stated, each goal associated with *manage_sector workload* should be assigned to a top superior position. It is specified that a *manage_sector workload* makes an agent 60% (0.6) busy with this duty. As a result, for every goal, a tree with only one position is created in the forest (organisational structure), as illustrated by Figure 23. This is a standby situation for most of the sensors, i.e., most of them are just scanning the area as there is no tracking underway.

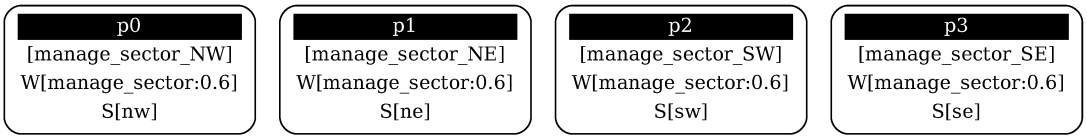


Figure 23 – Candidate #1 (unique) when there is no target being detected.

In another situation, it is considered that the sensors have detected a target in the southeast (*se*) sector. For this situation, there are five goals to be achieved by the organisation: manage each of the four sectors and manage the tracking, which is identified by *manage_track_1*. Figure 24 illustrates the set of goals for this situation. It is specified that an agent that is managing a tracking is 20% busy with this goal. In this sense, it is possible to have the same agent managing a sector and managing a target tracking. However, it would create a less *idle* structure, which is not desired according to the user's preferences.

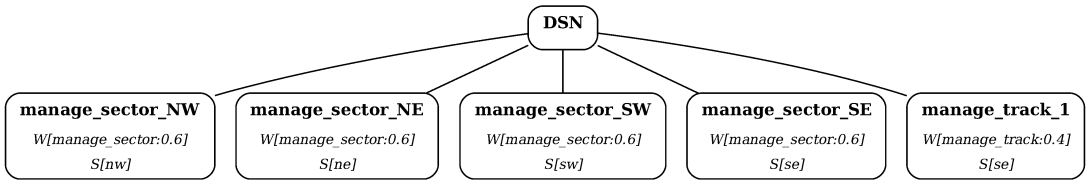


Figure 24 – The set of goals in which one target is being detected.

For the scenario with one target being detected on the sector *se*, *GoOrg4DSN* has generated 8 candidates. The best candidate is illustrated by Figure 25. It considers

the user's preference for delegating the *manage_track_1* goal to a sensor in the same sector, and that it is better to not assign this goal to an agent that is managing a sector. This candidate has a new position, a subordinate of the position in charge of managing the sector *se*. The second best solution generated has the sector manager also assigned to *manage_track_1*, which is not so good considering the user's preference because the agent would be busier. The rest of the candidates are even worse because they suggest delegating the *manage_track_1* goal to agents of other sectors and also assigning this goal to the managers of other sectors.

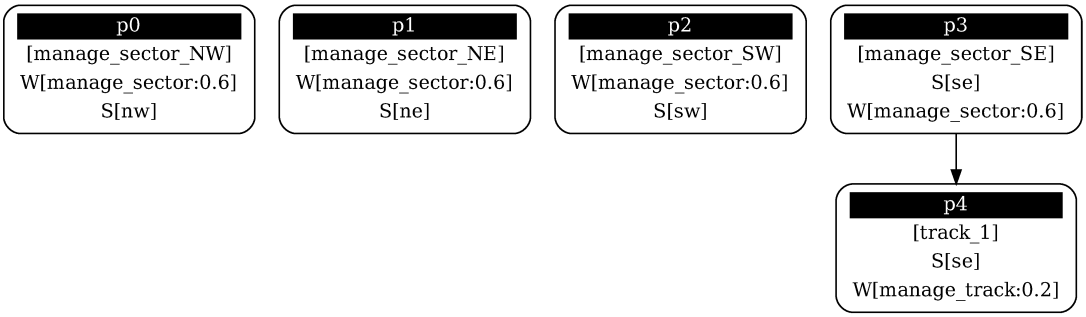


Figure 25 – Candidate #1 for the scenario with 1 target being detected.

To illustrate a situation with multiple targets, it is considered a scenario in which three targets are detected, two on the sector *se* and one target on the sector *nw*. For this scenario with 7 goals (4 goals to manage sectors and 3 goals to manage tracks), *GoOrg4DSN* has generated 612 candidates. According to the user's preferences, the best candidate is the one illustrated by Figure 26. In this candidate, two positions are created as subordinates of the position in charge of managing the sector *se*, and one position is created as a subordinate of the position in charge of managing the sector *nw*. For this situation of multiple targets, compared to candidate #1, the second best solution found differs because it has only one subordinate of the position in charge of managing the sector *se*. This position is assigned to track two objects which makes it busier, and this structure is less preferred according to the user's preferences, compared to Candidate #1. The rest of the candidates are even worse according to the user's preferences because *manage_track* goals are assigned to positions of other sectors and also because existing positions accumulate duties.

As seen, this scenario is very dynamic as targets move along the scanned area. In the case of a target passing from one sector to another, it is assumed that the goal to track this target will have its feature *sector* changed to the other sector, which changes the goals of the organisation and requires a redesign. Besides, as illustrated in different scenarios, in the case of a target entering or leaving the area, a redesign is also necessary. In case of an agent failure, other agents (possibly some of the stand-by agents) may occupy the position, which is done by a simple re-allocation. However, a change in agents' availability may also drive to a structure-switching, for instance, if there are 4 targets

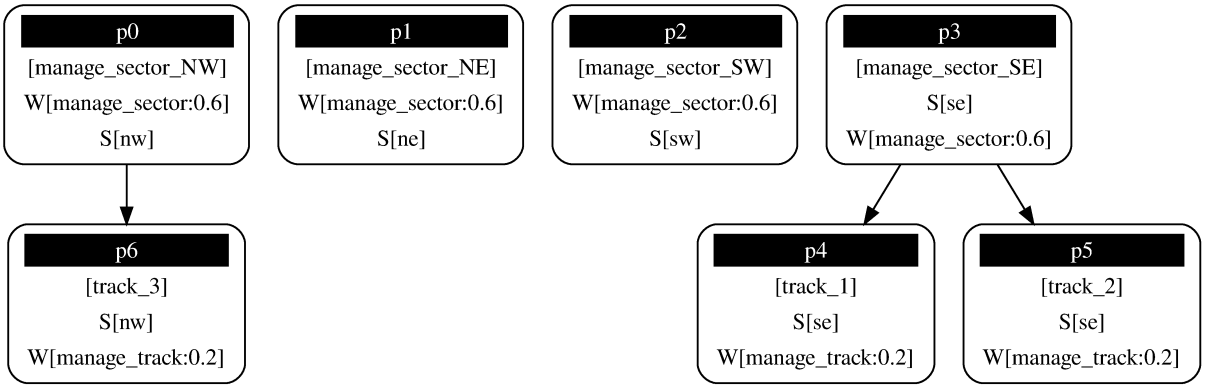


Figure 26 – Candidate #1 for the scenario in which three targets are being detected.

being detected in a sector and one of the agents fail. A new binding between agents and positions would revise the idlest structure as not feasible, making other candidates more suitable in this condition. Section 7.1 presents a discussion about different organisation adaptation procedures.

6 GoOrg: Implementation

This chapter goes over GoOrg implementation in depth. It shows the tools and programming languages that were used to implement GoOrg and its software architecture. It also describes the input and output formats, and explains how to extend GoOrg for specific domains.

6.1 Tools and Programming Languages

GoOrg is implemented in Java, an object-oriented programming language, that is interpreted by a Java Virtual Machine (JVM). A JVM is available in many computing platforms, which allows running Java applications in many computer architectures. GoOrg is compatible with Java 8 or superior.[1] For compiling GoOrg, Gradle is used. Gradle is an automation software development tool, often used to build and manage project dependencies. The Gradle script is defined in a file called *build.gradle* which is usually written in a Java interoperable language (such as Groovy, Kotlin and Scala). For managing dependencies, the Gradle tool can retrieve packages from Apache Maven repositories.[2] Apache Maven repositories are very popular, most of the libraries written for Java are available in Maven-compatible repositories.

As input, GoOrg implementation uses a $\mathcal{M}oise^+$ like organisation specification (Hübner and Sichman, 2003). $\mathcal{M}oise^+$ is an organisational model in which organisations can be defined, including their goals which are specified in form of an organisational scheme. The organisation specification is defined in an XML file, which is a popular format for storing and transmitting arbitrary data. Since GoOrg may add some features to the goals, the $\mathcal{M}oise^+$ XML format should be extended according to the extra information added to the goals. $\mathcal{M}oise^+$ also does not expect a set of available agents as GoOrg does. In this sense, another extension of the XML is the element *available-agents* and the nested elements *agent* to indicate each available agent and its features, according to the domain.

GoOrg outputs organisational structures in form of graphs, which can be rendered using Graphviz.[3] The graphs may represent organisational charts (organigrams) of hierarchical structures and other kinds of structural shapes. Graphviz is an application that

[1] The implementation was tested with Java 11, 13 and 17.

[2] The mentioned tools and more information about them are available at `https://gradle.org/`, `https://kotlinlang.org/`, `https://groovy-lang.org/` and `https://scala-lang.org/` and `https://maven.apache.org/`.

[3] Graphviz is available at `https://www.graphviz.org/`.

produces graphical visualisation in many formats using as input a text language description of elements such as shapes and arrows. GoOrg also generates a list containing all the generated structures and other relevant data such as the bindings between agents and positions, and a JaCaMo project file (JCM) for launching the agents that are bound with organisational positions (Boissier et al., 2016).

6.2 GoOrg Implementation Architecture

GoOrg instantiation starts in the class *OrganisationApp*. The generation process considers an XML file as input that can be provided by the command line. The application uses the class *OrganisationXMLParser* to get the inputs from a given XML file. In this case, GoOrg parses the *functional-specification* element of the XML file defined by $\mathcal{M}oise^{+}$. This element has other nested elements such as goals and plans. GoOrg basically gets the *goals* in the original format defined by $\mathcal{M}oise^{+}$, but with the addition of features. Listing 6.1 illustrates a $\mathcal{M}oise^{+}$ like scheme for the *Feed Production Scenario with Three Goals*. The goal identified as *GetBox*, and the goals *MoveBox* and *PlaceBox*. For instance, as seen in the XML notation, the goal *GetBox* has the *workload lift* associated, and also the *dataload box ready*. The *workload* and *dataload* elements are part of the necessary extension of the $\mathcal{M}oise^{+}$ specification.

Listing 6.1 – A given set of goals as a $\mathcal{M}oise^{+}$ scheme.

```
<functional-specification>
  <scheme id="scheme">
    <goal id="GetBox">
      <workload id="lift" value="8.00"/>
      <dataload id="box␣ready" value="8.00" recipient="MoveBox"/>
      <plan operator="sequence">
        <goal id="MoveBox">
          <workload id="move" value="8.00"/>
          <dataload id="items␣ready" value="8.00"
            recipient="PlaceBox"/>
        </goal>
        <goal id="PlaceBox">
          <workload id="pnp" value="8.00"/>
        </goal>
      </plan>
  </goal>
  </scheme>
</functional-specification>
```

The list of available agents is provided by XML elements nested within the *available-agents* element. Listing 6.2 illustrates a given set of agents, which was created for *GoOrg4Prod* domain. In this domain, the agents have *skills* which is used to match agents and posi-

tions, according to their associated *workloads*. In this sense, the XML elements must be defined according to the domain.

Listing 6.2 – A given set of available agents

```
<available-agents>
  <agent id="bt">
    <skill id="move"/>
  </agent>
  <agent id="pp">
    <skill id="pnp"/>
  </agent>
  <agent id="ie">
    <skill id="lift"/>
  </agent>
</available-agents>
```

Figure 27 shows the class diagram of GoOrg. Near the top of the diagram, it is illustrated the starting point class, *OrganisationApp*, and its relationships, for instance with the class *OrganisationXMLParser*. The parsed goals are instantiated as *GoalNode* objects. These objects are associated with the *GoalsTree* class, which has information about node relationships. The parsed agents are instantiated as *Agent* objects, which are associated to an *AgentSet* object.

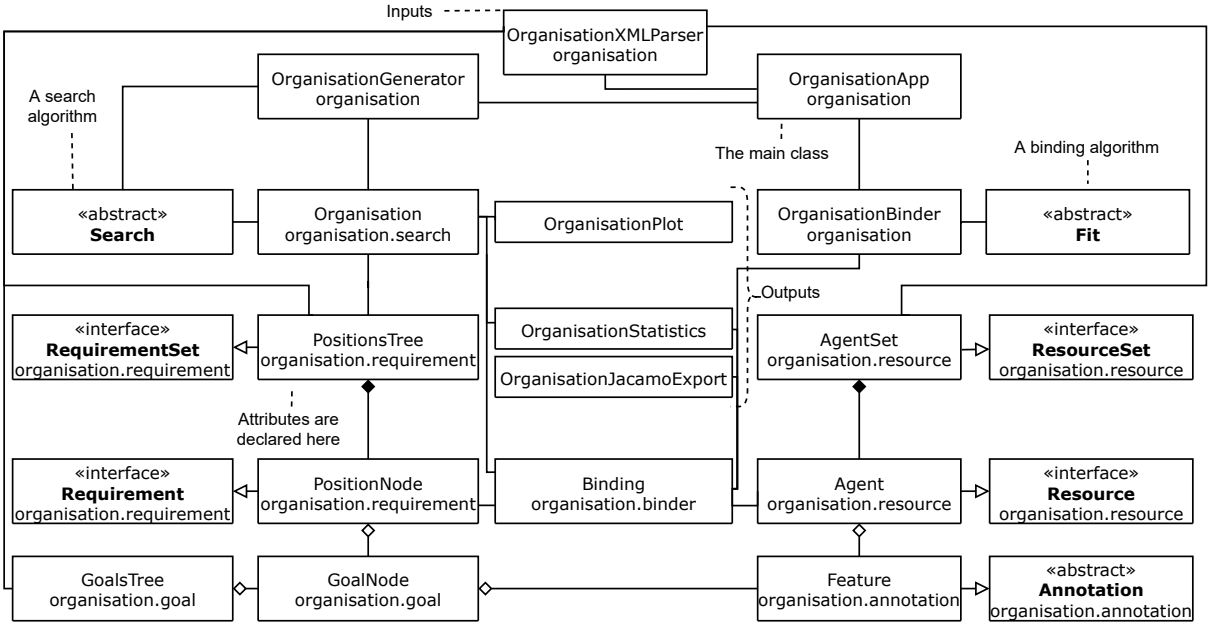


Figure 27 – GoOrg simplified class diagram.

GoOrg implements two main processes: (i) the organisation generation process, which searches for the space visiting states and performs structure transformations; and (ii) the binding process which binds agents and positions matching their features. The classes used in the generation process are presented on the lefthand side and the classes

for the binding process on the righthand side of the diagram. The centre of the diagram illustrates the classes that provide facilities to generate the outputs.

The generation process is handled by the class *OrganisationGenerator*, which picks a search algorithm (currently only Breadth-First). The search state is defined by the class *Organisation*. The initial search state has an empty *PositionsTree* and all goals of the *GoalsTree* to be assigned. As the search is performed, the state that is being visited is cloned and the *PositionsTree* (organisational structure) of the clone is subjected to a transformation. The transformations and the algorithm to generate successor states (Algorithm 2) are also defined in the class *Organisation*. The search reaches a target state when all the goals of the *GoalsTree* are assigned to positions in the *PositionsTree*. When a target state is visited, an organisational structure candidate is plotted in Graphviz format. Besides, information about the solution is printed in the statistics file.

After finishing the search for generating candidates, i.e., there is no state to be visited, the binding process is started. The binding process is handled by the class *OrganisationBinder*, which picks a binding algorithm (currently only First-Fit). It tackles the binding of the organisations in order of preference, from greatest to least. The result of the binding is printed in the statistic file and a JaCaMo project file is created considering the agents that were bound with the positions of each candidate.

6.2.1 Executing GoOrg Implementation

GoOrg is compiled, tested and executed using Gradle. By default, the Gradle run task compiles and launch the project, which includes downloading and linking the project dependencies. This task can receive one or more arguments. The first argument should be a $\mathcal{M}oise^{+}$ like XML organisation specification containing a scheme which defines the organisation's set of goals and optionally a set of available agents for the binding process. If no agents are specified, the generation process runs nornally, just the binding process is not executed. Since no user's preferences were set, by default each search state has a *UNITARY* weight (cost), resulting in an arbitrary search order. The following command line illustrates how to launch *GoOrg4DSN* specifying the XML as input.

```
$ ./gradlew run --args="examples/dsn.xml"
```

The next arguments refer to the criteria for ordering the generation process and, consequently, the final list of candidates. The criteria also impact the generation process order, because according to the chosen criteria the cost functions change, making preferable structures the first ones to be generated. For instance, if GoOrg is parametrised to generate more *GENERALIST* structures, the states to visit that present more *GENERALIST* structures are less expensive in terms of the search cost function, which makes the algorithm explore these states first. In the command line, it is possible to specify any

number of criteria, in which the last is the higher priority. The following command line illustrates how to launch *GoOrg4Prod* specifying an XML as input and the user's preferences. In this example, it is preferred the more *GENERALIST* structures and secondly the more *EFFICIENT*, followed by the *FLATTER* structures.

```
$ ./gradlew run --args="examples/Feed_production_line_evaluation.xml \
   FLATTER EFFICIENT GENERALIST"
```

6.2.2 GoOrg Implementation Inputs

GoOrg uses as inputs a set of goals and a set of agents. These sets are generically provided in a $\mathcal{M}oise^{+}$ like XML file. Other inputs refer to the user's preferences. To summarise, the model inputs are:

- An XML containing the goals and agents;
- In the XML file, the set of goals follows the $\mathcal{M}oise^{+}$ scheme with extra elements in the XML referring to the features according to the domain;
- In the XML file, the set of agents is given using the elements "available-agents", "agent" and others according to the domain to refer to the agent's features;
- The sorting criteria in which the last mentioned criterion is the higher priority (if not specified UNITARY criteria is used, meaning no differentiation among states, i.e., arbitrary generation order).

6.2.3 GoOrg Implementation Outputs

The main output that GoOrg produces is a set of organisational structures which are represented by graphs. A graph format is used since the positions of a tree may have different relationships, which can have different meanings according to the domain. Along with other examples that are illustrated in this work, Figure 29 shows some of the graphs generated for the *Feed Production scenario with Three Goals*.

In GoOrg, graphs are used to represent sets of goals and positions, since both can be seen as nodes with relationships. Indeed, in *GoOrg4Prod* and *GoOrg4DSN* domains, it is mainly used relationships to refer to superordinate-subordinate relationships, commonly referring to authority relationships. However, it is also possible to represent other kinds of relationships as *GoOrg4Prod* represents a bandwidth usage (*dataload*) in which agents that are in charge of some particular goals must be aware. To summarise, the model outputs are:

- The set of goals in Graphviz format;

- Organisational structure candidates in Graphviz format;
- Statistics about candidates including their attributes in CSV format;
- A JaCaMo (JCM) file for launching the bound agents.

Table 3 shows a sample statistic output file which is used to choose organisations. Some of the organisational structures mentioned in this data are illustrated in Figure 29. The name of the columns stand for: $|P|$ is the number of synthesised positions, *Idle* for a % of idleness, *Effi* for a % of efficiency (complement of idlenes), *Genr* for a % of generality, *Spec* for a % of specificness (complement of generality), *Tall* for a % of talness, *Flat* for a % of flatness (complement of talness), *Feas* for a % of feasibility, and *Levels* for the number of hierarchy levels. As specified in Listing 6.2, there are three available agents with the following skills: agent `bt` can *move*, agent `pp` can do *pnp* and agent `ie` can *lift*. The column match shows bound positions and agents. The number of pairs of positions and agents should be the same number of positions to have 100% of feasibility. As seen, structures #1 to #10 are not 100% feasible.

The statistics generated refer to the attributes each candidate presents and the matching result of positions and agents. This data helps to easily find the best feasible candidate. In case of agents' availability changes, it can be revised and a new candidate can be quickly picked. Regarding the JaCaMo project file (JCM), GoOrg generates a file with the bound agents for each candidate.[4]

6.3 Extending GoOrg

GoOrg is designed to be extended for different domains.[5] Regarding GoOrg project implementation, each domain requires a revision on the *annotation* package. The abstract class *Annotation* gives the skeleton of a feature. In the GoOrg project the class *Feature* extends *Annotation*, as an example of extension. It represents a very generic feature which only has an identification. However, a feature is often a piece of more complex information, for instance, something associated with an expected effort (such as a *workload*). As new features are added, the class *OrganisationXMLParser* should be adapted for parsing the specific XML elements of the domain.

Each *Annotation* can be associated with both a position to be occupied and with an agent. In GoOrg, the *PositionNode* and the *Agent* classes implement more generic classes called *Requirement* and *Resource*, respectively. These generic classes support other kinds of constraints that can be implemented in the future. The *OrganisationBinder* class defines bindings between requirements and resources, matching their *Annotation* objects. In this

[4] Appendix D illustrates a JCM file generated by GoOrg implementation.
[5] Considering the use of a search space algorithm, GoOrg can be easily extended.

Table 3 – Organisational structures generated for Feed Production with three goals.

id	|P|	Idle	Effi	Genr	Spec	Tall	Flat	Feas	Levels	Match
1	1	0	100	100	0	0	100	0	1	
2	2	50	50	25	75	0	100	50	1	p1=ie
3	2	50	50	25	75	0	100	50	1	p1=pp
4	2	50	50	25	75	0	100	50	1	p1=bt
5	2	50	50	25	75	50	50	50	2	p1=pp
6	2	50	50	25	75	50	50	50	2	p1=bt
7	2	50	50	25	75	50	50	50	2	p1=ie
8	2	50	50	25	75	50	50	50	2	p0=ie
9	2	50	50	25	75	50	50	50	2	p0=pp
10	2	50	50	25	75	50	50	50	2	p0=bt
11	3	67	33	0	100	0	100	100	1	p0=pp p2=ie p1=bt
12	3	67	33	0	100	50	50	100	2	p2=pp p0=ie p1=bt
13	3	67	33	0	100	50	50	100	2	p0=pp p2=ie p1=bt
14	3	67	33	0	100	50	50	100	2	p1=pp p0=ie p2=bt
15	3	67	33	0	100	50	50	100	2	p1=pp p0=ie p2=bt
16	3	67	33	0	100	50	50	100	2	p0=pp p2=ie p1=bt
17	3	67	33	0	100	50	50	100	2	p2=pp p0=ie p1=bt
18	3	67	33	0	100	50	50	100	2	p1=pp p2=ie p0=bt
19	3	67	33	0	100	50	50	100	2	p1=pp p0=ie p2=bt
20	3	67	33	0	100	50	50	100	2	p0=pp p1=ie p2=bt
21	3	67	33	0	100	100	0	100	3	p0=pp p1=ie p2=bt
22	3	67	33	0	100	100	0	100	3	p1=pp p2=ie p0=bt
23	3	67	33	0	100	100	0	100	3	p2=pp p0=ie p1=bt
24	3	67	33	0	100	100	0	100	3	p1=pp p0=ie p2=bt
25	3	67	33	0	100	100	0	100	3	p0=pp p2=ie p1=bt
26	3	67	33	0	100	100	0	100	3	p2=pp p1=ie p0=bt

sense, the classes *PositionNode* and *Agent* should override the method *getAnnotationIds/0* in order to return the annotations that matters in the domain. The bindings are registered in the class *Binding*, which informs the feasibility attribute of the organisational structures according to the given agent set.

For other domains, the *Organisation* class must also be revised in terms of structural transformations and organisational attributes. The transformations are applied to every state successors generation (Algorithm 2). The output for a domain may also have particular constraints. For instance, in *GoOrg4Prod* and *GoOrg4DSN* hierarchies were applied for generating a forest of hierarchies. Other domains may require other kinds of structures such as holarchies and teams (Horling and Lesser, 2004). Different kinds of structures may call for other kinds of structural transformations. In this sense, according to the constraints of the outputs, different structure transformations should be defined.

For defining proper attributes for another domain, the *PositionsTree* class must

be revised. This class gather information about the structure. Most of the attributes can be inferred from the organisational structure shape and relationhips.[6] By default, GoOrg define attributes and their counterparts in complementary ranges between 0 and 1. Listing 6.3 illustrates an attribute implementation. Attributes are used by the search cost function whilst the search is in progress.

Listing 6.3 – Implementation of the attributes efficiency and idleness

```
public double getEfficiency() {
  double capacity = this.tree.size() * Parameters.getMaxWorkload();
  double occupancy = this.getSumWorkload();

  return occupancy / capacity;
}

public double getIdleness() {
  return 1 - getEfficiencyFactor();
}
```

According to the domain, the given set of goals can also be the subject of processes that should be executed before the generation process. For instance, in *GoOrg4Prod* the goals are subject to a split process that divide large efforts and large data loads into smaller amounts. It serves as an example of an adaptation of the inputs. Other adjustments that are not mentioned here may also be necessary.

[6] Other studies such as So and Durfee (1998) and Grossi et al. (2007) also propose methods for evaluating an organisation.

7 Discussion

This section presents a discussion about aspects related to the organisational design, comparing GoOrg to other approaches. Five aspects are considered: (i) procedures for organisational adaptation; (ii) assigning goals to named agents, roles or positions; (iii) planning resources (agents) of organisations; (iv) the difference between synthesising positions and having user-defined roles; and (v) the difference between using goals as input rather than roles and behaviours. At the end of this section, a summary of this discussion is presented.

7.1 Organisational Adaptation

Due to the dynamism of the system, along with the organisation's life cycle, different situations may require different organisational adaptations (Hübner et al., 2006; Martínez-Berumen et al., 2014; von Bertalanffy, 1968). Some circumstances might only affect the resources' availability, in which a replacement of the resource solves the situation. Other situations cannot be solved using the current organisational structure. For instance, a change in the set of available agents or in the user's preferences may require another structure, possibly an already generated one. Still, there are situations that cannot be solved using any already generated candidate, such as a change in the goals, which requires a complete redesign for generating new candidates. In this sense, three adaptation procedures can be applied according to the situation: (i) reallocation, (ii) structure-switching, and (iii) redesign.[1]

The separation between the processes of generating organisations and binding agents and positions fosters these different adaptations. It splits the complexity of a design model into modules, which facilitates the implementation and maintenance of proper methods for both tasks. Besides, a design may be performed faster when its requirements can be satisfied by running just one of the processes. This separation is a characteristic of *GoOrg4Prod* and *GoOrg4DSN*, two GoOrg specialisations introduced in this work.

To present how the mentioned procedures can be executed, it is considered the motivating scenario illustrated in Figure 28. It requires an organisation of agents that has three goals to feed a production line in a factory (see Appendix D for more details). The agents must lift down boxes from shelves, and move them to place the required items on

[1] Comparing to the classification stated by DeLoach and Matson (2004), a reallocation is a *state reorganisation* and a redesign is a *structure reorganisation*.

a conveyor belt. For achieving these goals, the agents must have specific skills.

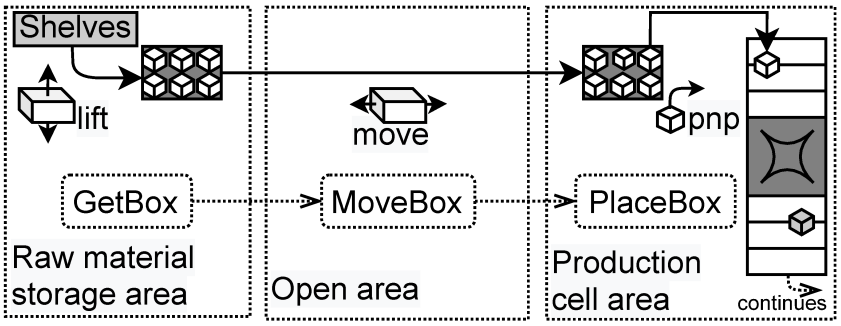


Figure 28 – Feed Production Scenario.

Figure 29 presents some of the 26 structures that *GoOrg4Prod* generates for the given scenario. The criteria used to sort these candidates is the most *generalist* (higher priority), then the most *efficient* and the *flatter* (lower priority) structure. Some of the generated structures are flat, and some of them have no superordinate-subordinate relationships, such as the illustrated structures #1 and #2. Other structures, such as #6, #15 and #26 may present hierarchies. Some structures may have very generalist positions, such as structure #1 in which the only position (`p0`) must achieve all the organisational goals. Oppositely, it is also possible to build structures in which the positions are very specialist such as the positions `p0`, `p1` and `p2` illustrated in the structures #15 and #26.

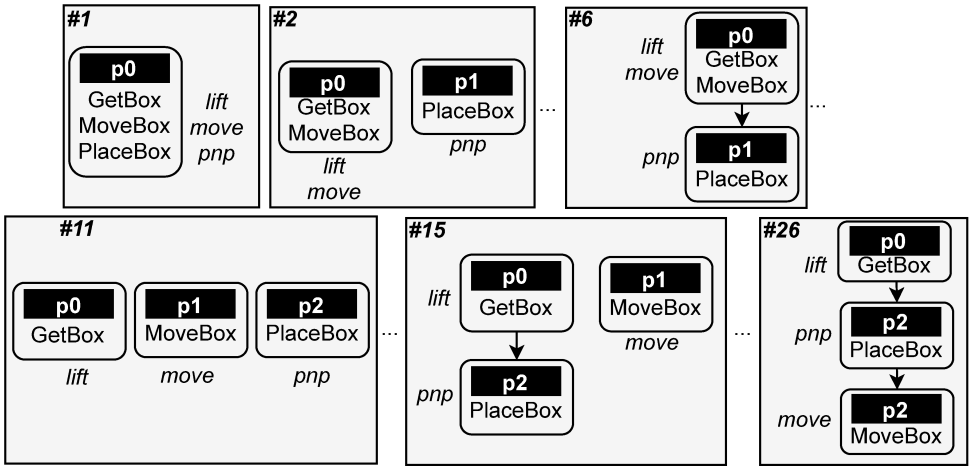


Figure 29 – Some of the candidates for the Feed Production Scenario.

In this example, each goal requires one skill $s \in S$ to be achieved. The set of all skills for this scenario is $S = \{move, lift, pnp\}$. For simplification, it is considered s as a symbol instead of a monuple as expected by *GoOrg4Prod*. A given structure is able to achieve the goals only if there are available agents having the mentioned skills to be allocated in the synthesised positions of each solution candidate. In this sense, the most generalist positions are also the ones that require agents with a higher number of skills.

Indeed, the structure #1 demands a factotum agent that must be able to *lift*, *move* and *pnp* altogether. Oppositely, the most specialist positions require agents with fewer skills. A specialist agent is focused on a few goals, possibly only one. The structures #11, #15 and #26 are examples of structures requiring just specialist positions.

It is considered that the previous structure may present different positions comparing to the new structure. Positions are considered similar when they are associated with the same goals, which implies that they are associated with the same features (in this example, the skills). To help in the analysis, it is introduced the concept of kinds of position and kinds of agent. Positions with the same skills are considered to be of the same kinds. Agents with the same skills are considered to be of the same kinds. For a given domain, both kinds of agents and kinds of positions are elements of a common set, denoted by R.[2]

Indeed, each position is of some kind $r \in R$ as it has associated a combination of skills which is one of the subsets of the finite set S. In other words, r is a subset of the set of skills. As expressed by Eq. 7.1, R is the set of all kinds of positions that may be synthesised for all generated structures. The set R can be defined as the power set of S, so its cardinality is given by $2^{|S|}$ (Rosen, 2012).

$$R = \mathcal{P}(S) \tag{7.1}$$

Following the same idea, R also represents kinds of agents (Eq. 7.1). For instance, r can be the empty set representing a position that requires no skill. When r refers to an agent, the empty set represents an agent with no skill. Positions and agents can also present skills as any other subset of S. For this particular motivating scenario, the set R is as follows.

$$R = \{\varnothing, \{lift\}, \{move\}, \{pnp\},$$
$$\{lift, move\}, \{lift, pnp\}, \{move, pnp\}, \{lift, move, pnp\}\}$$

To check if an agent can be bound to a position it is considered the functions $pr(p)$ and $ar(a)$. The function $pr(p)$ maps the set of skills that the position p requires. The function $ar(a)$ maps the set of skills that the agent a has.

$$pr : P \rightarrow R$$
$$ar : A \rightarrow R$$

To allocate an agent a to a position p, the agent must have all the skills required by the position. Thus, if $pr(p) = ar(a)$ the position and the agent have an exact match,

[2] A subset of skills can also be seen as a role, see Appendix F for details.

i.e., the agent has exactly the skills that the position requires. However, an agent a that has more skills that the required skills of the position p is also suitable to be allocated to p. The function $match(a, p)$ indicates if the agent a is able to perform the position p (Eq. 7.2). An organisation that has agents to fill all its positions is considered feasible. The feasibility of the organisation o is represented as $\kappa(o)$ (Eq. 3.1).

$$match : A \times P \rightarrow \{0, 1\}$$

$$match(a, p) = \begin{cases} 1, & \text{if } pr(p) \subseteq ar(a) \\ 0, & \text{otherwise} \end{cases} \tag{7.2}$$

In the next sections, these definitions are used to explain the organisational adaptation procedures and possible costs that are associated with each of them.

7.1.1 Reallocation

A reallocation is a procedure for changing a binding relationship between an agent and a position. It does not change the structure of the organisation. Many situations and many purposes may demand a reallocation. For example, for some reason, it may be required to interchange two agents between their positions. It also may be needed an agent replacement, for instance, due to an agent failure, as illustrated in Figure 30 in which the agent D is replacing the failed agent B.

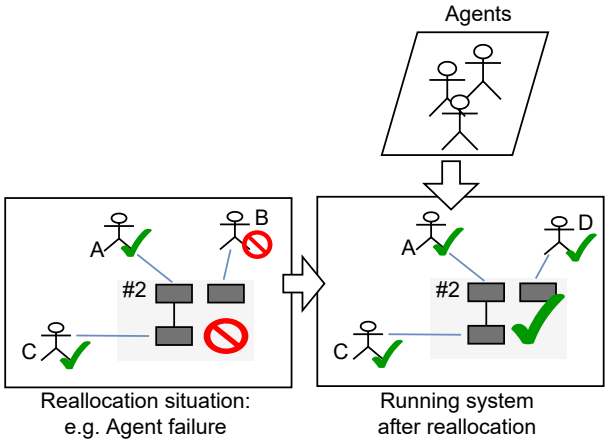


Figure 30 – A reallocation by replacing an agent by another.

Although some kinds of agents match some kinds of positions, some of them have skills that surpass the required skills. An agent that has more skills than required is said overqualified (Summerfield, 2016). The overqualification is a drawback since agents with more skills are usually more expensive and often scarce, thus it represents a cost. The *overqualification cost* is denoted by the function oc.

$$oc : A \times P \rightarrow \mathbb{R}^{+}$$

The function $oc(a, p)$ indicates an *overqualification cost* of allocating the agent a into the position p (Eq. 7.3). Notice that it is a simplistic estimation based on unitary costs. The overqualification cost depends on the context and different skills may represent different costs. To calculate, it is being considered the subset r of a position p (given by $pr(p)$), and of an agent a (given by $ar(a)$). If $pr(p) = ar(a)$, there is no *overqualification cost*. For the other cases, it is considered that each skill that a has and p does not require represents a cost factor of 1. If a does not match with p the *overqualification cost* is considered infinite.

$$oc(a, p) = \frac{|ar(a)| - |pr(p)|}{match(a, p)} \tag{7.3}$$

The *overqualification cost* of allocating each possible kind of agent in each kind of position can be calculated. For this analysis, it is considered that there is an agent of each kind $r \in R$, and there is a position of each kind $r \in R$. Thus, it is applied the function oc (Eq. 7.3) on all kinds of positions and agents given by Eq. 7.1. For this particular example, the set R has 8 elements, thus there are 8 kinds of agents and 8 kinds of agents. Figure 31 presents the *overqualification cost* of matching every kind of agent to every kind of position. The darker squares represent higher *overqualification costs*, i.e., the kind of agent has more skills than needed. The lighter squares represent zero *overqualification cost*, i.e., the kind of agent has exactly the required skills. The white squares represent that the kind of position does not match with the kind of agent (infinite cost).

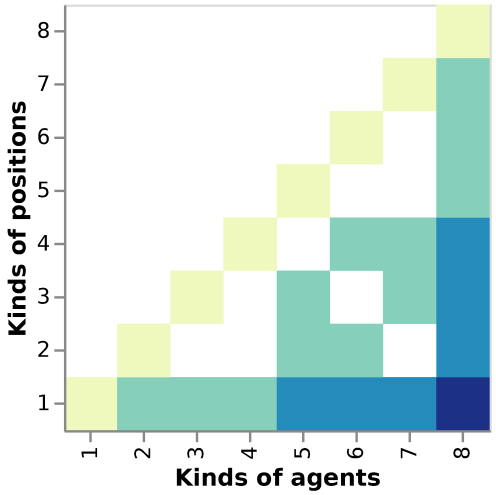


Figure 31 – Matching kinds of agents and kinds of positions

The kinds of agents #2, #3 and #4 of Figure 31 have just one skill each, thus they can only match with the kinds of positions that require no skills and with the positions

that require exactly the skill they have. The kinds of agents #5, #6 and #7 have two skills each, thus they match with positions that required any combination of their skills. The kind of agent #8 is the factotum agent since it has all three skills in this scenario. The factotum agent can be allocated in any position, including the position that requires the three skills, in which it is the only kind of agent that matches.

Assuming that there are multiple complete solutions for a given domain, agents that match multiple positions provide flexibility in terms of the place in the organisational structure that an agent occupies. A structure made up of positions that match with multiple agents also provides flexibility in terms of the agent that is chosen to occupy a particular position. Besides flexibility, such systems can be more robust, considering that there are agents that can replace others. However, it comes with an *overqualification cost*, which is a drawback that must be contrasted with the benefits of flexibility and robustness.

7.1.2 Structure-switching

A structure-switching is the procedure for changing one structure by another candidate. In this case, it is expected that a structure from a set of previously designed candidates is picked. From this set, structures are chosen according to their attributes. In the case in which the number and kinds of positions change, the binding relationships between agents and positions are also affected. Even when the new structure has the same number and kinds of positions as the previous structure, it depends on the switching strategy whether the binding relationships will be maintained or not.

Since selecting a new structure merely involves creating new bindings between agents and positions, the process can be completed quickly. Although it is not exploited in this work, because of its quick responses, such a procedure may suit the *online* planning/design concept, i.e., when the planning/design is executed during runtime (Cardoso and Bordini, 2019).

The structure-switching procedure may be triggered, for instance, due to a change in the agents' availability. Consider that the structure previously chosen calls for an agent with two skills, but that agent is no longer available and there is no other agent with those necessary skills. Considering the availability of more specialised agents, switching to a more specialised structure can solve this issue, keeping the system running.

This procedure also may occur due to a user's preference change. It is illustrated in Figure 32. In this example, the organisations are re-sorted using the new preferences. Another feasible structure may become the best choice to be used by the running system. A revision on the bindings between running agents and positions is usually required if a new organisational structure is adopted.

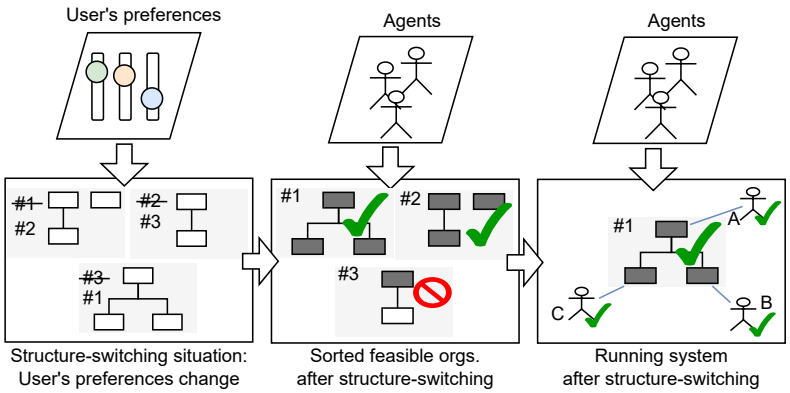


Figure 32 – A required structure-switching due to a change in the user's preferences.

A structure-switching may also be applied to make a change only on the hierarchy places (in the positions relationships) and also to make a small change in the assignment of goals into positions. For instance, the user may change the preference from a *taller* structure to a *flatter* structure. In this case, there are structures with similar positions as before, but in different places of the hierarchy.

In terms of the user's preferences, switching a structure may have little impact on the organisation. Indeed, according to the current structure in use, it is possible to have many other structures that have similar attributes. Figure 33 shows a correlation matrix of the *generality* of the 26 generated structures for the motivating scenario of this analysis.[3] In this example, there is no other candidate with the same *generality* of structure #1. In this sense, wether the user prefers more general structures and the structure #1 is in use, a switching to another means the adoption of a structure with less *generality*. However, if another structure is in use, for instance, the structure #2, there are other options that have similar *generality*, like the structure #3 (in this case, any structure between #2 and #10). In terms of *generality*, all the structures between the structure #11 and the structure #26 are also similar. In this sense, a change between structures that are 100% similar, considering a particular attribute, has no difference regarding the user's preference.

However, according to the current structure, a change may have a cost, for instance, from the structure #1 to structure #11 which are considered with 0% of relative similarity in terms of *generality*.[4] If the newly adopted structure is less preferred by the user, it has an *unsuitability factor*. Such factor depends on the attribute, in this example, it is being assessed the *generality* which in *GoOrg4Prod* model is given by Eq. 4.4. Considering o_i the previous structure and o_j the new structure, the *unsuitability factor* can be simply given by the relation of a given attribute of the previous and of the new structure. Thus,

[3] Appendix A presents correlation matrices for other organisational structure attributes.
[4] The relative similarity considers equal attributes as 100% similar and the farthest values as 0% similar.

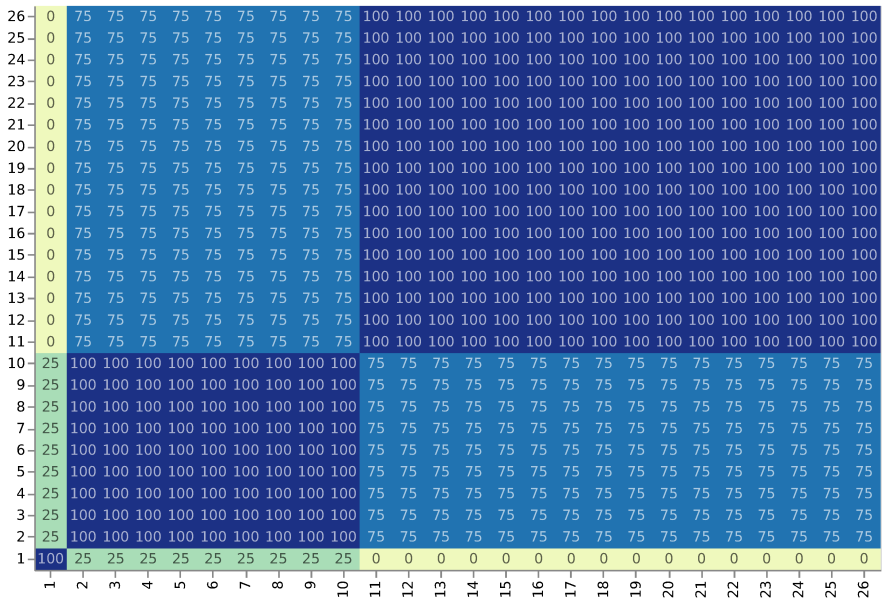


Figure 33 – Comparing the generality of candidates of the motivating scenario.

the *unsuitability factor* for the attribute *generality* is given by $\theta(o_i)/\theta(o_j)$. This factor can be lower than 1, meaning that the new structure is actually preferred over the previous structure, and also equals 1 if for the user's preferences there is no difference in a structure-switching. A cost may be associated with this factor according to the attribute and the domain. For instance, a switch from a more *generalist* to a more *specialist* structure may decrease the overall MAS robustness, which should imply a cost (or risk).

Nevertheless, even among similar structures, a structure-switching may result in differences in the kinds of positions between them. A new position may reflect the need for acquiring a new agent, an *acquisition cost*, denoted by the function ac. For this, P_j is the set of positions in the new structure and P_i is the set of the positions of the previous structure.

$$ac : R \times P_i \times P_j \rightarrow \mathbb{R}^+$$

It is considered that there is an *acquisition cost* for each position that does not exist in the previous structure, considering a unitary cost for each (Eq. 7.4).[5] This equation checks the sum of each kind of position in the new structure compared to the previous structure. In this sense, the *acquisition cost* is considered in case of having new kinds of positions and also when the cardinality of an existing position of the new structure is superior compared to the same kind of position in the previous structure. This equation is ignoring the possibility of reuse in the case of a previous position that requires more

[5] Using bracket notation to define 1 when the statement between brackets is true, and 0 when it is false.

skills being replaced by a new position that requires fewer skills. For instance, it does not take into account that an agent occupying a position that requires $\{lift, pnp\}$ could also occupy a new position requiring just $\{lift\}$. Besides, this equation assumes that the agents in use in the previous structure have the exact required skills, i.e., they are not overqualified, so they cannot be allocated to new positions that require more skills than the previous position. In this case, it would be necessary to have information about the agents bound with each previous position. For example, if in the previous structure all the agents allocated are factotum agents, the equation will not consider any *acquisition cost* in a switch.

$$ac(R, P_i, P_j) = \sum_{r \in R} max(0, \sum_{p_j \in P_j} [pr(p_j) = r] - \sum_{p_i \in P_i} [pr(p_i) = r]) \tag{7.4}$$

Regarding the *acquisition cost* of a structure-switching, it is being considered the *acquisition cost* of 1 for acquiring each kind of position. It is considered zero cost when the previous and the new structure have the same kinds of positions and cardinalities of each kind of position.

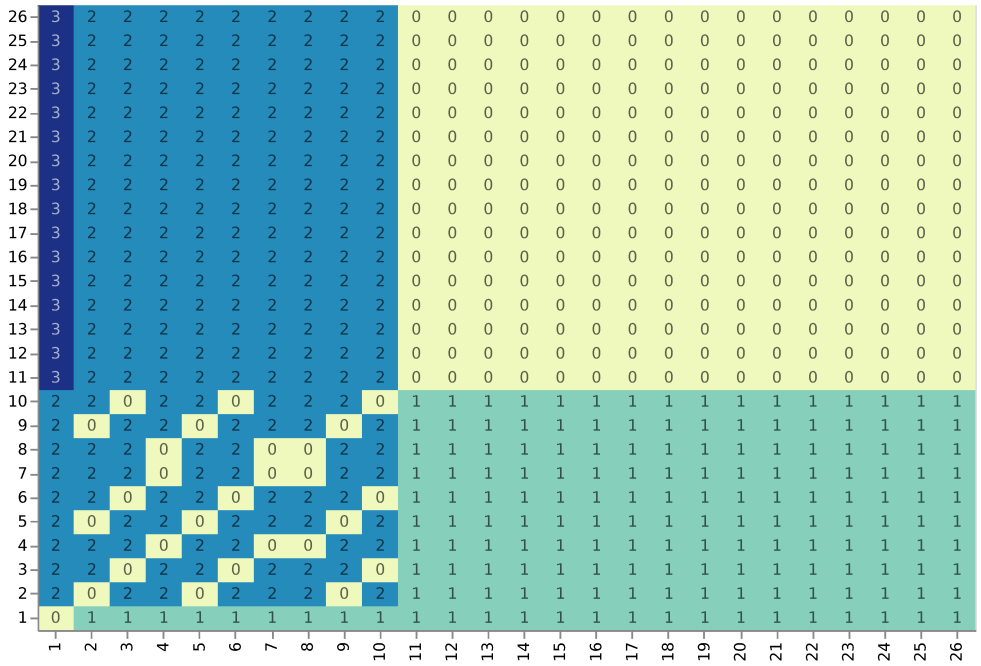


Figure 34 – Acquisition cost when switching between structures of the motivating scenario.

Figure 34 illustrates the *acquisition cost* for the motivating scenario considering a change from one of the 26 structures to another. For instance, a switch from the structure #1 which has only one position (Figure 29) to the structure #2 requires the acquisition of two positions that are not present in #1. The switch in the opposite direction (from #2 to #1) costs less since the structure #1 requires the acquisition of only one position that is not present in #2.

Instead of switching only similar positions, it is also possible to reuse agents of the previous positions which exceed the required skills of the new positions. The *acquisition cost* could not be considered when a position in the previous structure (in the set P_i), surpasses the requirements of a position in the new structure (in the set P_j). However, the reuse of such an agent represents an *overqualification cost*.

7.1.3 Redesign

Finally, Figure 35 illustrates a complete redesign procedure. A complete process of synthesising positions and generating structures should be triggered when either the set of goals or design parameters change. This situation is illustrated in Section 5.4 while presenting GoOrg4DSN results in different situations in which the number of targets changes. In those situations, when a target enters or leaves an area (or even passes over a sector to another), the set of goals changes, requiring a redesign.

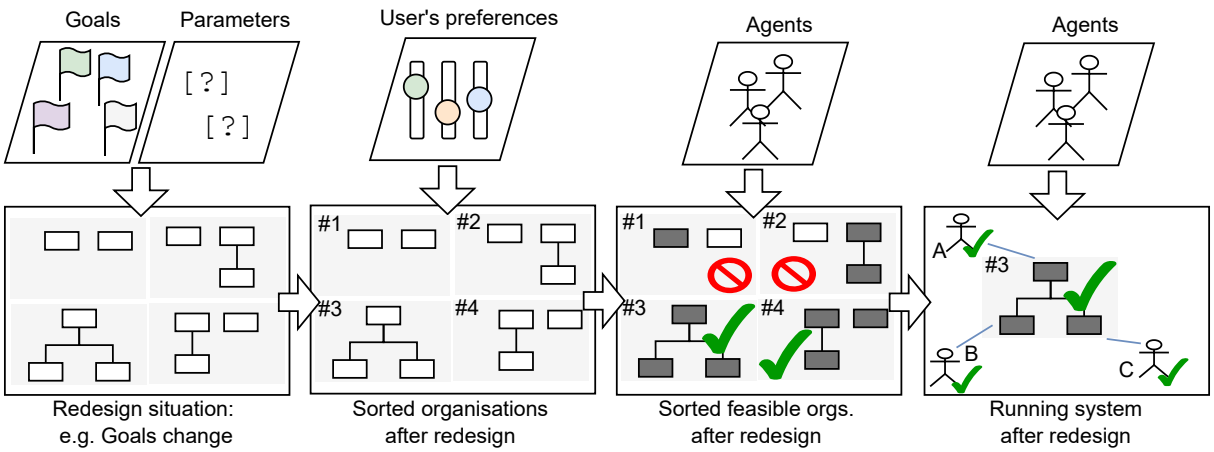


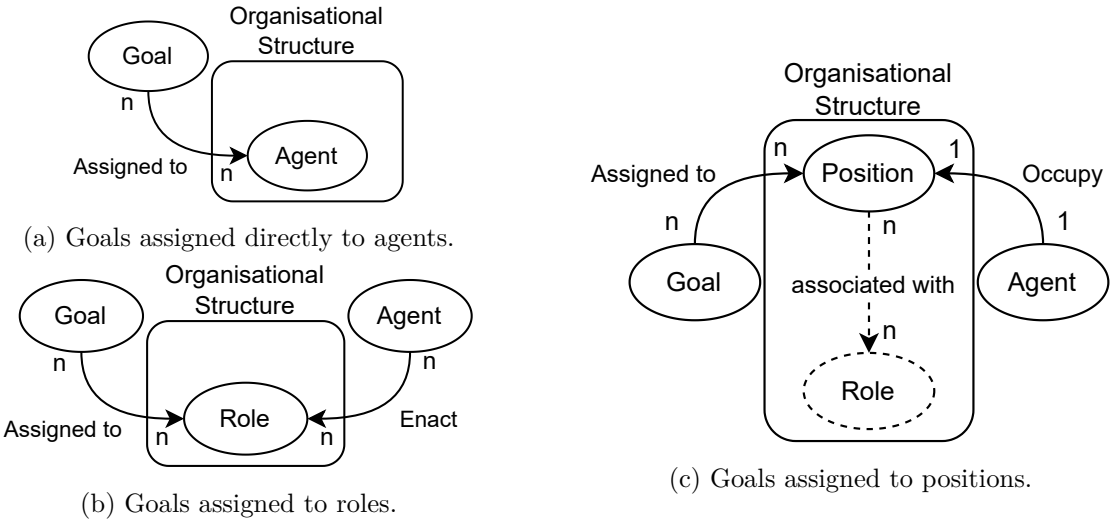
Figure 35 – A complete redesign after a change to the set of goals.

A redesign brings a *design cost*, which occurs on generating organisational structure candidates that depend on the complexity of the problem and the given G. When switching the previous structure to a newly designed structure it is also possible to have *overqualification costs* and *acquisition costs*.

7.2 Assigning Goals to Named Agents, Roles or Positions

Task allocating models (Cardoso and Bordini, 2019; Decker, 1995; Sleight, 2014) can generate organisations. These models assign goals (or tasks) to named agents, as illustrated in Figure 36a. For instance, let us consider a *marketplace* organisation that has the (root) goal *do business*. This root goal can be decomposed into the subgoals *sell product* and *buy product*, which can be decomposed into other subgoals, and so on. One agent assigned to *sell product* interacting with another agent assigned to *buy product* are sharing

the same root goal, and should cooperate to achieve it. In this sense, the distribution of subgoals to a group of agents generates organisations. The assignment of goals to named agents can be simpler to implement compared to impersonal representations of agents, such as roles and positions. However, the assignment of goals to named agents produces fixed organisations. These organisations are *closed* as they are formed only by the named agents. In other words, a change in the set of goals or in agents' availability implies a *redesign*.



(a) Goals assigned directly to agents.

(b) Goals assigned to roles.

(c) Goals assigned to positions.

Figure 36 – Different forms of assigning goals to organisational members.

Other studies (DeLoach and Matson, 2004; Horling and Lesser, 2008; Sims et al., 2008) use the concept of roles. Roles are largely used in human organisations and have been adopted by the MAS community. For instance, a person that enacts the role *assembler* in a factory is responsible to assemble parts of products in a production line. Sometimes, there are many-to-many relationships such as when that person concomitantly enacts another role like the *supervisor* role, dividing their working hours between assembling and supervising activities. Figure 36b represents an organisational structure formed by roles. In this case, the goals are assigned to roles and the agents enact roles, becoming in charge of their assigned goals. Using the concept of roles, the organisation is decoupled from the agents. As it allows many-to-many relationships, this approach can be very flexible at runtime, allowing agents to enact roles for the short or long term. Using roles, a generator is not limited to the availability of agents, i.e., it can generate organisations that best match design criteria, such as the best distribution of goals. In this sense, this approach allows a generation of more appropriate solutions according to design criteria, since it has no concerns about the specificities of agents. Besides, a proper parametrisation of a generator can avoid the design of organisations that are not fillable by the available agents.

Another characteristic is that the design of the organisation (as an entity) can be

separated from the process of binding agents and roles. It simplifies the generating process since the binding part is delegated to another independent process. When these processes are separated, the redesign can be a lighter procedure, as discussed in Section 7.1. This approach also avoids redesigning processes, which are usually computationally heavy. To exemplify how it avoids redesigns, let us compare the approaches represented in Figure 36a and Figure 36b. In the former, goals are assigned to agents that are fixed to the organisational structure. In that case, if an agent is not able to perform its job, a redesign is needed. In the latter, the approach uses roles, thus agents are associated with roles. It makes the organisation and the agents decoupled entities. In this case, if an agent is unable to achieve the goals assigned to its role, no redesign is required; instead, another agent must be found to fill that role. Thus, an organisation made up of roles can be an open system, i.e., agents can join, enact roles and leave the organisation at any time.

GoOrg introduces the use of organisational positions. In this approach, goals are assigned to positions. A position is a place in the organisational structure that has a one-to-one relationship with an agent. Agents can occupy and leave positions, but an organisational position can be occupied by only one agent at a time. Like roles, positions decouple the organisation and agents, supporting the development of open MAS. Besides having most of the advantages that roles bring, positions foster planning resources, as discussed in Section 7.3. Indeed, as illustrated in Figure 36c, a position directly reflects an agent, i.e., it may have all relevant characteristics an agent has for a design process, but without naming it.

7.3 Planning Resources of Organisations

In contrast to other models, GoOrg considers that an organisation has a set of positions rather than a set of roles. Roles and positions within an organisation are roughly analogous. Both of them are impersonel representations of agents in an organisation. As discussed in Section 7.2, the use of roles or positions instead of named agents provides more freedom to the organisation's design because roles and positions do not need to carry restrictions as agents do. In other words, the design process may generate more possibilities, including an ideal distribution of responsibilities. The adoption of a better solution can be made at various stages of the organisational life cycle. In this sense, having more design solutions, even if they are unfeasible at some points, increases flexibility.

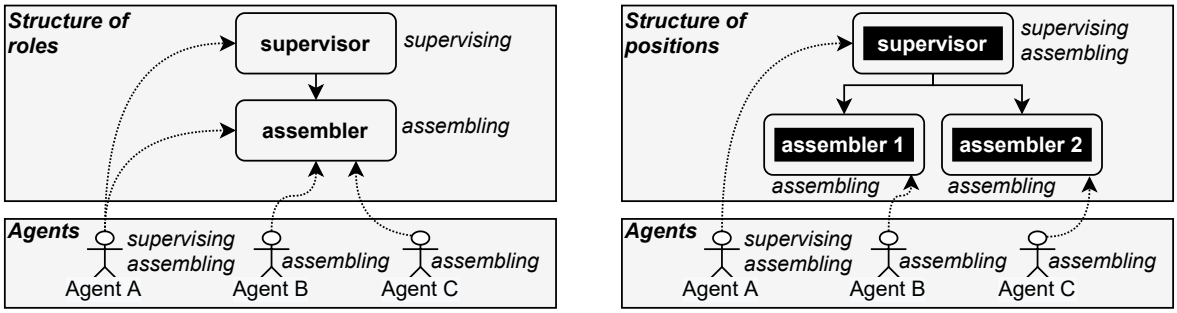
Roles and positions decouple organisations from agents, making organisations' specifications independent of agents' constraints. They are also useful to describe the responsibilities and rights of each member of an organisation. Besides, they support the development of open systems since they foster members' entrances and exits (Deloach et al., 2008).

Unlike a position, a role can be performed by multiple agents who are each responsible for achieving the role assigned goals. Additionally, an agent can perform multiple roles (many-to-many relationships). Models that incorporate the concept of roles offer a high level of flexibility at runtime. For instance, an agent may enact the role *assembler* and the role *supervisor* at the same time. It may also leave one of these roles and keep the another, and many more combinations over time. When an agent enacts or leaves a role, it is changing its assigned goals. This flexibility can also avoid the need for changing the structure. For instance, structure-switching is necessary when a different distribution of goals along positions is needed. In a structure of roles, if a combination of multiple roles satisfies the changing needs, no structure-switching is necessary, it is just needed to change agent enactments. However, such flexibility makes the estimation of resources a tough task, especially when role enactments are very dynamic.

A structure made up of positions (one-to-one relationships), on the other hand, directly reflects resource needs. The use of positions is also an intuitive approach for defining sets of responsibilities and relationships. In the example of an assembler that also supervises the production, a design model can synthesise a position responsible for both activities. In this sense, this position is reflecting the actual runtime dynamics but with the advantage of defining it in the design time, which facilitates resource needs estimation. Positions can be still very flexible at runtime, for instance, when in the absence of the agent that usually occupies a position, an available agent may come to occupy it.

Figure 37 illustrates two structures for the same scenario. In this scenario, there is a production cell for assembling some products which has three agents in the team. Figure 37a illustrates a structure using the concept of roles. *Agent A* enacts two roles and *Agent B* and *Agent C* both enact the same role. Figure 37b illustrates a possible solution given by a model that extends GoOrg. It is a structure of three positions to be occupied by the *Agent A*, *Agent B* and *Agent C*. Since only *Agent A* is able to do both supervising and assembling activities, it is bound to the supervisor position. *Agent B* and *Agent C* are occupying the other positions. At design time, the need for three agents is only clear in a structure of positions. Although both structures represent solutions for the same scenario, only the structure of positions accounts for resource needs, enabling its usage as a resource planning tool.

An organisational structure made up of roles as shown in Figure 37a adheres to the standard for organisational charts (organigrams), which does not detail the cardinalities of roles (Mintzberg and Van der Heyden, 1999). Nevertheless, a model may synthesise a quantity of roles of a certain kind, as instances. The studies presented by Horling and Lesser (2008) and Sierra et al. (2004) synthesise instances of roles based on agents' behaviours. Figure 38 illustrates a structure made up of instances of roles. Still, as demonstrated in this scenario, the quantity of instances of roles may not infer the quantity of



(a) A structure of roles.

(b) A structure of positions.

Figure 37 – Comparing a structure of roles and a structure of positions.

required agents, as it does not reflect resource demands. In fact, for a model based on roles, resource planning is not facilitated even when using instances.

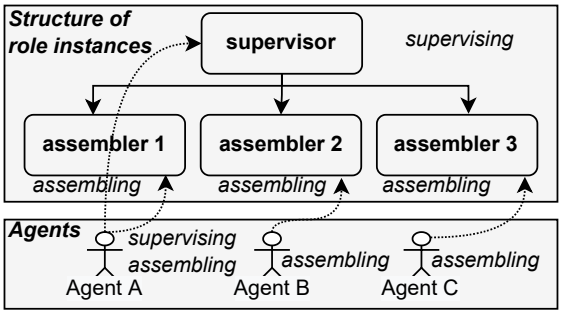


Figure 38 – A structure of instances of roles.

7.4 Synthesising Positions Instead of Requiring User-defined Roles

In other organisation generators (DeLoach and Matson, 2004; Horling and Lesser, 2008; Sierra et al., 2004; Sims et al., 2008), the user (engineer) has to specify roles a priori. For these studies, "it is the task of the engineer to determine which roles will be present at the level of the society design by means of an electronic institution" (Sierra et al., 2004, p. 3). Usually, for defining roles, users conceive an arrangement for the system and assess some characteristics of goals and behaviours and capabilities of known agents. To exemplify, for the DSN domain, one may specify the roles *sector_manager*, *sensor* and *track_manager*, since it is intended to have groups of agents organised into physical sectors. Basing role definitions on known elements facilitates the specification task and may generate coherent structures according to existing elements and to the user's conception of the system. Yet, it may come with biases, for instance, a wrong assumption on specifying a role may make it infeasible for available agents to play such a role.

The user-defined roles approach presumably produces a smaller number of solutions compared to a model that synthesises positions. A reduced number of candidates

should be faster to generate. However, a generator that produces fewer candidates using defined roles that might be specified with biases may not generate feasible solutions. Indeed, the user-defined roles approach may struggle when the system conditions do not match with the user's design assumptions.

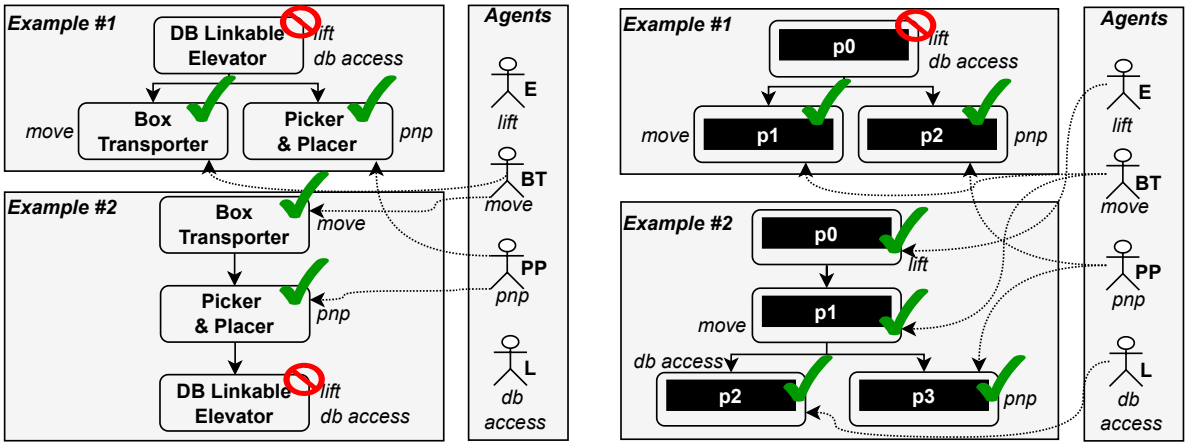
Alternatively, the user may set roles with few requirements. This strategy may increase occurrences of agents enacting multiple roles, making it easier to generate feasible organisations. Indeed, using the concept of roles and allowing many-to-many relationships between roles and agents, a model that defines a role for each given goal, for instance, can achieve a higher number of combinations at runtime. However, this option makes the coordination and resource planning of the system more complex because of the increased possibilities for an orchestrating system to assess. While complicating the runtime orchestrating task, such an approach makes the design less relevant since it delegates what is arguably a design task to the runtime orchestrating mechanism. The approach of using roles with little job to do also removes the possibility to plan at design time some important issues. For instance, if the design model distribute goals to different roles, it is delegating to the orchestrating mechanism to handle situations like avoiding that the performer of the goal $g0$ be the performer of the goal $g1$ (when such goals should not be performed by the same agent, like assembling and checking quality tasks). When synthesising positions instead, the model that assigns goals to the position is already defining what is expected from the performer of each position.

One may argue that at runtime, the number of combinations can be enlarged using many-to-many relationships between agents and roles. However, it must be taken into account that roles defined a priori still limit the combinations, since an agent should be able to perform all defined responsibilities of a role. A problem may occur, for instance, if the user sets a role with too many requirements, in a situation in which there is no agent that matches all requisites.

For the example illustrated by Figure 13, one can define the following roles: *DB Linkable Elevator*, *Box Transporter* and *Picker & Placer*. Models based on user-defined roles generate different structures for these given roles. Figure 39 presents examples that can be generated by each of these approaches.[6] In Figure 39a there are two of the possible structures of user-defined roles. In this example, the given set of available agents does not match the assumption used to define the roles. The role *DB Linkable Elevator* requires two skills that no agent has, i.e., in both examples, there is no way to fill the positions by the given agents. Other possible solutions beyond these examples are also made up of the same set of roles. Thus, there is no feasible solution to the exemplified condition. As a result, a design model based on those user-defined roles cannot find a feasible solution without a

[6] The assigned goals of the structure of positions are omitted to simplify this example, showing only skills as a matching feature for both role-based and position-based approaches.

user intervention redefining the roles and running the process again. Figure 39b, illustrates two solutions that *GoOrg4Prod* generates from synthesised positions. The example #1 is a solution presenting the same limitation of Figure 39a examples, exemplifying that *GoOrg4Prod* also generates non-feasible solutions. However, since *GoOrg4Prod* synthesises positions, among the solutions, there are structures in which the features of the set of positions match with the given agents as illustrated by the example #2 of Figure 39b.



(a) Structures of user-defined roles.

(b) Structures of synthesised positions.

Figure 39 – Comparing examples of user-defined roles and synthesised positions structures.

The importance of synthesising positions is also highlighted when comparing structures by their attributes. Figure 40 shows a correlation matrix considering *efficiency*, *height* and *generality* of the 26 generated structures of the scenario Feed Production with three goals. This figure presents averages of differences between every attribute of each structure comparing them to other structures. It shows that there are structures that are equal to others even considering all the *GoOrg4Prod* attributes. For instance, Candidates #2, #3 and #4 are equal in terms of their attributes.

It is worth going into detail comparing some of the structures that are pointed as similar by Figure 40. It exemplifies small differences between synthesised positions. Although small, it is essential to synthesise these differences and generate a range of candidates combining similar positions into different shapes and also different positions into some particular shapes. Considering all these possibilities, the chance of finding a suitable and feasible solution for a domain is increased. Taking candidates #2, #3 and #4 as an example, all these candidates have two positions and one hierarchy level, differing only in terms of the assigned goals as illustrated in Table 4.

Indeed, considering *GoOrg4Prod* transformations, these are the possible assignments of the three given goals into structures with two positions and one level. It encompasses a range of possibilities of kinds of agents for this particular structure configuration (with two positions and one hierarchical level). For instance, if the most skilful agent has

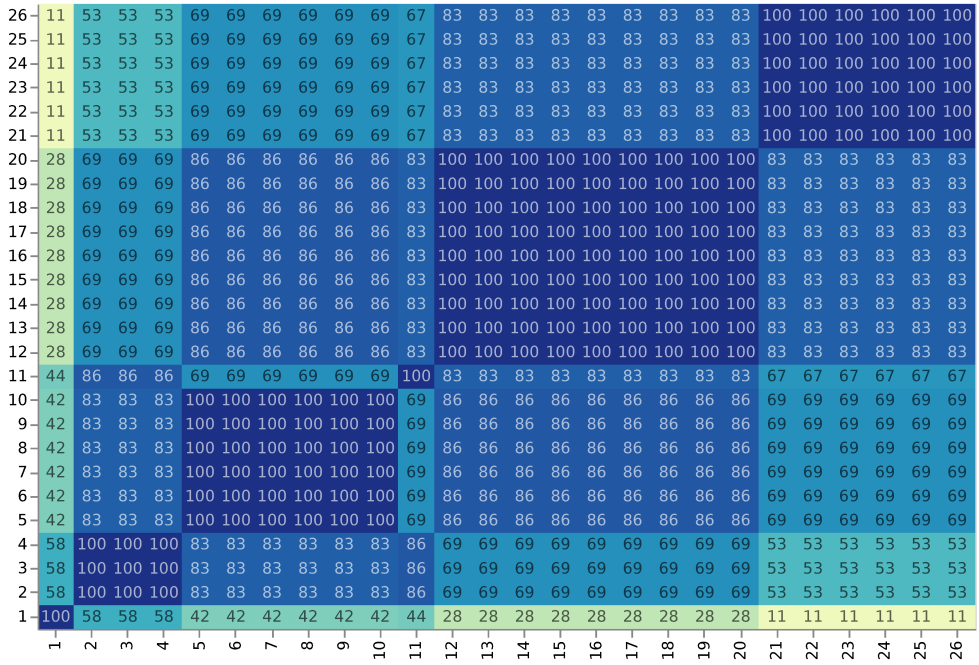
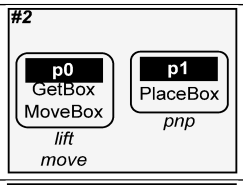
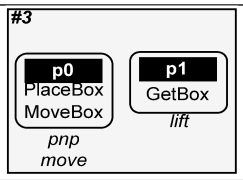
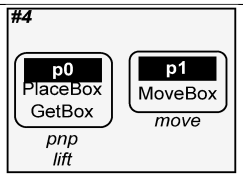


Figure 40 – Comparing structures for the given scenario regarding all attributes.

the skills $lift$ and $move$, the only solution having two positions and one level is Candidate #2.

Table 4 – Candidates #2, #3 and #4 for Feed Production scenario with three goals.

Candidate	Chart	Description
#2		Candidate #2 requires an agent with at least the skills $\{lift, move\}$ and another with at least $\{pnp\}$
#3		Candidate #3 requires an agent with at least the skills $\{pnp, move\}$ and another with at least $\{lift\}$
#4		Candidate #4 requires an agent with at least the skills $\{pnp, lift\}$ and another with at least $\{move\}$

Although when comparing the attributes, some structures are similar, in practice there can be a great difference between them, since in some cases only one of them may be feasible. Besides, there are some structures that are unique in terms of attributes, which is the case of Candidate #1 which is the only structure with only one position, and of Candidate #11 which presents three positions and just one hierarchy level (Figure 29).

Besides the presented examples that illustrate that user-defined inputs bring biases and limit the set of possibilities, it is arguably complex and demanding to provide a priori definition of roles as input to a model. It is illustrated in this work that although not requiring a priori definitions, a model that automatically synthesises positions (or roles) can reach similar (or more) results compared to user definitions. For instance, it was not defined a priori that a sector manager role/position should exist in the DSN situation illustrated in Figure 22, but the solution presented in Figure 23 has generated positions that have this purpose. As *GoOrg4DSN* may find similar solutions with fewer input parameters, it can be simpler to parametrise, easing the job of the user that is setting up the organisation generator. The parametrisation of models and their demanded design effort are discussed in Section 7.5.

The drawback of synthesising positions is the computational cost. Indeed, the automated design model have to put in the work instead of the user, who is now freed from having to specify roles or positions.

7.5 Using Goals as Input Instead of Roles and Behaviours

The definition of the system goals is often a very early step in a design. One may say that stating the goals of the system is the first step of a design since other definitions depend on it, including the definition of agents' behaviour. GoOrg and other works such as the organisation generators presented by DeLoach and Matson (2004) and Sims et al. (2008) are based on this assumption.

In contrast, the organisation generators presented by Horling and Lesser (2008) and Sierra et al. (2004) are based on roles and their expected behaviours. Their models require roles behaviours expressed by equations in which their models can calculate different situations to generate a suitable organisational structure. A behaviour is the course of actions an agent playing a particular role takes to achieve a goal. In their models, the definition of such actions should be given by the user (engineer) as input to the generator, and they are used by the generator to constrain the search for possible organisational structure descriptions.

If the user's definitions are correct, these models can generate precise and coherent organisations for a MAS. In such a case, the search is supposed to be more constrained and faster when compared to generators that tend to create wider search spaces such as GoOrg. Although it is sometimes not explicit, one can say that generators which require roles and their expected behaviours are also based on goal definitions. Indeed, roles are associated with a set of responsibilities (goals) an agent is supposed to be committed to, and behaviours are actions agents should take to achieve their responsibilities (goals).

Figure 41 illustrates sets of user's definitions that should be provided as input

to different generators. Figure 41a represents an organisation generator model such as Horling and Lesser (2008)'s and Sierra et al. (2004)'s models in which goals are previously defined (even if not explicitly), and then roles and behaviours are defined by the user and given as inputs. Figure 41b represents an organisation generator model based on GoOrg such as *GoOrg4DSN* in which goals and features are user-defined and given as inputs.

(a) Roles and behaviours as inputs.

(b) Goals and features as inputs.

Figure 41 – Comparing generators with roles and goals as input.

In the approach of Figure 41a, since roles and behaviours definition depends on goals, it may require a wider revision of inputs if the system goals change. For instance, in the DSN domain one may suggest replacing the sector approach with a less hierarchical approach in which the agent that has the stronger signal of the target follows it until passing over this duty to another agent that becomes to have the stronger signal. In this case, the goal *manage_sector* would not exist and the goal *manage_track* would be performed differently.[7] With such a change to the system goals, for a generator that requires roles as input, a revision to the roles and expected behaviours is necessary. In a model based on GoOrg, a change to the system goals requires only a revision to goals and their features, which is supposed to be a simpler job.

The kinds of inputs in the approaches presented in Figure 41 are also different. The inputs *goals* and *features* are both about **what** should be done. The input *behaviours* is about **how** things should be done. The specification of **how** things should be done must be accurate according to system runtime behaviour, which brings an extra concern when using models that require such inputs.

7.6 Summary of this discussion

To sum up what has been discussed in this chapter, it is highlighted the following aspects:

- Decoupling agents from organisations: GoOrg uses impersonal representations for agents, which decouples the organisation and the agents, allowing generating more structure candidates. It also makes the design process more flexible since it has no concern with agents' specificities and with the agents' availability.

[7] In the less hierarchical approach suggested, the agent in charge of *manage_track* would also have to communicate to other agents to compare their signal strength and negotiate a possible delegation of the goal *manage_track*.

- Planning organisational resources: GoOrg uses positions as *place-holders* for agents, which have one-to-one relationships with agents. Positions directly reflect resource needs. On a generated structure, each synthesised position represents a required resource, i.e., an agent that must occupy that position. While bringing the advantage of allowing the planning of resources and making the design process more relevant, when compared to the variety of relationships that roles and agents can present, the drawback of only providing one-to-one relationships is the loss of flexibility at runtime. Indeed, this might require organisational adaptations (such as *redesigns*) more often.

- Design bias: GoOrg synthesises positions instead of requiring user-defined roles. It reduces the effect of the user's (engineer) bias over the organisational design. However, it incurs a computational cost because a larger search space must be traversed in order to synthesise positions.

- Design inputs: GoOrg requires goals as input. Goals are essential parameters for a design since they represent **what** the organisation must achieve. GoOrg avoids inputs that are related to **how** the organisation's members must perform something since it is usually more complex to define and frequently has more bias.

- Design outputs: GoOrg generates organisational structures with attributes. The attributes are used to sort the generated structures and allow the user to pick one of the generated structures as preferred.

- Supporting organisational adaptations: GoOrg supports *structure-switchings*, *reallocations* and *redesigns*, giving more options for making an appropriate adaptation when needed. The benefits and costs associated with each procedure must be taken into account.

  - When the available agents are full of skills, they can be (re)allocated into different positions, and they can take over other agents' duties if necessary. It gives the system flexibility and robustness. However, it brings *overqualification* costs.
  - In the case of structure candidates that require the same resources as others, *structure-switching* can be a quick adaptation procedure. However, it may represent a less suitable solution according to the user's preferences (possibly affecting the system's overall behaviour).
  - A structure that satisfies the user's preferences presumably has important attributes. However, the best structure according to the user's preferences frequently requires agents that are not available, resulting in an *acquisition cost*. This cost must be balanced against the benefits of having the structure exactly as desired.

8 Conclusion

Organisational design has been refined continuously over the years. Many studies in the Administration Research Field propose theories and frameworks for this task. In the 2000s, studies on *Automated Organisational Structure Generators* have gained traction spurred on by challenges like the DSN. One may think that automating the design process could make the task easier for users. However, it is crucial to make such a process simple to be parametrised by the user. Indeed, the parametrisation complexity of some existing models may require high logic and programming skills from the user. For addressing this issue, GoOrg aims for simplicity. It is extendible for dealing with the specificity and complexity of each domain. Besides, GoOrg has great concern for simplifying its parametrisations, in which the automated position synthesising approach is highlighted.

Other models such as DeLoach and Matson (2004); Horling and Lesser (2008); Sierra et al. (2004); Sims et al. (2008) rely on the strategy of delegating to the user the definition of roles. On the one hand, these generators can prevent the generation of some incoherent structures, since the provided roles may support only feasible sets of responsibilities. On the other hand, it makes the parametrisation more complex, and constrains the range of possibilities. Indeed, such an approach would not be able to generate proper solutions if the assumptions used to define the roles are wrong. Besides, synthesising positions may generate more candidates in which feasible solutions can be filtered by its attribute.

This study adopted positions instead of roles for designing organisational structures. The reason is that positions carry the same advantage of the roles in respect to being detached from named agents, while numerically reflecting the need for resources. It means that the feasibility for a specific state of an organisation can be checked during the design.

To evaluate GoOrg, this thesis presents two specialisations. The specialisations added elements, attributes and relationships as required to generate structures for different domains. Some of the added items are similar to both specialisations, showing that existing specialisations can help to accelerate the development of extensions for other domains.

The specialisations can generate sets of structures which are candidates when considering a range of possible agents to occupy positions. The generated candidates have quantified attributes which enables a multi-criteria approach to choosing the “best” organisation. For both specialisations, distinguishing processes were defined. For instance,

the generating and binding (matching) processes are separated from each other, which supports lighter adaptation procedures. Indeed, having this distinction is crucial to foster organisation adaptation in dynamic environments and when the availability of resources is volatile. A reallocation and a structure-switching may have little or no impact on the user's preferences, and they are arguably simpler and computationally cheaper procedures if compared to a complete redesign. The proposed algorithms can solve simple problems in a satisfactory time.

However, time is not the only variable that should be taken into account on an organisation adaptation such as a switch between structures. Indeed, the impact is small when the new structure has the same number and kind of positions, in which the agents may remain in similar positions just being placed in another hierarchical arrangement. However, they still may bring extra costs considering the *overqualification cost* and the *acquisition cost*. As long as the new positions allow the maintenance of previous bindings, a reallocation is preferred to avoid extra costs. Considering possible refinement of the definitions of the goal, features and attributes, the benefits of a more suitable solution according to the user's preferences may justify the *design cost*. To help with the decision of changing or not the organisational structure, it is necessary to calculate other costs according to the attributes of the domain and to assign a weight to the different kinds of costs.

As future work, it is planned to: (i) test organisation runtime adaptations in situations that require simple reallocations to complex redesigns, and the model's applicability as an *online* design tool; (ii) adopt an existing solution or implement an algorithm that uses heuristics combined with an *anytime* approach which may produce faster answers for the search algorithm and make it suitable for more complex problems (Dean and Boddy, 1988);[1] (iii) implement a better algorithm for binding agents and positions or adopt an existing one (to determine feasibility and perhaps offer a solution to the runtime orchestration mechanism); (iv) implement more specialisations of the model to test its applicability in other domains, in practical situations and designing other kinds of structures; (v) synthesise organisational roles, their relationships, organisational norms and organisational missions; (vi) test organisational aspects such as power, span of control, accountability, and trust; (vii) finish the development and make available GoOrg integrated with JaCaMo platform (Boissier et al., 2016); and (viii) allow the use of a Directory Facilitator (DF) to retrieve the available agents and their features that can be used in the binding process.

[1] This and other planned improvements on GoOrg implementation are detailed in Appendix E.

Bibliography

Amaral, C.J., J.F. Hübner, and T. Kampik (2020a), "Towards jacamo-rest: A resource-oriented abstraction for managing multi-agent systems." In *Proceeding of 14th Workshop-School on Agents, Environments, and Applications (WESAAC 2020)*, 140–151, URL https://arxiv.org/abs/2006.05619. Citado na página 139.

Amaral, Cleber J., Vitor Luis Babireski Furio, Robson Zagre Junior, Timotheus Kampik, Maiquel de Brito, Maicon R. Zatelli, Tiago L. Schmitz, Jomi F. Hübner, and Mauri Ferrandin (2021), "Jacamo builders: Team description for the multi-agent programming contest 2020/21." In *In proceedings of MAPC 2021: The Multi-Agent Programming Contest 2021*, 134–157, URL https://doi.org/10.1007/978-3-030-88549-6_6. Citado na página 140.

Amaral, Cleber Jorge (2018), "Embedding multi-agent system frameworks : A benchmarking." In *Anais do Computer on the Beach*, 939–941, Univali, Florianópolis, URL https://periodicos.univali.br/index.php/acotb/article/download/12867/7362. Citado na página 139.

Amaral, Cleber Jorge, Sérgio Pereira Bernardes, Mateus Conceição, Jomi Fred Hübner, Luis Pedro Arenhart Lampert, Otávio Arruda Matoso, and Maicon Rafael Zatelli (2019a), "Finding new routes for integrating multi-agent systems using apache camel." In *13th Workshop-School on Agents, Environments, and Applications (WESAAC 2019)*, Florianópolis, URL http://arxiv.org/abs/1905.10490. Citado na página 139.

Amaral, Cleber Jorge, Stephen Cranefield, Jomi Fred Hübner, and Mario Lucio Roloff (2020b), "Integrating industrial artifacts and agents through apache camel." URL https://arxiv.org/abs/2006.11694. Citado na página 139.

Amaral, Cleber Jorge, Stephen Cranefield, Jomi Fred Hübner, and Mário Lucio Roloff (2019b), "Giving camel to artifacts for industry 4.0 integration challenges." *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11523 LNAI, 232–236, URL http://doi.org/10.1007/978-3-030-24209-1_20. Citado na página 139.

Amaral, Cleber Jorge and Jomi Fred Hübner (2019), "Goorg: Automated organisational chart design for open multi-agent systems." In *PAAMS* (Fernando De La Prieta, Alfonso González-Briones, Pawel Pawleski, Davide Calvaresi, Elena Del Val, Fernando Lopes,

Vicente Julian, Eneko Osaba, and Ramón Sánchez-Iborra, eds.), 318–321, Springer International Publishing, Cham, URL http://doi.org/10.1007/978-3-030-24299-2_28. Citado 2 vezes nas páginas 32 and 139.

Amaral, Cleber Jorge and Jomi Fred Hübner (2020a), "From goals to organisations: Automated organisation generator for mas." In *Engineering Multi-Agent Systems* (Louise A. Dennis, Rafael H. Bordini, and Yves Lespérance, eds.), 25–42, Springer International Publishing, Cham, URL https://doi.org/10.1007/978-3-030-51417-4_2. Citado 2 vezes nas páginas 32 and 139.

Amaral, Cleber Jorge and Jomi Fred Hübner (2020b), "Jacamo-web is on the fly: An interactive multi-agent system ide." In *Engineering Multi-Agent Systems* (Louise A. Dennis, Rafael H. Bordini, and Yves Lespérance, eds.), 246–255, Springer International Publishing, Cham, URL https://dl.acm.org/doi/10.5555/3398761.3399086. Citado na página 140.

Amaral, Cleber Jorge, Jomi Fred Hübner, and Stephen Cranefield (2022), "Generating and choosing organisations for multi-agent systems." URL https://doi.org/10.21203/rs.3.rs-1825069/v1. Citado na página 139.

Amaral, Cleber Jorge, Timotheus Kampik, and Stephen Cranefield (2020c), "A framework for collaborative and interactive agent-oriented developer operations." In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, 2092–2094, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, URL https://dl.acm.org/doi/10.5555/3398761.3399086. Citado na página 140.

Bellifemine, Fabio, Agostino Poggi, and Giovanni Rimassa (2001), "Developing multi-agent systems with a fipa-compliant agent framework." *Software: Practice and Experience*, 103–128. Citado na página 135.

Boissier, Olivier, Rafael H Bordini, Jomi F Hübner, Alessandro Ricci, and Andrea Santi (2013), "Multi-agent oriented programming with JaCaMo." *Science of Computer Programming*, 78, 747–761. Citado 3 vezes nas páginas 25, 67, and 132.

Boissier, Olivier, Jomi F Hübner, and Alessandro Ricci (2016), "The JaCaMo Framework." *Governance and Technology Series*, 30. Citado 4 vezes nas páginas 25, 37, 80, and 108.

Bordini, Rafael H., Jomi Fred Hübner, and Michael Wooldridge (2007), *Programming Multi-Agent Systems in AgentSpeak using Jason*, 1st edition. Series in Agent Technology, Wiley-Interscience. Citado na página 25.

Burns, Tom and G. M. Stalker (1994), "Mechanistic and Organic Systems of Management." In *The Management of Innovation*, volume 21, 96–125, Oxford University Press. Citado na página 37.

Burton, Richard M, Børge Obel, and Gerardine Desanctis (2011), *Organizational design: a step-by-step approach*. Cambridge University Press. Citado na página 31.

Cardoso, Rafael C and Rafael H Bordini (2019), "Decentralised Planning for Multi-Agent Programming Platforms." In *AAMAS'19: Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems*, 799–807. Citado 3 vezes nas páginas 33, 92, and 96.

Cardoso, Rafael C. and Angelo Ferrando (2021), "A review of agent-based programming for multi-agent systems." *Computers*, 10, 1–15. Citado na página 25.

Daft, Richard L. (2009), *Organization Theory and Design*, 10th edition. South-Western College Pub, Centage Learning. Citado 4 vezes nas páginas 25, 36, 41, and 137.

Dastani, Mehdi, Virginia Dignum, and Frank Dignum (2003), "Role-assignment in open agent societies." *Proceedings of the second international joint conference on Autonomous agents and multiagent systems - AAMAS '03*, 489. Citado na página 36.

De Pinho Rebouças De Oliveira, Djalma (2006), *Estrutura Organizacional: Uma Abordagem Para Resultados e Competitividade*. ATLAS EDITORA. Citado na página 31.

Dean, Thomas and Mark Boddy (1988), "An analysis of time-dependent planning." In *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence*, AAAI'88, 49–54, AAAI Press. Citado na página 108.

Decker, Keith, Katia Sycara, and Mike Williamson (1997), "Cloning for intelligent adaptive information agents." *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1286, 63–75. Citado na página 34.

Decker, Keith S. (1995), *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD Thesis, University of Massashusets. Citado 2 vezes nas páginas 33 and 96.

DeLoach, Scott A. (2002), "Modeling organizational rules in the multi-agent systems engineering methodology." In *Advances in Artificial Intelligence* (Robin Cohen and Bruce Spencer, eds.), 1–15, Springer Berlin Heidelberg, Berlin, Heidelberg. Citado na página 25.

DeLoach, Scott A and E Matson (2004), "An Organizational Model for Designing Adaptive Multiagent Systems." In *The AAAI-04 Workshop on Agent Organizations: Theory and Practice (AOTP 2004)*, 66–73, AAAI Press. Citado 8 vezes nas páginas 25, 26, 37, 87, 97, 100, 104, and 107.

Deloach, Scott A., Walamitien H. Oyenan, and Eric T. Matson (2008), "A capabilities-based model for adaptive organizations." In *Autonomous Agents and Multi-Agent Systems*, 13–56. Citado 2 vezes nas páginas 37 and 98.

Durfee, Edmund H., Victor R. Lesser, and Daniel D. Corkill (1987), "Coherent Cooperation Among Communicating Problem Solvers." *IEEE Transactions on Computers*, C-36, 1275–1291. Citado na página 36.

Ferber, Jacques and Olivier Gutknecht (1998), "A meta-model for the analysis and design of organizations in multi-agent systems." *Proceedings - International Conference on Multi Agent Systems, ICMAS 1998*, 128–135. Citado 2 vezes nas páginas 32 and 36.

Fink, S.L., R.S. Jenks, and R.D. Willits (1983), *Designing and Managing Organizations*, 1st edition. Irwin Series in Financial Planning and Insurance, R.D. Irwin. Citado 2 vezes nas páginas 33 and 36.

Furio, Vitor Luis Babireski, Maiquel de Brito, Tiago L. Schmitz, Cleber J. Amaral, Robson Zagre Junior, Maicon R. Zatelli, Mauri Ferrandin, and Timotheus Kampik (2021), "Descoberta de tamanho de mapas ilimitados através da cooperação entre agentes." In *15th Workshop-School on Agents, Environments, and Applications (WESAAC 2021)*, 178–188, Rio de Janeiro, URL `https://doi.org/10.5281/zenodo.5774181`. Citado na página 140.

Galbraith, Jay R. (1995), *Designing organizations: an executive briefing on strategy, structure, and process*. Jossey-Bass Publishers - San Francisco. Citado na página 31.

Gasser, Les (2001), *Perspectives on Organizations in Multi-agent Systems*, 1–16. Springer, Berlin, Heidelberg. Citado na página 25.

Grossi, Davide, Frank Dignum, Virginia Dignum, Mehdi Dastani, and Làmber Royakkers (2007), "Structural aspects of the evaluation of agent organizations." *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4386 LNAI, 3–18. Citado na página 86.

Hatch, M.J. (1997), *Organization Theory: Modern, Symbolic, and Postmodern Perspectives*. Oxford University Press. Citado 4 vezes nas páginas 25, 32, 35, and 37.

Horling, Bryan and Victor Lesser (2004), "A survey of multi-agent organizational paradigms." *Knowledge Engineering Review*, 19, 281–316. Citado 3 vezes nas páginas 41, 64, and 85.

Horling, Bryan and Victor Lesser (2008), "Using quantitative models to search for appropriate organizational designs." *Autonomous Agents and Multi-Agent Systems*, 16, 95–149. Citado 12 vezes nas páginas 25, 26, 38, 67, 68, 75, 97, 99, 100, 104, 105, and 107.

Hübner, Jomi Fred, Olivier Boissier, Rosine Kitio, and Alessandro Ricci (2010), "Instrumenting multi-agent organisations with organisational artifacts and agents: "Giving the organisational power back to the agents"." *Autonomous Agents and Multi-Agent Systems*, 20, 369–400. Citado na página 36.

Hübner, Jomi Fred and Jaime Simão Sichman (2003), "Organização de sistemas multiagentes." *III Jornada de MiniCursos de Inteligência Artificial JAIA03*, 8, 247–296. Citado 2 vezes nas páginas 32 and 79.

Hübner, Jomi Fred, Jaime Simão Sichman, and Olivier Boissier (2002), "A Model for the Structural, Functional, and Deontic Specification of Organizations in Multiagent Systems." In *SBIA '02: Proceedings of the 16th Brazilian Symposium on Artificial Intelligence: Advances in Artificial Intelligence*, 118–128, Springer. Citado na página 25.

Hübner, Jomi Fred, Jaime Simão Sichman, and Olivier Boissier (2006), "$\mathcal{S}$-$\mathcal{M}oise^+$: A middleware for developing organised multi-agent systems." In *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems* (Olivier Boissier, Julian Padget, Virginia Dignum, Gabriela Lindemann, Eric Matson, Sascha Ossowski, Jaime Simão Sichman, and Javier Vázquez-Salceda, eds.), 64–77, Springer Berlin Heidelberg, Berlin, Heidelberg. Citado na página 87.

Ishida, Toru, Les Gasser, and Makoto Yokoo (1992), "Organization Self-Design of Distributed Production Systems." *IEEE Transactions on Knowledge and Data Engineering*, 4, 123–134. Citado na página 34.

Kamboj, Sachin and Keith S. Decker (2007), "Organizational self-design in semi-dynamic environments." *AAMAS*, 5, 1. Citado na página 34.

Kampik, Timotheus, Cleber Jorge Amaral, and Jomi Fred Hübner (2021), "Developer operations and engineering multi-agent systems." In *Engineering Multi-Agent Systems: 9th International Workshop, EMAS 2021, Virtual Event, May 3–4, 2021, Revised Selected Papers*, 175–186, Springer-Verlag, Berlin, Heidelberg, URL `https://doi.org/10.1007/978-3-030-97457-2_10`. Citado na página 140.

Katz, Daniel and Robert Kahn (1987), *Psicologia Social da Organizações*, 3rd edition. Atlas. Citado na página 36.

Kilmann, Julie, Michael Shanahan, Andrew Toma, and Kuba Zielinski (2010), "Demystifying Organization Design." Technical report, Boston Consulting Group - BCG White Paper. Citado 2 vezes nas páginas 37 and 65.

Kota, Ramachandra, Nicholas Gibbins, and Nicholas R. Jennings (2012), "Decentralized approaches for self-adaptation in agent organizations." *ACM Transactions on Autonomous and Adaptive Systems*, 7, 1–28. Citado na página 34.

Krausburg, Tabajara, Jürgen Dix, and Rafael H. Bordini (2021), "Computing sequences of coalition structures." In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, 01–07. Citado na página 34.

Kühne, Thomas (2006), "Matters of (meta-) modeling." *Journal on Software and Systems Modeling*, 5, 369–385. Citado na página 31.

Labella, Thomas H., Marco Dorigo, and Jean-Louis Deneubourg (2007), "Division of labor in a group of robots inspired by ants' foraging behavior." *ACM Transactions on Autonomous and Adaptive Systems*, 1, 4–25. Citado na página 34.

Lesser, Victor, Charles L. Ortiz, and Milind Tambe, eds. (2003), *Distributed Sensor Networks: A Multiagent Perspective*. Springer US. Citado na página 67.

Martínez-Berumen, Héctor A., Gabriela C. López-Torres, and Laura Romo-Rojas (2014), "Developing a method to evaluate entropy in organizational systems." *Procedia Computer Science*, 28, 389–397. Citado na página 87.

Matoso, O.A., L.P.A. Lampert, J.F. Hübner, M. Conceição, S.P. Bernardes, C.J. Amaral, M.R. Zatelli, and M.L. de Lima (2020), "Agent programming for industrial applications: some advantages and drawbacks." Citado na página 139.

Matson, Eric T. and Scott A. Deloach (2005), "Autonomous organization-based adaptive information systems." *2005 International Conference on Integration of Knowledge Intensive Multi-Agent Systems, KIMAS'05: Modeling, Exploration, and Engineering*, 2005, 227–234. Citado na página 37.

McAuley, John, Joanne Duberley, and Phil Johnson (2007), *Organizational Theory: Challenges and Perspectives*, 1st edition. Prentice-Hall. Citado na página 35.

Mintzberg, H. and L. Van der Heyden (1999), "Organigraphs: drawing how companies really work." *Harvard Business Review*, 77. Citado 2 vezes nas páginas 47 and 99.

Mintzberg, Henry (1983), *Structure in fives*, 1st edition. Prentice-Hall. Citado na página 36.

Newman, Derek A. (1973), *Organization Design: An analytical approach to the structuring of organisations*, 1st edition. Edward Arnold. Citado na página 37.

Ohta, Naoki, Atsushi Iwasaki, Makoto Yokoo, Kohki Maruono, Vincent Conitzer, and Tuomas Sandholm (2006), "A compact representation scheme for coalitional games in open anonymous environments." *Proceedings of the National Conference on Artificial Intelligence*, 1, 697–702. Citado na página 34.

Pattison, H. Edward, Daniel D. Corkill, and Victor R. Lesser (1987), "Chapter 3 - instantiating descriptions of organizational structures." In *Distributed Artificial Intelligence* (Michael N Huhns, ed.), 59 – 96, Morgan Kaufmann. Citado na página 35.

Pettigrew, Andrew M. and Evelyn M. Fenton (2000), *The Innovating organization*, 1st edition. SAGE Publications. Citado na página 37.

Rahwan, Talal, Tomasz P. Michalak, Michael Wooldridge, and Nicholas R. Jennings (2015), "Coalition structure generation: A survey." *Artificial Intelligence*, 229, 139–174. Citado 2 vezes nas páginas 25 and 34.

Robbins, Stephen and Mary Coulter (2012), *Management*, 11th edition. Prentice-Hall. Citado na página 37.

Rosen, Kenneth H (2012), *Discrete mathematics and its application*, 7th edition. McGraw-Hill. Citado na página 89.

Seidewitz, Ed (2003), "What models mean." *IEEE Software*, 20, 26–32. Citado na página 43.

Shehory, O., K. Sycara, P. Chalasani, and S. Jha (1998), "Agent cloning: an approach to agent mobility and resource allocation." *IEEE Communications Magazine*, 36, 58, 63–67. Citado na página 34.

Sierra, Carles, Jordi Sabater, J. Augusti, and Pere Garcia (2004), "The SADDE Methodology: Social agents design driven by equations." *Methodologies and software engineering for agent systems. Springer - Boston.* Citado 8 vezes nas páginas 25, 26, 37, 99, 100, 104, 105, and 107.

Sims, Mark, Daniel Corkill, and Victor Lesser (2004), "Separating domain and coordination in multi-agent organizational design and instantiation." *Proceedings - IEEE/WIC/ACM International Conference on Intelligent Agent Technology. IAT 2004*, 155–161. Citado na página 32.

Sims, Mark, Daniel Corkill, and Victor Lesser (2008), "Automated organization design for multi-agent systems." *Autonomous Agents and Multi-Agent Systems*, 16. Citado 9 vezes nas páginas 25, 26, 34, 35, 38, 97, 100, 104, and 107.

Slade, Samantha (2018), *Going Horizontal: Creating a Non-Hierarchical Organization, One Practice at a Time*, 1st edition. Berrett-Koehler Publishers, Inc. Citado na página 26.

Sleight, Jason (2014), "Agent aware organizational design (doctoral consortium)." In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, 1739–1740, Paris, France. Citado 2 vezes nas páginas 33 and 96.

Sleight, Jason Lee, Edmund H Durfee, Satinder Singh Baveja, Associate Amy E M Cohn, and Emeritus Victor R Lesser (2015), *Agent-Driven Representations, Algorithms, and Metrics for Automated Organizational Design*. Ph.D. thesis, University of Michigan. Citado na página 34.

So, Young-Pa and Edmund H Durfee (1998), "Designing Organizations for Computational Agents." *Computational Organization Theory (Simulating Organizations)*, 2, 47–64. Citado 5 vezes nas páginas 25, 26, 34, 36, and 86.

Stoner, J.A.F. and R.E. Freeman (1992), *Management*, 1st edition. Prentice-Hall. Citado 2 vezes nas páginas 31 and 37.

Summerfield, Fraser (2016), "Matching skill and tasks: Cyclical fluctuations in the overqualification of new hires." Citado na página 90.

Tambe, Milind (1997), "Towards Flexible Teamwork." *Journal of Artificial Intelligence Research*, 7, 83–124. Citado na página 32.

Uez, Daniela Maria and Jomi Fred Hübner (2014), "Environments and organizations in multi-agent systems: From modelling to code." In *Engineering Multi-Agent Systems* (Fabiano Dalpiaz, Jürgen Dix, and M. Birna van Riemsdijk, eds.), 181–203, Springer International Publishing, Cham. Citado na página 37.

von Bertalanffy, Ludwig (1968), *General System Theory: Foundations, Development, Applications*. Penguin University Books. Citado na página 87.

Wu, Zhaohui, Shuiguang Deng, and Jian Wu (2015), "Chapter 7 - service composition." In *Service Computing* (Zhaohui Wu, Shuiguang Deng, and Jian Wu, eds.), 177–227, Academic Press, Boston, URL `https://www.sciencedirect.com/science/article/pii/B9780128023303000072`. Citado 2 vezes nas páginas 31 and 32.

Ye, Dayong, Minjie Zhang, and Danny Sutanto (2014), "Cloning, resource exchange, and relation adaptation: An integrative self-organisation mechanism in a distributed agent network." *IEEE Transactions on Parallel and Distributed Systems*, 25, 887–897. Citado na página 34.

Ye, Dayong, Minjie Zhang, and Athanasios V Vasilakos (2016), "A Survey of Self-organisation Mechanisms in Multi-Agent Systems." *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47. Citado na página 34.

APPENDIX A – Comparing organisational attributes among candidates

Regarding organisational attributes in the context of organisation adaptation procedures, this section is presenting similarities of organisational attributes for the Feed Production scenario (Fig 28). Along with the comparison illustrated in Figure 33, Figure 42 shows a correlation matrix of the *height* of the 26 generated structures for the motivating scenario of this analysis. In this example, there are four candidates with the same *height* of structure #1, which has only one hierarchy level (Figure 29). In terms of *height*, a structure switch, for instance, from structure #1 to structure #2, has no impact (at least for this user's preference). However, the comparison shows that, for instance, structure #1 and structure #21 are relatively 0% similars in terms of *height* since structure #21 is the tallest structure with three levels.

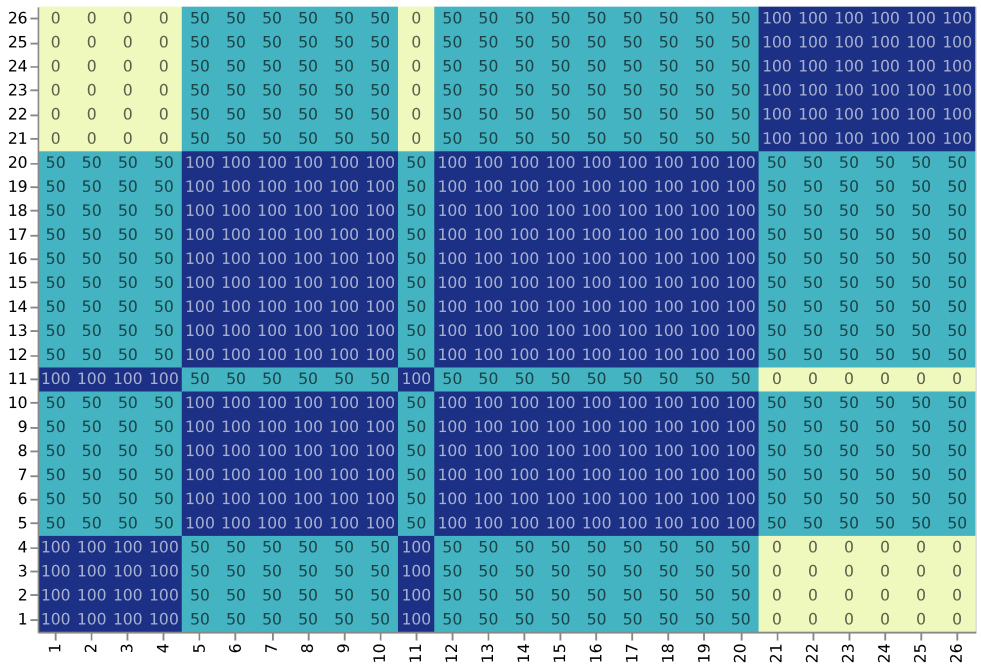


Figure 42 – Comparing the height of structures for the given scenario.

Figure 43 shows a correlation matrix of the *efficiency* of the 26 generated structures for the motivating scenario. In this scenario, Candidate #1 is the only structure having only one position, being 100% efficient (relatively to the others). From Candidate #2 to Candidate #10, all of them have two positions, i.e., the efficiency is the same, which is

50% of the efficiency of Candidate #1. The rest of the candidates have three positions, thus comparing to the most efficient structure they are 33% efficient.

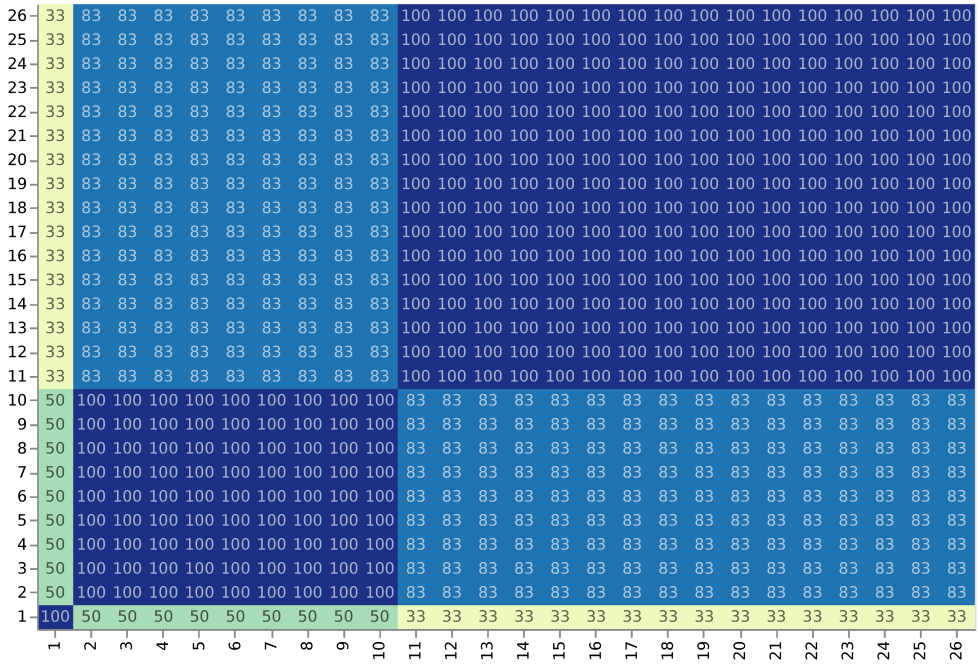


Figure 43 – Comparing the efficiency of structures for the given scenario.

APPENDIX B – XML specification of Feed Production with four goals

One of the motivating examples used in this work specifies the Feed Production scenario with four goals. It generates 1646 candidates, which illustrates how a considerable quantity of candidates can be generated, and how they differ from each other. The workload grain is smaller than the workload associated with the goal **MoveBox**, so it is split into two parts. Figure 10 illustrates the set of goals of this scenario in which it is necessary to access a database to get requests (represented by the goal **FeedProduction**), then **GetBox** from shelves, **MoveBox** to near a conveyor belt in which the goal **PlaceBox** must be achieved. The dataloads specified do not constrain the search for solutions, since the grain size is larger than the specification of this particular example. Figure 13 illustrates the available agents of this specification. This organisation specification is used to generate candidates executing the following command:

```
$ ./gradlew run --args="examples/Feed_production_line.xml \
          FLATTER EFFICIENT GENERALIST"
```

Listing B.1 shows the content of the file `examples/Feed_production_line.xml` of *GoOrg4Prod* project. The nested elements in *automated-design-parameters* specify design parameters such as the maximum workload and dataload allowed per position and the grain size of these features. The nested elements in *functional-specification* specify the organisational goals (the set G). The nested elements in *available-agents* specify the available agents (the set A).

Listing B.1 – Feed Production 4 goals XML $\mathcal{M}oise^{+}$ like specification.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?xml-stylesheet href="http://moise.sourceforge.net/xml/os.xsl"
4    type="text/xsl" ?>
5
6  <organisational-specification id="organisation" os-version="0.8"
7
8    xmlns='http://moise.sourceforge.net/os'
9    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
10   xsi:schemaLocation='http://moise.sourceforge.net/os
11                       http://moise.sourceforge.net/xml/os.xsd'>
12
13   <automated-design-parameters>
```

```
14      <parameter id="maxWorkload" value="24.0"/>
15      <parameter id="maxDataLoad" value="1000.0"/>
16      <parameter id="workloadGrain" value="4.0"/>
17      <parameter id="dataLoadGrain" value="1000.0"/>
18      <parameter id="oneSolution" value="false"/>
19    </automated-design-parameters>
20
21    <functional-specification>
22      <scheme id="scheme">
23        <goal id="FeedProduction">
24          <workload id="db_access" value="0.10"/>
25          <dataload id="request box" value="8.00" recipient="GetBox"/>
26          <plan operator="sequence">
27            <goal id="GetBox">
28              <workload id="lift" value="4.00"/>
29              <dataload id="box ready" value="8.00" recipient="MoveBox"/>
30            </goal>
31            <goal id="MoveBox">
32              <workload id="move" value="8.00"/>
33              <dataload id="items ready" value="8.00" recipient="PlaceBox"/>
34            </goal>
35            <goal id="PlaceBox">
36              <workload id="pnp" value="1.00"/>
37            </goal>
38          </plan>
39        </goal>
40      </scheme>
41    </functional-specification>
42
43    <available-agents>
44      <agent id="bt">
45        <skill id="move"/>
46      </agent>
47      <agent id="pp">
48        <skill id="pnp"/>
49      </agent>
50      <agent id="ie">
51        <skill id="db_access"/>
52        <skill id="lift"/>
53      </agent>
54    </available-agents>
55
56  </organisational-specification>
```

APPENDIX C – XML specification of DSN with 4x5 sensors and 3 targets

For the DSN domain, a motivating example used in this work specifies a scenario with five sensors in each of the four sectors, and three targets being detected. Figure 26 illustrates the first candidate generated for this scenario. The set of goals for this scenario is an extension of the set G illustrated in Figure 22. In this case, there are seven goals, the four goals illustrated in the mentioned figure, plus three goals for tracking three targets. To generate candidates in the order that this work has shown, it is being used the default preferences criteria which are the *nearest* agents and most *idle* structure. Thus, this organisation specification is used to generate candidates executing the following command:

```
$ ./gradlew run --args="examples/dsn.xml IDLE NEAREST"
```

Listing C.1 shows the content of the file `examples/dsn.xml` of *GoOrg4DSN* project. The nested elements in *functional-specification* specify the organisational goals (the set G). The nested elements in *available-agents* specify the available agents (the set A).

Listing C.1 – DSN 4x5 sensors + 3 targets XML $\mathcal{M}oise^{+}$ like specification.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?xml-stylesheet href="http://moise.sourceforge.net/xml/os.xsl"
4    type="text/xsl" ?>
5
6  <organisational-specification id="house_contruction" os-version="0.8"
7
8    xmlns='http://moise.sourceforge.net/os'
9    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
10   xsi:schemaLocation='http://moise.sourceforge.net/os
11                       http://moise.sourceforge.net/xml/os.xsd'>
12
13   <functional-specification>
14     <scheme id="dsn_sch">
15       <goal id="manage_sector_NW">
16         <workload id="manage_sector" value="0.6"/>
17         <sector id="nw"/>
18         <plan operator="sequence">
```

```
19          <goal id="manage_sector_NE">
20            <workload id="manage_sector" value="0.6"/>
21            <sector id="ne"/>
22          </goal>
23          <goal id="manage_sector_SW">
24            <workload id="manage_sector" value="0.6"/>
25            <sector id="sw"/>
26          </goal>
27          <goal id="manage_sector_SE">
28            <workload id="manage_track" value="0.6"/>
29            <sector id="se"/>
30          </goal>
31          <goal id="track_1">
32            <workload id="manage_sector" value="0.2"/>
33            <sector id="se"/>
34          </goal>
35          <goal id="track_2">
36            <workload id="manage_sector" value="0.2"/>
37            <sector id="se"/>
38          </goal>
39          <goal id="track_3">
40            <workload id="manage_sector" value="0.2"/>
41            <sector id="nw"/>
42          </goal>
43        </plan>
44      </goal>
45    </scheme>
46  </functional-specification>
47
48  <available-agents>
49    <!-- 5 sensor on Sector nw -->
50    <agent id="sensor_02_12">
51      <sector id="nw"/>
52    </agent>
53    <agent id="sensor_02_18">
54      <sector id="nw"/>
55    </agent>
56    <agent id="sensor_05_15">
57      <sector id="nw"/>
58      <intent id="manage_sector"/>
59    </agent>
60    <agent id="sensor_08_12">
61      <sector id="nw"/>
62    </agent>
63    <agent id="sensor_08_18">
64      <sector id="nw"/>
65    </agent>
```

```
66      <!-- 5 sensor on Sector ne -->
67      <agent id="sensor_12_12">
68        <sector id="ne"/>
69      </agent>
70      <agent id="sensor_12_18">
71         <sector id="ne"/>
72      </agent>
73      <agent id="sensor_15_15">
74        <sector id="ne"/>
75        <intent id="manage_sector"/>
76      </agent>
77      <agent id="sensor_18_12">
78        <sector id="ne"/>
79      </agent>
80      <agent id="sensor_18_18">
81        <sector id="ne"/>
82      </agent>
83      <!-- 5 sensor on Sector sw -->
84      <agent id="sensor_02_02">
85        <sector id="sw"/>
86      </agent>
87      <agent id="sensor_02_08">
88        <sector id="sw"/>
89      </agent>
90      <agent id="sensor_05_05">
91        <sector id="sw"/>
92        <intent id="manage_sector"/>
93      </agent>
94      <agent id="sensor_08_02">
95        <sector id="sw"/>
96      </agent>
97      <agent id="sensor_08_08">
98        <sector id="sw"/>
99      </agent>
100     <!-- 5 sensor on Sector se -->
101     <agent id="sensor_12_02">
102       <sector id="se"/>
103     </agent>
104     <agent id="sensor_12_08">
105       <sector id="se"/>
106     </agent>
107     <agent id="sensor_15_05">
108       <sector id="se"/>
109       <intent id="manage_sector"/>
110     </agent>
111     <agent id="sensor_18_02">
112       <sector id="se"/>
```

```
113     </agent>
114     <agent id="sensor_18_08">
115       <sector id="se"/>
116     </agent>
117   </available-agents>
118
119 </organisational-specification>
```

APPENDIX D – XML specification and outputs for Feed Production with three goals

Listing D.1 shows the content of the file `Feed_production_line_evaluation.xml` of *GoOrg4Prod* project, which specifies the Feed Production scenario with three goals used as a motivating example in this work. It specifies three goals with some workload associated. This is a suitable example to depict the way *GoOrg4Prod* synthesises positions and searches for 26 candidates it generates. From the parameters the goals are not split, they remain as original. The parameters also allow assigning all goals to a unique position since the *maxWorkload* is 24 and each workload is 8. Figure 28 illustrates this scenario in which it is necessary to **GetBox** from shelves, **MoveBox** to near a conveyor belt in which the goal **PlaceBox** must be achieved. The two dataloads specified do not constrain the search for solutions, since the grain size is larger than the specification of this particular example. Although the specification of available agents is not being discussed in this work, from the file description it can be inferred that only feasible structures are the most specialised ones (with three positions), since each agent has only one skill. This organisation specification is used to generate candidates executing the following command:

```
$ ./gradlew run --args="examples/Feed_production_line_evaluation.xml \
          FLATTER EFFICIENT GENERALIST"
```

Listing D.1 – Feed Production 3 goals XML $\mathcal{M}oise^{+}$ like specification.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?xml-stylesheet href="http://moise.sourceforge.net/xml/os.xsl"
4    type="text/xsl" ?>
5
6  <organisational-specification id="organisation" os-version="0.8"
7
8    xmlns='http://moise.sourceforge.net/os'
9      xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
10   xsi:schemaLocation='http://moise.sourceforge.net/os
11                       http://moise.sourceforge.net/xml/os.xsd'>
12
13   <automated-design-parameters>
14     <parameter id="maxWorkload" value="24.0"/>
```

```
15      <parameter id="maxDataLoad" value="1000.0"/>
16      <parameter id="workloadGrain" value="8.0"/>
17      <parameter id="dataLoadGrain" value="1000.0"/>
18      <parameter id="oneSolution" value="false"/>
19    </automated-design-parameters>
20
21    <functional-specification>
22      <scheme id="scheme">
23        <goal id="GetBox">
24          <workload id="lift" value="8.00"/>
25          <dataload id="box ready" value="8.00" recipient="MoveBox"/>
26          <plan operator="sequence">
27            <goal id="MoveBox">
28              <workload id="move" value="8.00"/>
29              <dataload id="items ready" value="8.00"
30                recipient="PlaceBox"/>
31            </goal>
32            <goal id="PlaceBox">
33              <workload id="pnp" value="8.00"/>
34            </goal>
35          </plan>
36      </goal>
37      </scheme>
38    </functional-specification>
39
40    <available-agents>
41      <agent id="bt">
42        <skill id="move"/>
43      </agent>
44      <agent id="pp">
45        <skill id="pnp"/>
46      </agent>
47      <agent id="ie">
48        <skill id="lift"/>
49      </agent>
50    </available-agents>
51
52  </organisational-specification>
```

As presented previously in this work, Table 3, shows the statistic output file that was generated for this particular scenario. In the following, Listing D.2 shows a JCM file generated for Candidate #11. For this candidate, the agents `pp`, `ie` and `bt` are lined up for the organisation.[1]

Listing D.2 – A JCM file generated by GoOrg.

```
1  /* JCM created automatically by GoOrg */
```

[1] As discussed in Appendix E, GoOrg current implementation is not fully compatible with JaCaMo.

```
2 mas Feed_production_line {
3   agent pp
4   agent bt
5   agent ie
6 }
```

APPENDIX E – Improving GoOrg

To bind agents and positions, GoOrg specialisations presented in this work have used the FirstFit algorithm (Algorithm 3). This algorithm registers the first match of a requirement and a resource. It is not optimal and it is not complete, since the first match may not be the best match. For instance, if the first match is between an agent with many capabilities and a position that can be occupied by many other agents, there may have no agent left to occupy another position that requires more capabilities. For a more elaborated matching process, other algorithms can be added. It is necessary to extend the class *Fit* and to implement the method *fitRequirements/3*

The search for states approach that GoOrg presented specialisations uses is the Breadth-First algorithm. This algorithm is optimal and complete, but computationally expensive in terms of memory and consumed time for finding solutions. GoOrg project already provides many other algorithms such as Depth-First, Hill Climbing and A* search algorithms. Other search algorithms can also be added. However, the *Organisation* class on the package *organisation.search*, which defines a search state have to be adapted for the chosen search technique.

GoOrg does not orchestrate a running MAS, it assumes that this task is performed by another mechanism. In this sense, it is necessary to improve the integration between GoOrg and an orchestrating mechanism. A mechanism that can be used for orchestrating agents is $\mathcal{M}oise^+$, which is part of the JaCaMo project. $\mathcal{M}oise^+$ has many possible configurations for different organisational compositions, constraints and behaviours. At least in a limited range of configurations, the following improvements provide better integration between GoOrg and $\mathcal{M}oise^+$ (JaCaMo).

- synthesise organisational roles, their relationships and groups of roles;
- synthesise organisational missions;
- synthesise organisational norms;
- export a $\mathcal{M}oise^+$ XML file with the systhesised elements.

$\mathcal{M}oise^+$ uses the concepts of roles and missions. A role is defined as a set of missions. A mission is a set of constraints that must be respected in achieving a set of

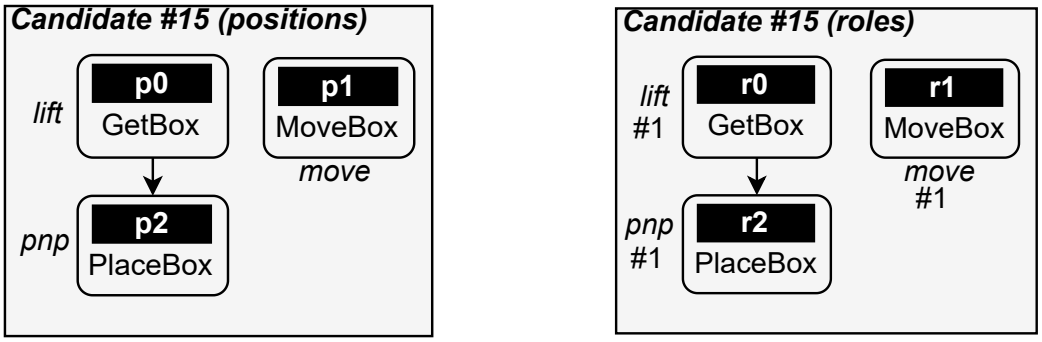
goals. Roles are bound with missions through norms. As an example, it is considered the *building a house* example (Boissier et al., 2013). In this example, the mission *paint_house* expects a minimum of one agent and a maximum of one agent to achieve the goals *exterior_painted* and *interior_painted*. A role *painter* is defined as belonging to a group called *house_group*, and there must have a minimum of one agent and a maximum of one agent playing the role *painter*. There is a norm to obligate the role *painter* to accomplish the mission *paint_house*. Therefore, the only agent that plays the role *painter* should be committed (actually, it is seen as obligated) to achieving the goals *exterior_painted* and *interior_painted*, accomplishing the mission *paint_house*.

For synthesising roles a preliminary study is presented in Appendix F. In a $\mathcal{M}oise^{+}$ XML file, the organisational roles are defined in the *structutal-specification* as simple identifiers. In a JaCaMo MAS, an agent can be associated with an identified role in the JCM file or it may also be hardcoded in the agent's code. In both cases, it is necessary that the organisation and the agent use the same role identification. For instance, an organisational specification may define the role *transporter* and on the JCM or on the agent's code, the agent is defined to play the role *transporter*. Following the preliminary study about synthesising roles, it is also possible to establish authority relationships between roles based on the authority relationships of synthesised positions. From the number of times that a role appears in an organisational structure of positions, the cardinality of each role can be inferred to form a single group.

After synthesising roles from positions, each role is associated with a set of goals which can be used to synthesise missions. Indeed, a mission has assigned a set of goals. Then, a norm is synthesised to bound a mission with a role. A simplistic method can infer the cardinality associated with a mission as the cardinality of the role.

To illustrate all the mentioned possible improvements, it is considered Candidate #15 of the Feed Production scenario which is shown in Figure 44. In Figure 44a, the candidate is represented as a structure of positions with assigned goals and the necessary skills to achieve them, a result produced by GoOrg. Figure 44b illustrated synthesised roles and relationships of the mentioned candidate using either of the methods described in Appendix F. The assigned goals and necessary skills are also present. The cardinality of each role is added as a result of the number of positions that refers to each synthesised role. In this example, there is no situation with two or more positions of the same role, which should produce a cardinality of this role greater than one. Besides, the structures are similar since each position is associated with a unitary and distinct sets of goals.

To export the *structural-specification* of $\mathcal{M}oise^{+}$ it is considered the synthesised roles `r0`, `r1` and `r2` in Figure 44b. The $\mathcal{M}oise^{+}$ XML nested elements in *role-definitions* can be defined just with the mentioned identifiers. Every role in this example has the cardinality equals one. With roles cardinalities the nested elements in *roles* of *group-*



(a) Candidate #15 as a structure of positions.

(b) Candidate #15 as a structure of roles.

Figure 44 – Synthesing roles, relationships, missions and norms from a GoOrg's output.

specification can also be defined. The only relationship in this example is an authority of `r0` over `r2`, which defines the nested elements in *links* of *group-specification* The nested element in *formation-constraints* of *group-specification* can be an option to be defined by the user at design time. Listing E.1 shows how these exported $\mathcal{M}oise^{+}$ XML elements may looks like.

Listing E.1 – A $\mathcal{M}oise^{+}$ XML structural-specification from synthesised roles and relationships.

```
<structural-specification>
  <role-definitions>
    <role id="r0" />
    <role id="r1" />
    <role id="r2" />
  </role-definitions>

  <group-specification id="g0">
    <roles>
      <role id="r0" min="1" max="1"/>
      <role id="r1" min="1" max="1"/>
      <role id="r2" min="1" max="1"/>
    </roles>
    <links>
      <link from="r0" type="authority" to="r2" scope="intra-group" />
    </links>
    <formation-constraints>
      <compatibility from="org" to="org" scope="intra-group" />
    </formation-constraints>
  </group-specification>
</structural-specification>
```

Considering that the *functional-specification* has already a scheme given as input, the rest of this specification can also be exported from the synthesised elements. An approach may consider that each mission is formed by the goals associated with a synthesised role, and the cardinality is the same cardinality of the related role, as illustrated

in Listing E.2. The minimal and maximum cardinalities in these cases are equal values based on the cardinality of the synthesised roles.

Listing E.2 – A $\mathcal{M}oise^{+}$ XML functional-specification from synthesised roles and relationships.

```
<functional-specification>
  <scheme id="scheme">
    <!-- This should be given as input-->
    <goal id="GetBox">
      <workload id="lift" value="8.00"/>
      <dataload id="box ready" value="8.00" recipient="MoveBox"/>
      <plan operator="sequence">
        <goal id="MoveBox">
          <workload id="move" value="8.00"/>
          <dataload id="items ready" value="8.00" recipient="PlaceBox"/>
        </goal>
        <goal id="PlaceBox">
          <workload id="pnp" value="8.00"/>
        </goal>
      </plan>
    </goal>

    <!-- This can be synthesised-->
    <mission id="m0" min="1" max="1">
      <goal id="GetBox"/>
    </mission>
    <mission id="m1" min="1" max="1">
      <goal id="MoveBox"/>
    </mission>
    <mission id="m2" min="1" max="1">
      <goal id="PlaceBox"/>
    </mission>
  </scheme>
</functional-specification>
```

The *normative-specification* bound the related role to each mission, as illustrated in Listing E.3.

Listing E.3 – A $\mathcal{M}oise^{+}$ XML normative-specification from synthesised roles and relationships.

```
<normative-specification>
  <norm id="n1" type="obligation" role="r0" mission="m0" />
  <norm id="n2" type="obligation" role="r1" mission="m1" />
  <norm id="n3" type="obligation" role="r2" mission="m2" />
</normative-specification>
```

With this improvement, GoOrg may provide more outputs without requiring other inputs. Indeed, the XML file provided to GoOrg having only the scheme (in the *functional-specification*), could be filled with the *structural-specification* with *role-definitions*, *group-*

specification and *links* representing relationships. It also can be filled with the *missions* in the *functional-specification* and with the *normative-specification*.

However, there would have still missing data. A role in $\mathcal{M}oise^{+}$ is usually a simple identification with no more data associated, which could bring a challenge for designed agents to be bound with automated synthesised roles. For instance, in Listing 6.1, the goal *MoveCrate* can be assigned to the synthesised position identified as `p1`, which could be synthesised as a role identified by `r1`. However, `r1` would have no meaning for a designed agent. In fact, `r1` is an arbitrary identifier that has no special meaning and does not provide information to understand what is behind such a role. In this sense, besides synthesising roles from positions it is also necessary to provide a way correlate agents and roles.

To address this issue, it is necessary to bind agents to roles (instead of positions). This information can be extracted from the bindings between agents and positions. For instance, the agent `a1` that is bound with the position `p1` which is associated with the roles `r1` and `r2`, should be bound with `r1` and `r2`. Thus, the bindings between agents and roles can be exported to the JaCaMo project file (JCM). Currently, GoOrg only generates a file with the name of the agents that are bound with any synthesised position. With this improvement, GoOrg should also export an organisation instance specification which may contain an arbitrary organisation identifier, a reference to the synthesised group that should be responsible to achieve the goals scheme given as input and that contains the bindings between agents and roles.

Another possible improvement is to make GoOrg retrieve the set of available agents from a DF. A DF as specified by Foundation for Intelligent Physical Agents (FIPA) has registers of agents and the services they provide (Bellifemine et al., 2001).[1] In the DF an agent should have only one entry, but its entry may have multiple services. Thus, among possibilities of the use of the DF, in an approach, the services of DF can be used as capabilities that must match with features associated with organisational positions. For instance, the agent *bob* can be registered in the DF with the service *send_budget* and *send_invoice*, meaning that *bob* is able to *send_budget* and to *send_invoice*. In GoOrg, the services *send_budget* and *send_invoice* must be added to the set of goals, for instance, as *workloads*. Considering that the given set of goals uses the same identifiers regarding agents' capabilities, a function can find the DF published service that matches with each of the synthesised positions by their associated workloads (skills). The use of other optional fields of the DF can also make possible to find meaningful role identifiers. For instance, the services *send_budget* and *send_invoice* can be both registered with the service type *seller*, referring to an associated role. Using a DF, the set of *available-agents* does not

[1] The document FIPA Agent Management Specification is available at `http://www.fipa.org/specs/fipa00023/SC00023K.html`.

need to be given in the XML $\mathcal{M}oise^+$ file.

APPENDIX F – Synthesizing organisational roles

The concept of roles is also present in a structure made up of positions. Among other characteristics, a role is a set of responsibilities (Daft, 2009). In this sense, from the goals that are assigned to a position, it is possible to synthesise the role(s) an agent bound to this position plays. This section presents a motivating scenario and possible methods that can be applied to synthesise roles from sets of goals.

To illustrate how to synthesise roles, it is considered the marketplace organisational structure of Figure 45 which has four positions and no hierarchy. *Positions_1* and *Position_2* have the same goals to achieve: *Pack* and *Send Product*. *Position_3* has to achieve the goal *Buy*. *Position_4* has to achieve the goals *Buy* and *Store Product*. Each role is a set of goals (responsibilities). This figure also illustrates two possible methods that can be used to synthesise roles from positions.

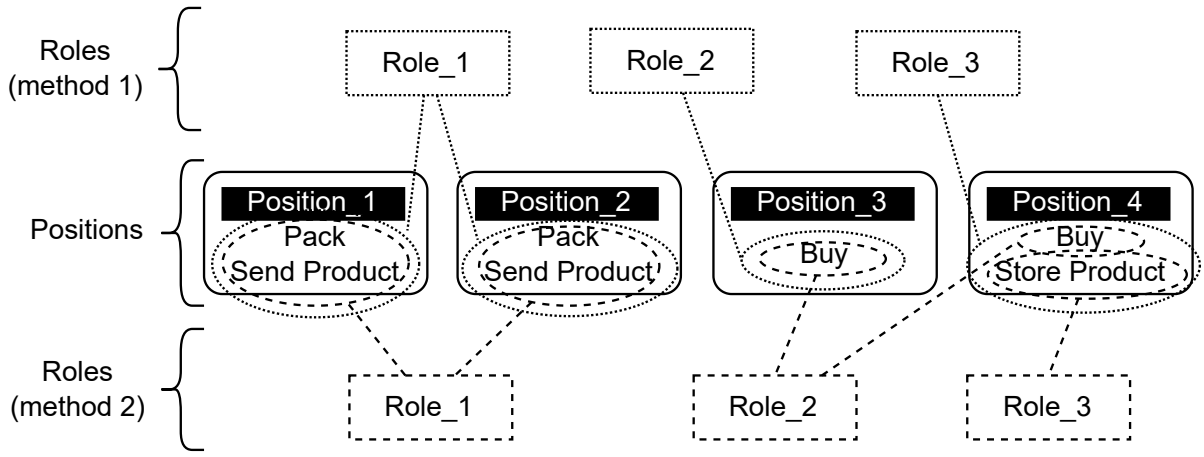


Figure 45 – The identification of the roles associated with positions.

The first method considers that a role can be synthesised considering the whole set of goals each position is associated with. In this method, a role is identified on *Position_1* and *Position_2* which share the same associated goals *Pack* and *Send Product*. *Position_3* has another specific set of goals (containing the goal *Buy*), which makes this set another role. The last role is synthesised from the set of goals present in *Position_4* which has the elements *Buy* and *Store Product*.

Finding sets of goals that along positions are always together can be another

method to synthesise roles. The first synthesised role using this method is, again, the set of goals formed by the elements *Pack* and *Send Product* that are present in *Position_1* and *Position_2*. A second role is associated with a set to a unique element, the goal *Buy*, which is found alone in *Position_3*. *Position_4* is also associated with this second role. The third role is associated with the remaining goal *Store Product* associated with *Position_4*, which apart of the goal *Buy* forms another set of an unique element. Notice that the *Position 4* is associated with two roles (named *Role_2* and *Role_3*).

In this example, both methods generated the same number of roles (three each). The second method tends to generate roles with fewer goals associated with each and tends to foster situations in which agents should enact more roles simultaneously.

APPENDIX G – Works developed during the PhD

The motivation and plan to solve the problem addressed by this thesis were first presented by Amaral and Hübner (2019) as an ongoing work at the 17th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS) - in Ávila (Spain) 2019 - with the title "Goorg: Automated organisational chart design for open multi-agent systems". The work was refined and presented by Amaral and Hübner (2020a) in the 10th workshop on Engineering Multi-Agent Systems (EMAS) - Montreal (Canada) 2019 - with the title "From goals to organisations: Automated organisation generator for MAS". Amaral, Hübner, and Cranefield (2022) have gathered the most relevant achievements and conclusions of this thesis in a journal paper that is currently under review.

Other works were also developed since the beginning of this PhD research, in October 2017. Amaral (2018) has published a benchmarking among Multi-Agent System platforms that were embedded in a Raspberry Pi2. This research was performed while attending classes on Multi-Agent Systems at UFSC. Later, motivated by the project AG-BR of Petrobras, which aimed to check possible applications of agents in the Oil and Gas industry, a team of researchers worked especially on the integration of agents and other existing software and hardware artefacts. Based on a previous background, Amaral, Cranefield, Hübner, and Roloff (2019b) and Amaral, Cranefield, Hübner, and Roloff (2020b) have continued studies on integrating a MAS with external non-autonomous entities, in the context of Industry 4.0, publishing these two research about the development of an Apache Camel component for CArtAgO software artefacts. Amaral, Bernardes, Conceição, Hübner, Lampert, Matoso, and Zatelli (2019a) have designed a new Apache Camel component for integrating both non-autonomous and autonomous (agents) external software artefacts. Matoso, Lampert, Hübner, Conceição, Bernardes, Amaral, Zatelli, and de Lima (2020) published the main achievements and conclusions of this project, summarising the advantages and drawbacks of using MAS in the context of the Oil and Gas industry. The integration challenges and also the lack of some programming facilities for developing MAS have motivated other studies that followed the project AG-BR of Petrobras. Amaral, Hübner, and Kampik (2020a) worked in Jacamo-REST, a resource-oriented abstraction for managing MAS, which provided endpoints to access a MAS from external software artefacts. This tool associated with the previous works on Apache Camel compo-

nents provides tooling for the MAS to act as a server to be consumed by external entities, and as a client for consuming external services. Besides providing endpoints for external entities consuming a MAS services, Jacamo-REST also provides tools for managing the MAS, allowing the development of a system interactively while it is running. Following this concept, Amaral and Hübner (2020b) presented Jacamo-WEB, an Integrated Development Environment (IDE) that facilitates the development of MAS on-the-fly. Amaral, Kampik, and Cranefield (2020c) have extended Jacamo-WEB providing new tooling for collaborative and interactive development of MAS. During this time, another team has joined to participate in the 2020/21 Multi-Agent Programming Contest (MAPC), as registered by Amaral, Furio, Junior, Kampik, de Brito, Zatelli, Schmitz, Hübner, and Ferrandin (2021) and by Furio, de Brito, Schmitz, Amaral, Junior, Zatelli, Ferrandin, and Kampik (2021). GoOrg has been taken into consideration in the MAPC, and a few strategies have been proposed to parametrise the model. However, the team spent its effort developing the system's fundamental elements, leaving no time to apply an orchestrating mechanism like $\mathcal{M}oise^{+}$. The new challenges that collaborative development has brought, which were experienced during the MAPC, such as testing properly autonomous entities (agents) and integrating parts of a whole system that are developed by different programmers, have motivated the work published by Kampik, Amaral, and Hübner (2021).