Autonomous Agents and Multi-Agent Systems (2023) 37:41
https://doi.org/10.1007/s10458-023-09623-8

UPDATE

Generating and choosing organisations for multi-agent systems

Cleber J. Amaral[1,2] · **Jomi F. Hübner**[2] · **Stephen Cranefield**[3]

Accepted: 5 September 2023 / Published online: 24 September 2023

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 paper 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 priori defined inputs. These characteristics facilitate the parametrisation, the use for resource planning and the chance of finding feasible solutions. This paper also introduces two model extensions, which define processes and constraints that illustrate how GoOrg suits different domains. Among aspects that surround an organisation design, this paper discusses models' input, agents' abstractions and resource planning.

Keywords Organisation design · Organisational structure · Automated design

1 Introduction

An agent is a software entity that has particular beliefs, goals and capabilities, and exhibits some autonomy in the sense that it determines how it will achieve its goals [1, 2]. Usually, an agent is not alone in the environment it is situated, i.e., it is part of a

✉ Cleber J. Amaral
cleber.amaral@ifsc.edu.br

Jomi F. Hübner
jomi.hubner@ufsc.br

Stephen Cranefield
stephen.cranefield@otago.ac.nz

[1] Área de Telecomunicações, Instituto Federal de Santa Catarina, R. José Lino Kretzer, São José 88103-902, SC, Brazil

[2] Automação e Sistemas, Universidade Federal de Santa Catarina, R. Agronômico Ferreira, Florianópolis 88040-900, SC, Brazil

[3] Information Science, University of Otago, Clyde Street, Dunedin 9016, Otago, New Zealand

Multi-Agent System (MAS). The autonomy of agents in a MAS requires some mechanism to organise and coordinate them to achieve the system's goals [3]. Organisations are used to address this issue by promoting a coherent mechanism, which constrains and enforces acceptable behaviour of agents [3–5]. Notice that organisations are independent of the agents, thus organisations and agents are usually designed using independent models and tools [6]. Additionally, the design of organisations is arguably as important as the design of the agents [7, 8].

The design of organisations is a laborious task, especially when there are many goals, constraints and agents [9]. In the last decades, researchers have created many organisational design models [5, 10–13]. Due to the specificity of each case and the high number of variables that surround the design process, no existing model can be considered a decisive solution [13]. Indeed, it is challenging to define an effective, flexible and easy to parametrise design model. This paper presents GoOrg, an MAS organisational design model for automatically 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. The set of goals are used to synthesise organisational positions. Given a set of available agents, the organisational structures' feasibility can be checked.

Existing models for generating organisational structures for MAS [5, 10–13] use the concept of roles, considering that agents and roles have many-to-many relationships. Although intuitive, the concept of roles does not ease planning resource use and checking organisation feasibility. GoOrg uses organisational positions instead, which have one-to-one relationships with agents [14], thus an organisational structure made up of positions reflects resource demands. This allows the user to know in advance (at design time) the required resources for an organisation to run. Above all, it is relevant when the agents are physical devices that have to be available and usually must be allocated to a unique operation at once. In an organisation based on GoOrg, this robot occupies a particular position synthesised by the model.

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. In contrast to related works, in GoOrg, organisational positions are synthesised. With synthesised positions, a broader range of possible solutions can be generated, increasing the possibility of finding structures fillable by the available agents. Besides, a generator that automatically synthesises positions from goals relieves the user from the task of defining roles.

This paper also introduces *GoOrg4Prod* and *GoOrg4DSN*, extensions of GoOrg for two case studies. They illustrate how GoOrg can be customised for different purposes by selecting particular constraints and defining particular processes.

The structure of this paper is as follows: Sect. 2 presents a background of the automated organisation design research area and the state of the art of *automated organisational structure generators*, a class of generators in which GoOrg is included; Sect. 3 describes the GoOrg generic model; Sect. 4 presents *GoOrg4Prod*, an extension of GoOrg for a factory domain; Sect. 5 presents *GoOrg4DSN*, an extension of GoOrg for the Distributed Sensor Networks (DSN) domain; Sect. 6 discusses organisation design aspects comparing GoOrg to other approaches; and Sect. 7 presents conclusions and suggestions for future work.

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 [15]. 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. Our research is focused on the ability to automatically conceive specifications of organisational structures. The generation of organisational structures is mainly performed at design time. Still, there are works not covered in this research, that instantiate specifications and orchestrate the MAS at run time.

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 1 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 (1999) [16], the *Star Model* proposed by Galbraith (1995) [9], the step-by-step framework proposed by Burton (2011) [17] and the design process introduced by De Pinho (2006) [18].

In the design of organisations of software, there are many studies of *Service Composition*, which can orchestrate applications by defining sequences of commands and managing communication among Web services [19]. *Service Composition* has no concern about the degree of autonomy of the software artefacts it is orchestrating. Also, there are many studies of organisations formed by computational agents, i.e., autonomous entities, 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 [19–21].

Considering just models for designing organisations for MAS, some examples of *non-automated organisation generators* are STEAM [22], AALADIN [23] and $\mathscr{M}oise^{+}$ [24]. These models allow explicit organisational design in a variety of structures considering aspects such as norms, roles, relationships, organisational goals,

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

[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.

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 section 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: Sect. 2.1 presents *automated organisational design by task planning*; Sect. 2.2 introduces *self-organisation* approaches; and Sect. 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 [25, 26].

2.1 Automated organisational design by task planning

Automated organisational design by task planning is the first class to be introduced. Generators in this class 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 [27]. 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) [28] 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 [29] has proposed a domain-independent model called Decentralised Online 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. 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.

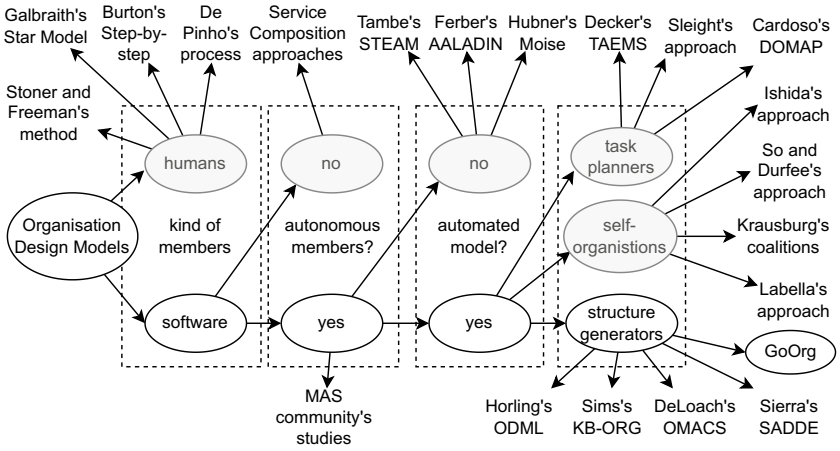


Fig. 1 Design model categories

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 [30]. 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 [31]. 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 [12]. Self-organisation approaches are more concerned about coordination policies and less concerned about the generated organisational structure itself [32].

In one of the earlier studies, Ishida et al. [33] has presented an adaptive self-design approach which creates and destroys agents to allocate tasks according to resources and environmental changes and needs. Decker et al. [34] proposed an approach for adaptation of MAS on the 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 [13] characterises 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 [36] extended the study performed by Decker et al. [34], adding a task representation framework, enabling this new method to become domain-independent and to reason about quantitative aspects of tasks. There are some studies on self-organised swarms based on computationally limited agents, which

[3] The cloning mechanism is used by agents when predicting or perceiving overload [35].

often do not even know about the presence of other agents and that are coordinated by simple mechanisms [31, 37].

More recently, Kota et al. [38] presented 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. [39] joined cloning and spawning functions in their self-design method. In this approach, 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. [40], Rahwan et al. [2], Krausburg et al. [41] 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 little relationship definitions fostering to handle overlapping tasks and have information flowing in many directions.

2.3 Automated organisational structure generators

Finally, the third class is that of *automated organisational structure generators*. It is concentrated on defining goals, examining the organisational environment, allocating resources, and generating organisational structures [25]. 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. Studies on task planning have attracted interest for a while, and in recent years, studies on self-organisations have gained even more attention. Nevertheless, the *automated organisational structure generators* are the only ones that carefully design organisations as first-class abstractions [12]. 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 in 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. [42, 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.

As stated by McAuley et al. [43, 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. [42, 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 [44], 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 [45].

Notably, the definition given by Katz and Kahn [44] 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" [23, p. 130], and roles "can be seen as *place-holders* for agents and represent the behaviour expected from agents by the society design" [46, 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 [13]. 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 minimal resources, 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 [47, 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 [48]. It refers to an administrative instrument resulting from identification, analysis, ordering and grouping of activities and resources of companies, including decision processes to achieve the expected goals [30]. The structure is intrinsically linked to many other organisational aspects. Besides, as pointed out by Durfee et al. [49, 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 [50]. The inseparability of organisation and structure concepts is observed in different studies in which correlated categorisations are presented [16, 25, 51, 52]. 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 [53, 54], goals provide the first pillar of an organisational design, representing the organisation's strategy. A goal is a desired state of the world [4], and thus can be used to define the system's overall behaviour [55].

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 [25]. Organisations are diverse in kind and form according to their purposes and environments. Arguably, it is practically impossible to address particularities of all sorts of organisational

purposes and environments. For this reason, domain-independent models in this class allow the user to adapt 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) [5]. 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 [10] proposed another approach called Organization Model for Adaptive Computational Systems (OMACS) (see also [56, 57]). 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. 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 roles defined a priori. 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. [12] 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 main improvement is a reduction in computing effort due to the segmentation of application-level and coordination-level functions in the planning process. 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 [11] introduced the 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 with 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 *Do not 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 less obvious roles, 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 Sects 6.1 and 6.4).

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.

Table 1 Comparison among automated organisational structure generators

Organisation generator	Goals are inputs	Do not 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
GoOrg	Y	Y	Y	Y	Y	Y	Y*	F	Y*	Y	Y
SADDE	Y*	–	Y	Y	Y	–	–	-	–	Y	–
OMACS	Y	–	Y	Y	Y	–	–	–	–	Y	–
KB-ORG	Y	–	Y	Y	Y	–	Y	–	–	Y	–
ODML	Y*	–	Y	Y	Y	–	–	–	–	Y	–

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

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. If all the roles/positions can be filled, an organisation is considered feasible. 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. Although it is resembling the idea of an organisational position, these models are unable to predict the number of agents needed to fill the structure (see Sect. 6.2, 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.

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; and (v) they generate feasible organisations. 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 produces multiple hierarchies such as organisations or departments of an organisation; and (d) GoOrg is the only model that uses the concept of positions, which facilitates resources planning.

Finally, it is essential to point out that there is no single type of organisation suitable for all situations [58]. It is also true that there is no individual approach ideal for creating all organisations [48]. 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 [59, 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.

GoOrg defines elements and relationships between them. Figure 2 presents an overview of GoOrg.[4] The shaded shapes represent the elements of the model. The hollow shapes represent attributes of an organisational structure. The solid lines indicate relationships with their cardinalities, in which filled diamonds indicate compositions, hollow diamonds indicate aggregations, hollow arrowheads indicate inheritances and no arrowhead indicate associations.

In the following, the model is presented in detail. Section 3.1 describes GoOrg from the perspective of the model's elements; Sect. 3.2 describes GoOrg from the perspective of organisational structure attributes. and, Sect. 3.3 highlights GoOrg characteristics and the differences between the generic model and its extensions.

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. An example of a goal is to move a box from point A to point B or to track an object that is being perceived by sensors. Formally, GoOrg defines a goal as follows.

Definition 1 (goal) A goal g is represented as a symbol, and the set of all goals is denoted by G.[5]

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

An agent is an entity that acts to achieve the goals it is committed to. An example of an agent is a robot that moves boxes across the floor 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. The organisation's goals are distributed across its organisational positions. If a position is not assigned to any organisation's goal, it is not explicitly considered to be a part of the structure. The agent that occupies a position is in charge of achieving 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

[4] The presented diagram follows the UML (Unified Modeling Language) class diagram convention for representing relationships between elements.

[5] In this work, a symbol is a single constant term, e.g., "MoveBox" and "manage_sector_NW".

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.

$$
\begin{aligned}
&p : symbol, p \in P \\
&gp : P \rightarrow 2^G \\
&\forall p \in P, gp(p) \neq \{\} \\
&ap : P \rightarrow A \cup \{\epsilon\} \\
&\forall p, p' \in P, (p \neq p') \wedge (ap(p) \neq \epsilon) \wedge (ap(p') \neq \epsilon) \Rightarrow (ap(p) \neq ap(p'))
\end{aligned}
$$

To check if an agent can occupy a position, GoOrg 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 requirement to fulfil the goal. Similarly, the agent *calculator* may have the feature *solve factorials* representing something the agent is able to do. In this case, the agent *calculator* is able to fulfil the goal *solve combinatorics* since it has the the requirement *solve factorials*. 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.

$$
\begin{aligned}
&f : \langle symbol, e_2, \ldots, e_n \rangle, f \in F \\
&fg : G \rightarrow 2^F \\
&fa : A \rightarrow 2^F
\end{aligned}
$$

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. If every organisational position has an agent to occupy it, the organisation is considered feasible.

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. 1). The set B contains

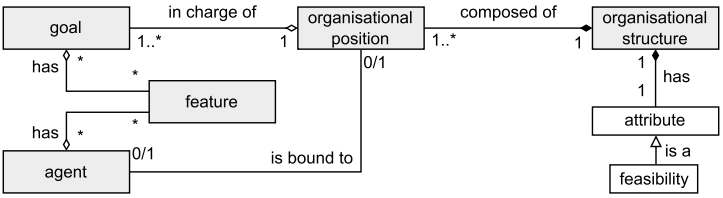


Fig. 2 GoOrg model

the agents that are bound to positions in P (Eq. 2). The organisation is entirely feasible ($\kappa(o) = 1$) when every position is bound to an agent.

$$\kappa(o) = \frac{|B|}{|P|} \tag{1}$$

$$B = \{ap(p) | ap(p) \neq \epsilon, p \in P\} \tag{2}$$

3.3 Highlighted characteristics of GoOrg

GoOrg is a generic model for designing organisations which considers that an organisation is represented by its structure of positions. The organisational goals are assigned to positions. Each position should be occupied by an agent. Based on the specifics of each domain, the goals may have certain features that help to assign goals to positions and to bind agents to them. In the generic model, features are defined in the generic sense, i.e., it has no specified meaning. The generic model includes goals, agents, features, positions and structure given that these elements have been recognised as fundamental to all domains.

For 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.

It is worth mentioning that the GoOrg 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 must be made.

As GoOrg must be extended to address the requirements of each domain, this paper presents two extensions: *GoOrg4Prod*, an extension of GoOrg for a factory domain; and *GoOrg4DSN*, an extension of GoOrg for the DSN domain. Table 2 describes the generic model elements that are present on its extensions, some of which are shared by the domains shown and others of which are exclusive to a particular domain. As a generic model, GoOrg does not specify any kind of relationship between positions. Indeed, even a common kind of relationship such as "is superior of" is not considered generic as it is not present in all kinds of structures (such as federations and holons [58]).

In both studied domains, a feature for quantifying the effort required to achieve a goal was defined (called *workload*). However, it is shown that the definition differs by

domain, which makes it not generic. In addition, we recognise that there are alternative approaches to quantifying the usage of a resource (such as the amount of sent data, energy consumption, etc.), depending on the specifics of the domain. In Table 2, the remaining defined features (skills, sectors, and indents) are already exclusive to a single domain, i.e., they are truthfully not generic. The next sections give more details about the elements and relationships that were defined for each domain.

GoOrg also does not specify a method for designing organisations. Although the presented extensions (*GoOrg4Prod* and *GoOrg4DSN*) use the same search algorithm for generating organisations, it can be seen that the processes slightly differ. Besides, we acknowledge that other techniques can be applied.

4 GoOrg4Prod: an extension for a factory production line domain

GoOrg4Prod illustrates how GoOrg can be used in a particular domain.[6] *GoOrg4Prod* generates structures of positions responsible for production activities in a factory.[7] In a particular production process, the goals can be achieved by many organisation structures, but they would perform differently in some characteristics. For instance, one may consider that it is better to have interchangeable agents to provide more robustness, or it is better to have a unique agent leading the others to get reports from a central unit, and so on. It is essential to know in advance which kinds of agents are necessary to execute the specified production process, notably when the agents are physical entities.

GoOrg4Prod synthesises positions generating organisations in which goals should be achieved routinely by executing some *workloads*.[8] To be able to execute *workloads*, agents should have some *skills*. *GoOrg4Prod* matches agents and positions using *skills*. *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. It is assumed that an external mechanism will pick the organisational structure indicated by *GoOrg4Prod* to use it in the run time, orchestrating the agents according to the organisation's definition.

The generation of hierarchical structures is considered in this domain. It is assumed that the hierarchical levels of each organisational member are relevant for the MAS at run time. In fact, a structure generated by *GoOrg4Prod* tells the places in the hierarchy and assigned goals of the synthesised positions. The use of the hierarchy defined by *GoOrg4Prod* is up to the running MAS (coded in the agents and/or an orchestrating mechanism—hich is out of our scope). In a running MAS, for instance, agents must report the goals they have accomplished to the agent occupying the top superior position. Roughly, hierarchies can be expressed by organisational charts.[9] As it is generating hierarchies, the structures present the attributes *height* and *generality*, as later explained.

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

[7] In this work, the term *GoOrg4Prod* refers to both a model's extension and an implementation that can generate organisation descriptions for a factory domain.

[8] Notice that this particular approach is using a baseline of 24 h (a day in a factory).

[9] There are criticisms arguing that organisational charts miss crucial aspects of organisational structures [60]. Although charts are limited, they are suitable for the purposes of this work which focus on representing superordinate-subordinate relationships.

Table 2 Generic and specific elements of the models

Item/Model	GoOrg	GoOrg4Prod	GoOrg4DSN
Goal element	✓	✓	✓
Agent element	✓	✓	✓
Structure element	✓	✓	✓
Position element	✓	✓	✓
Feature element	✓	✓	✓
"Is superior of" Relationship		✓	✓
Workload Feature		✓	✓
Skill Feature		✓	
Sector Feature			✓
Intent Feature			✓

A maximum capacity of handling *workloads* can be set on *GoOrg4Prod* to avoid exceeding agents' capacities. Figure 3 highlights 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 in which 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). In this example, it is necessary to consult a database for getting orders. Figure 4 illustrates how this scenario is modelled.

4.1 GoOrg4Prod elements

GoOrg4Prod extends GoOrg elements by specifying two features: *workloads* and *skills*. These features constrain the organisation design while synthesising positions, arranging hierarchies and binding agents to positions.

From the agents' perspective, it is considered that agents have *skills*. A *skill* s is defined as a singleton tuple which belongs to the set S of all *skills*, as follows.

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

For achieving goals, it may be required the execution of *workloads*. A *workload* w represents a demanded effort $e \in \mathbb{R}^+$ which requires a *skill* $s \in S$ to be performed.[10] In the example of Fig. 4, there are four *workloads*: (i) the *skill db access* with a predicted effort of 0.1 h; (ii) the *skill lift* with a predicted effort of 4 h; (iii) the *skill move* with effort equals to 8 h; and (iv) the *skill pnp* with a predicted effort of 1 h. The function *wg* maps goals to their *workloads*, as follows.

[10] The model assumes that there is no difference in agents' performance and all the given agents are fully available to be used by the organisation.

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

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

$$F = W \cup S \tag{3}$$

As a design choice, *GoOrg4Prod* considers that organisational positions may have superordinate-subordinate relationships, which are represented as "is superior of" relationships. Such a relationship can be used by a running orchestrating mechanism, for instance, to define which position is in charge of delegating small grain sized tasks or solving conflicts. 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 is 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 by 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. It is also possible to have a forest with only one tree. 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*. In response to this practical issue, in *GoOrg4Prod*, $\phi_p \in \mathbb{R}^+$ is defined to represent the maximum *workload* allowed on each position. To allow splitting goals into smaller ones when necessary, $\phi_g \in \mathbb{R}^+$ is defined to refer to the maximum grain size for *workloads*.

4.2 GoOrg4Prod added attributes

GoOrg4Prod defines new attributes of an organisational structure. Based on superordinate-subordinate relationships, the structure *height* is calculated. 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 5 illustrates the organisation attributes space.

Height refers to how centralised and bureaucratic an 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 on the organisation workflow.

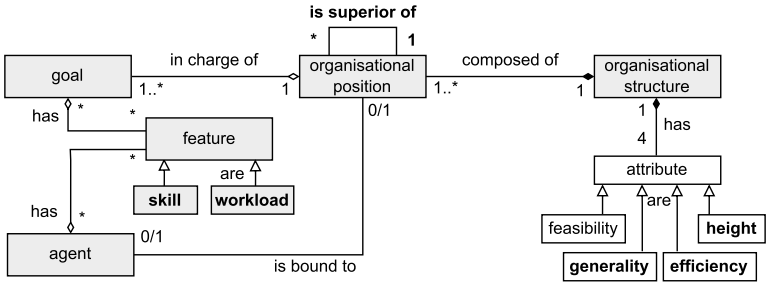


Fig. 3 GoOrg4Prod model

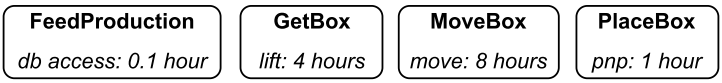


Fig. 4 A given set of goals with associated *workloads*

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 failing. 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]. Roughly, $\tau(o)$ is the ratio between the actual height and the maximum height that the model generates (Eq. 5). The function $l(p)$ maps a position p to an integer representing the hierarchical level that the position p is situated at (Eq. 4). 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.[11]

$$l(p) = \begin{cases} 0 & sp(p) = \epsilon \\ 1 + l(sp(p)) & \text{otherwise} \end{cases} \tag{4}$$

$$\tau(o) = \begin{cases} \frac{\max_{p \in P}(l(p)) - 1}{|G| - 1} & |G| \geq 2 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The generality of an organisational structure measures the similarity of positions considering their assigned goals. The most generalist organisation has all goals assigned to

[11] $|G|$ should be equal to or greater than 2 to generate different and comparable candidates.

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.

Formally, the generality of an organisational structure o is represented as $\theta(o)$, a real number in the range [0,1]. Roughly, it is the ratio between the actual and the maximum possible number of goals assigned to positions (Eq. 6). The set GP contains the recorded goals for all positions (Eq. 7), 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 $|G||P|$. 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.

$$\theta(o) = \frac{|GP| - min(|G|, |P|)}{|G||P| - min(|G|, |P|)} \tag{6}$$

$$GP = \bigcup_{p \in P} gp(p) \tag{7}$$

In *GoOrg4Prod* the efficiency of a structure o is represented as $\eta(o)$ a real number in the range [0,1]. In this context, it is given by the ratio between *utilisation* and *capacity* (Eq. 8). The *utilisation* is given by the sum of *workload*'s efforts (in hours) associated with all given goals.[12] The organisation's *capacity* is the number of positions of the organisational structure times ϕ_p (the maximum *workload*'s efforts (in hours) allowed per position).

$$\eta(o) = \frac{\sum_{g \in G} \sum_{w \in wg(g)} \pi_2(w)}{|P|\phi_p} \tag{8}$$

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 subprocesses: (i) the given goals in G are split down into smaller goals according to the granularities (ϕ_p and ϕ_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. Figure 6 illustrates the referred subprocesses.[13]

[12] $\pi_i(a)$ refers to the ith element of the tuple a.

[13] Although this work is suggesting a method for organisation generation, it is not claiming it is the only possible one.

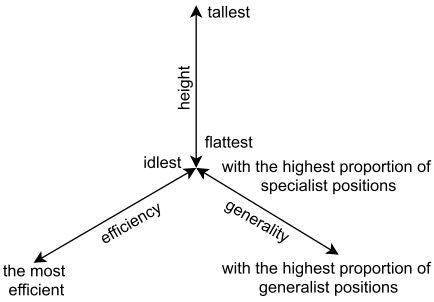


Fig. 5 Organisational structure attributes in three dimensions

4.3.1 Preparing goals for assignments

GoOrg4Prod splits the given goals into smaller ones according to a given *workload* grain ϕ_g. In this sense, the goals of G that are set with more effort than the respective grain size should split into smaller ones, creating the set G'. Splitting goals allows assigning the same goal (with less effort) to multiple positions, which may increase the generality of the final organisation since it potentially creates interchangeable positions. This extension assumes that achieving all parts implies the achievement of the original goal.

To exemplify, it is considered the set G illustrated in Fig. 4. This set has four goals (**FeedProduction**, **GetBox**, **MoveBox** and **PlaceBox**) which are associated with *workloads*. Considering that the grain ϕ_g is set as 4 hours, 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. The resulting set of goals that suit the given granularities are illustrated in Fig. 7.

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 a position.[14] 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 goal in two stages. In the first stage, a new structure is generated assigning one goal $g \in G$ to a new top superordinate position. In the second stage, each remaining $g' \in (G \setminus \{g\})$ 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; (ii) assigned to new positions that are created to be subordinate of every $p \in P$; and (iii) assigned to every existing $p \in P$ (no position is created). These transformations are detailed as follows.

[14] To simplify, it is being considered that $G = G'$, i.e., G is the set of goals after some of them were split.

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. The set S' records the skills that are associated to workloads, which stands in for the skills an agent needs in order to occupy this new position. This transformation from the structure o to the structure o' is formalised as follows.[15]

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

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.

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

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'). This transformation is formalised as follows.

$$joinAPosition(g, p)\frac{o = \langle P, G, F, A, gp, fg, fa, ap, sp, wg\rangle}{\begin{array}{r} S' = \{\pi_1(w)\,|\,w \in wg(g)\} \\ o' = \langle P, G, F, A, (gp \setminus \{p \mapsto gp(p)\}) \cup \{p \mapsto (gp(p) \cup \{g\})\}, \\ ((fg \setminus \{g \mapsto fg(g)\}) \cup \{g \mapsto (fg(g) \cup S')\}), fa, ap, sp, wg\rangle \end{array}}$$

[15] Barred arrow notation for elements are 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.

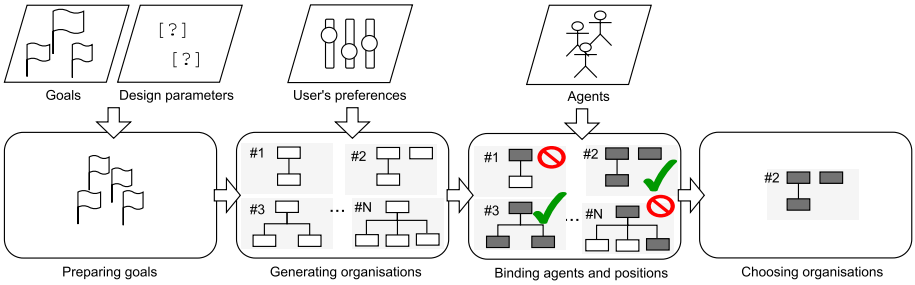


Fig. 6 The four subprocesses of *GoOrg4Prod*

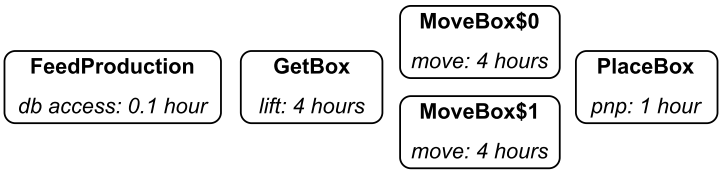


Fig. 7 The set G' of split goals for $\phi_g = 4$

These transformations are applied by the search algorithm on every state exploration. The Algorithm 1 specifies the successor states of the search. The set G_{na} has all non-assigned goals and 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.[16]

[16] To generate successors of the created search states, the algorithm use G'_{na} such that $G'_{na} = G_{na} \setminus \{g\}$.

Algorithm 1 Successors - creates new states to explore

Require: G_{na} a list of not assigned goals, P the current set of positions
Ensure: U a list of successor states

```
U ⇐ ∅        // The successors list
if P = ∅ then
    for Goal g of G_na do
        U.add(addSuperiorPosition(g))        // Add as a superordinate
    end for
else        // There is at least one existing position
    for Goal g of G_na do
        U.add(addSuperiorPosition(g))        // Add as a superordinate
        for Position p of P do
            U.add(addASubordinate(g, p))        // Add as a subordinate of p
            U.add(joinAPosition(g, p))        // Assign g to existing p
        end for
    end for
end if
```

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.

4.3.3 Binding agents and positions

GoOrg4Prod binds agents and positions matching their features. It allows the verification of the organisation feasibility considering the available agents. As presented in Eq. (3), the set of features F is formed by the sets of *workloads* and *skills* (both workloads and agents' *skills*).

To bind agents to positions, *GoOrg4Prod* uses the First Fit algorithm. 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).

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).[17]

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

The user may define multiple criteria which follow a priority order.[18] 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 subprocess 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 $\overset{>}{}_\rho$, in which $o \overset{>}{}_\rho o'$ means that o is preferred to o'. If two criteria were set ($\Gamma = \{1, 2\}$), $o \overset{>}{}_\rho o'$ is defined as: $o \overset{>}{}_\rho o' \iff [c_1(o) > c_1(o') \vee (c_1(o) = c_1(o') \wedge c_2(o) > c_2(o'))]$ which can be generalised as stated by the following formula.[19]

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

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. (9) gives the worst estimation of the number of states visited by *GoOrg4Prod* search algorithm.[20]

$$1 + n + 2^{(n-1)} n! n = O(2^n n!) \tag{9}$$

As presented in Algorithm 1, 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 Fig. 7, in which $|G| = 5$. According to Eq. (9), the algorithm may explore 9,606 states as the worst estimation for searching for all possible solutions of this case.

[18] The feasibility attribute ($\kappa(o)$) is not being used to order but to exclude non-feasible organisations.

[19] In the case of two or more organisations are in the same level, considering all criteria, any of these organisations is chosen.

[20] Eq. (9) does not take into account states that are pruned for being similar to existing ones (with the same goals assigned to similar structures).

4.4 GoOrg4Prod results

To illustrate how the best candidate is chosen, the set of goals represented in Fig. 7 is considered. It is assumed that the user prefers the most *generalist*, *efficient* and *flatter* structure, in this priority order. Figure 8 illustrates the three kind 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.

For the given example, the generation subprocess has produced 1,646 organisational structure candidates.[21] Every organisational structure candidate has all the given goals assigned to positions. These candidates are represented in Fig. 9. Each circle represents one or more candidates, since they can be overlapped.

In Fig. 9, the *height* axis represents the τ attribute (Eq. 5) as a percentage given by 100τ, and the *generality* axis represents the θ attribute (Eq. 6) as a percentage given by 100θ. The sizes of the circles represent the efficiency η (Eq. 8) as a percentage given by 100η. The biggest circle represents 54.6% of efficiency and the smallest ones represent 10.9% of efficiency. 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 as feasibility is considered volatile and that 100% feasibility is a requisite (not a preference), it does not affect the order for generating candidates.

As depicted in Fig. 10a, the candidate #1 presents only one organisational position, which would be occupied by one agent that is responsible for achieving all goals. This 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 h) over the baseline (24 h). 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**.

Figure 10b depicts the candidate #2, the second best solution according to the user's preferences. This candidate has also the lowest height, since the two generated positions are in the same hierarchy level. However, 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.

Figure 11 depicts two feasible candidates for the given example. The candidate #134 is one of the flattest structures, since it presents just one hierarchy level. However, its *generality* is even more reduced comparing 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.

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

Figure 11 gives another example of a feasible solution: the candidate #446. The *generality*, *efficiency* and *feasibility* attributes are same as for the 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 arrows), which makes this solution less flat.[22]

The candidate #1646 (Fig. 12) is on 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 case of having only one unit of each kind of agent, this candidate is just 60% feasible.

As demonstrated, the three transformations of *GoOrg4Prod* can produce structures with superordinate-subordinate relationships as found in classic organisational charts.[23] A superior (superordinate) is often responsible for some kind of coordination of its subordinates. A superordinate-subordinate relationship may imply, for instance, power and accountability of a position to another [50]. 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: an extension for the distributed sensors network domain

GoOrg4DSN is another extension of GoOrg.[24] It addresses the problem of Distributed Sensor Networks (DSN), introduced by Lesser et al. [61]. *GoOrg4DSN* is concerned with the generation of organisational structures for tracking one or more moving targets in an area.[25] 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 provide the coordinates of targets as sensors signals are triangulated. The resolution track depends on how the sensors used on triangulations are distributed along the area and their distance of 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.

[22] This candidate could be the first choice if the user's preferences were set for the tallest and most general and efficient solution.

[23] Hierarchies are one of the existing MAS organisational paradigms [58].

[24] In this paper, the term *GoOrg4DSN* refers to both a model's extension and an implementation that can generate organisation descriptions for the DSN domain.

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

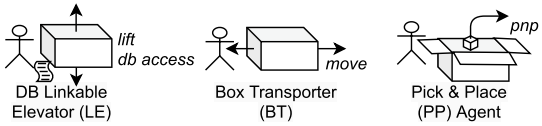


Fig. 8 The available agents

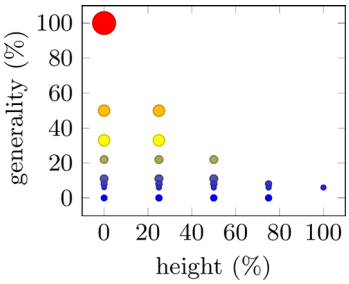
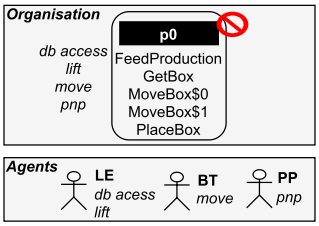
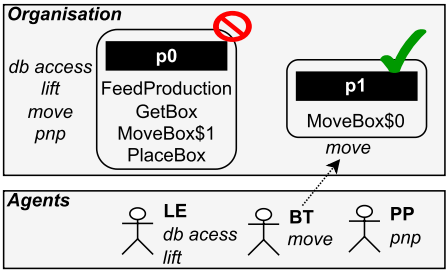


Fig. 9 The generated candidates quantified according to the user's preferences



(a) Candidate #1.

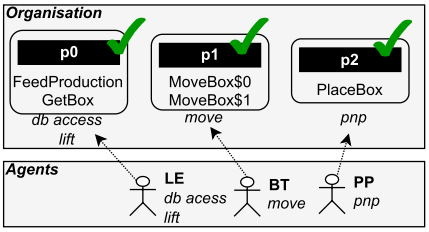


(b) Candidate #2.

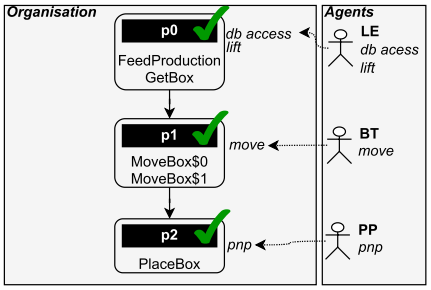
Fig. 10 The two best structures, which are not 100% feasible

GoOrg4DSN is based on the approach proposed by Horling and Lesser [11]. 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, communicates with other *Sector Managers* and elect a sensor to enact 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's implementation, the geographical area of each sector is arbitrarily defined according to the quantity 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 [7]. In MAOP, autonomous entities are modelled as agents, non-autonomous entities such as environmental tools and resources exploited by the agents are modelled as artefacts, and coordination mechanisms are modelled as organisations. Adapting the solution



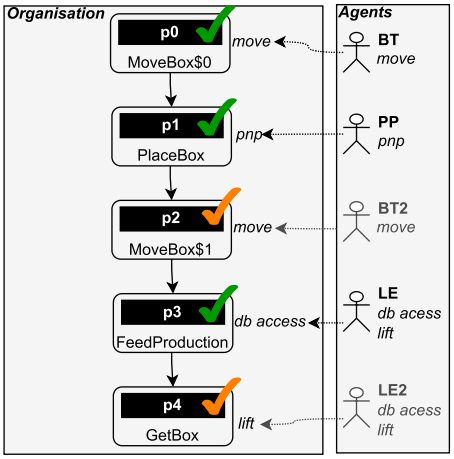
(a) Candidate #134.



(b) Candidate #446.

Fig. 11 Two feasible structures for the given example

Fig. 12 Candidate #1646



proposed by Horling and Lesser [11], *GoOrg4DSN* uses a new conceptual model for sensors. In *GoOrg4DSN*, the scanning task that referred author assigned to *Sensor Agents* are instead executed by non-autonomous entities (artefacts).[26] The main conceptual changes are the absence of explicit roles and that the procedural task of scanning the area is placed on another level of abstraction. This MAOP approach is represented in Fig 13.

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, i.e., changes its strategy. 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

[26] In Horling and Lesser'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.

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 bottom dashed rectangle of Fig. 13 represents a sector that contains five geographically distributed sensors. The central sensor with a shaded inner circle represents the device that was a priori defined to host the agent to manage the sector. Figure 13 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,[27]

Figure 14 illustrates how it 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.[28] To represent this, 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 to identify 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.[29] 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 organisation 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 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.

[27] The network has distributed processing along sensors. In *GoOrg4DSN* each sensor hosts an artefact process and also an agent process, which is often a stand-by agent. Indeed, most of the sensors host an available agent which can be bound to an organisational position, becoming an organisation's member.

[28] According to signal ranges, sector managers are usually the agents that can better reach sector sensors, and also reach other sector managers.

[29] 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*.

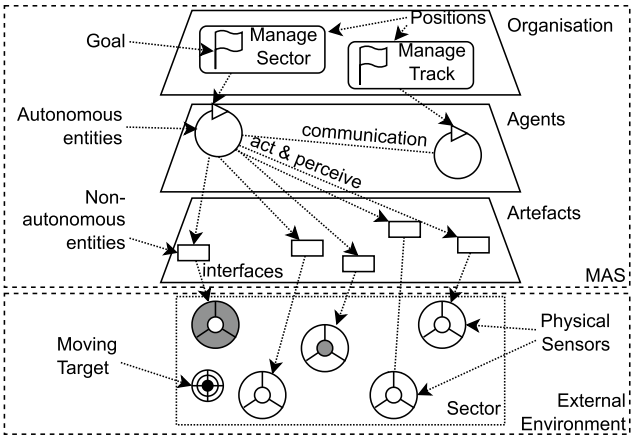


Fig. 13 An MAOP approach for the DSN domain

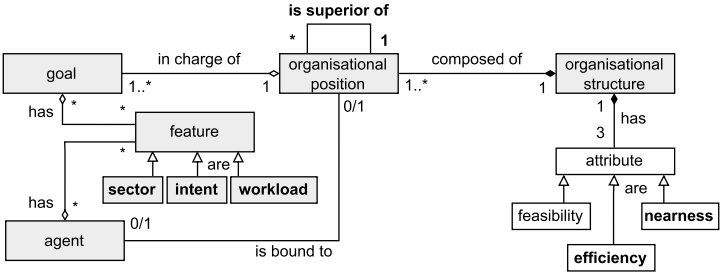


Fig. 14 GoOrg4DSN model

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

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.

$$F = W \cup T \cup C \tag{10}$$

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* (Sect. 4.1).

GoOrg4DSN also adds a design parameter to prevent positions 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. (8). 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.[30] Formally, the *nearness* of an organisational structure o is represented as $\rho(o)$, a real number in the range [0,1] (Eq. 11). The nearness reduces when a position has a different *sector* comparing 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*.

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

$$\rho(o) = \frac{\sum_{p \in P} n(p)}{|P|} \tag{11}$$

5.3 GoOrg4DSN processes

GoOrg4DSN generates and chooses organisations in a chain of three subprocesses[31]: (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. Figure 15 illustrates these subprocesses.

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 1, the difference is in the constraints of the transformation.

[30] 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.

[31] Comparing to *GoOrg4Prod,* the subprocess for preparing goals is not necessary in the DSN domain, since for this domain the goals are not divisible.

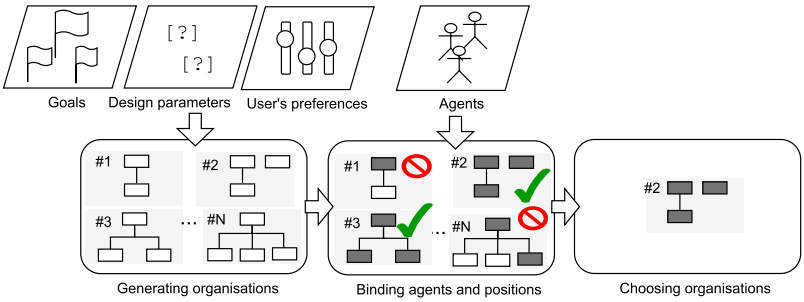


Fig. 15 The three subprocesses of *GoOrg4Prod*

GoOrg4DSN applies three structure transformations for every goal into two stages. In the first stage, a transformation assigns one goal $g \in G$ that are associated with the *workload manage_sector* to new top superordinate positions. In the second stage, each remaining $g' \in (G \setminus \{g\})$ are 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*.[32] 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.[33] This transformation from the structure o to the structure o' is formalised as follows.

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

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

[32] Since, 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.

[33] 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 subprocesses.

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.

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

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.

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

5.3.2 Binding agents and positions

To bind agents and positions, *GoOrg4DSN* uses the First Fit algorithm. 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 match by *intents* and the first element of *workload* tuples.[34]

[34] Notice that GoOrg4DSN implements a binding process just to check the organisation 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* (Sect. 4.3.4).

5.3.4 Computational complexity

The worst estimation of the number of search states visited is the same of *GoOrg4Prod* (Eq. 9). 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.[35]

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 (Fig. 16). Each sector contains five sensors.

The user's preferences are based on Horling and Lesser's [11] approach. Due to computational and communication limitation of the sensors that host agents, it is preferred *idle* structures (less *efficient*).[36] 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 being

[35] This took 1.0 s of user time on an Intel® Core™ i7-8550U CPU @ 1.80GHz with a 16 GB RAM computer. 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 by *GoOrg4DSN* To exemplify, for $|G| = 7$ *GoOrg4DSN*, took 1.1 s to generate 612 candidates and for $|G| = 8$ it took 1.7 s to generate 5812 candidates.

[36] In respect to computational limitation of the sensors, in 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. In the other hand, the preference for more idle structures using the attribute *efficiency* avoids both of the referred situations.

considered a situation in which no sensor is detecting any target. It is only necessary to manage each sector, as illustrated by Fig. 17.

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 Fig. 18. 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.

In another situation, it is considered that the sensors have detected a target on the south-east (*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 19 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 a tracking goal. However, it would create a less *idle* structure, which is not desired according to the user's preferences.

For the scenario with one target being detected on the sector *se*, *GoOrg4DSN* has generated 8 candidates. The best candidate is illustrated by Fig. 20. 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. In this candidate, a new position was created as 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 to assign this goal to the managers of other sectors.

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 Fig. 21. 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 less preferred according to the user's preferences, comparing to the 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 on agents' availability may also

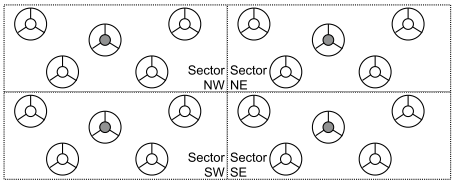


Fig. 16 A motivating scenario with four sectors, each one with 5 sensors

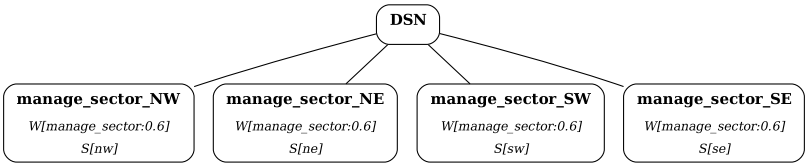


Fig. 17 The set of goals in which no target is being detected

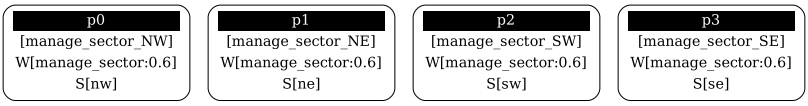


Fig. 18 Candidate #1 (unique) when there is no target being detected

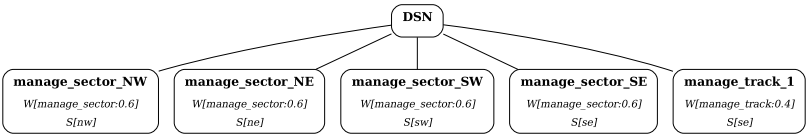


Fig. 19 The set of goals in which one target is being detected

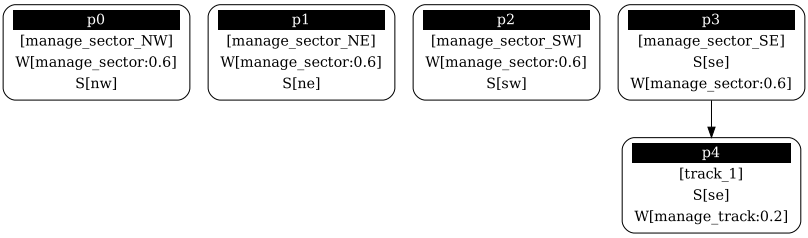


Fig. 20 Candidate #1 for the scenario with 1 target being detected

drives to a structure-switching, for instance, if there are 4 targets being detected in a sector and one of the agents fails. A new binding between agents and positions would revise the idlest structure as not feasible, making other candidates more suitable in this condition.

6 Discussion

This section presents a discussion about aspects related to the organisation design, comparing GoOrg to other approaches: (i) assigning goals to named agents, to roles or to positions; (ii) planning resources (agents) of organisations; (iii) the difference between synthesising positions and having a priori user-defined roles; and (iv) the difference of using goals as input instead of roles and behaviours.

6.1 Assigning goals to named agents and to impersonal representations

Task allocating models [27–29] can generate organisations. These models assign goals (or tasks) to named agents, as illustrated in Fig. 22a. 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. These organisations are fixed and *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 new design.

In the particular class of generators in which GoOrg is included, goals are somehow assigned to impersonal representations of agents, such as roles and positions. The studies DeLoach and Matson [10], Horling and Lesser [11] and Sims et al. [12] 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 *assembler* role 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 22b 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.

GoOrg proposes the use of organisational positions. In this approach, goals are assigned to positions. A position is a place in the organisational structure that has 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 time. As illustrated in Fig. 22c, 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.

Using impersonal representations, the organisation is decoupled from the agents. Indeed, 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, the use of impersonal representations allows a generation of more appropriate solutions according to design criteria, since it has no concerns about specificities of agents.[37]

Additionally, when using impersonal representations, the design of the organisation (as an entity) can be apart of the process of binding agents and positions or roles. It simplifies the generating process since the binding part can be delegated to another process. When these processes are separated, the redesign can be a lighter procedure.

[37] A proper parametrisation of a generator can avoid the design of organisations that are not fillable by the available agents.

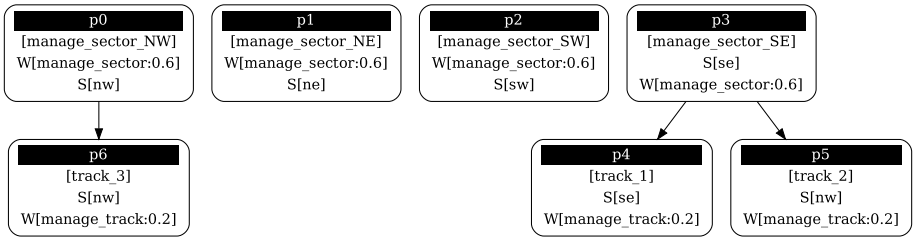


Fig. 21 Candidate #1 for the scenario in which three targets are being detected

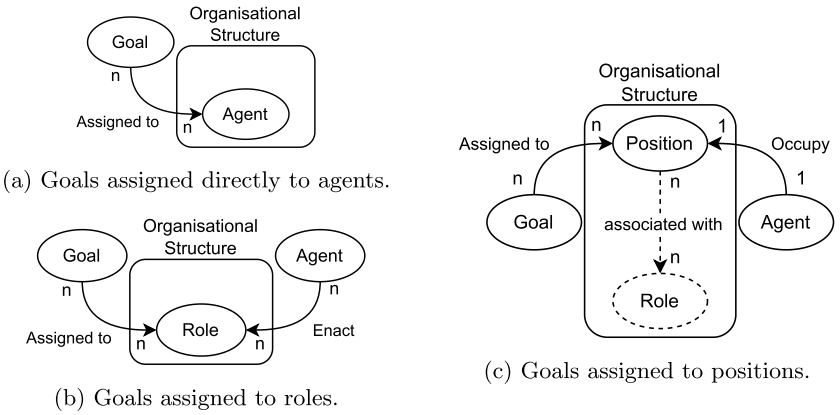
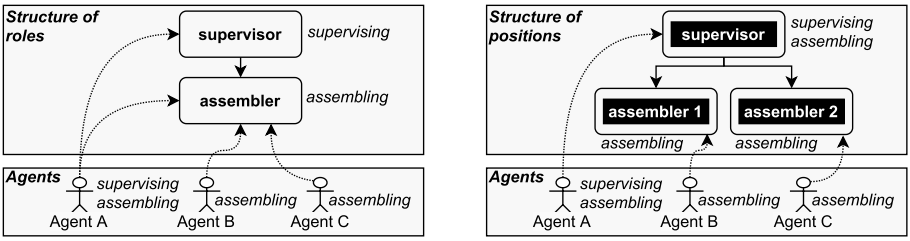


Fig. 22 Different forms of assigning goals to organisational members

The use of impersonal representations also prevents the need for redesigning processes, which is usually computationally heavy. To exemplify, let us compare the approaches represented in Fig. 22a against the approaches represented in Fig. 22b and c. 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. Using impersonal representations, the agents are associated with roles or positions. They make the organisation and the agents decoupled entities. In these cases, if an agent is not able to achieve the goals assigned to its role or position, no redesign is necessary, it is just needed to have another agent enacting that role or occupying that position. Thus, an organisation made up of roles or positions can be an open system, i.e., agents can join and leave the organisation at any time.

6.2 Planning resources of organisations

Comparing models based on impersonal representations for agents, roles provide more flexibility at run time than positions. 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 the 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, whether a combination of multiple roles satisfies the changing needs, no structure-switching is necessary, it is just needed to



(a) A structure of roles. (b) A structure of positions.

Fig. 23 Comparing a structure of roles and a structure of positions

change agent enactments. However, such flexibility makes the estimation of resources a tough task, especially when role enactments are very dynamic.[38]

In contrast, a structure made up of positions (one-to-one relationships) 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 run time dynamics but with the advantage of defining it in the design time, which facilitates resource needs estimation. Positions also provide some flexibility at run time, for instance, when in the absence of the agent that usually occupies a position, an available agent may come to occupy it.

Figure 23 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 23a 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 23b 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 represents resource needs, enabling its usage as a resource planning tool.

6.3 Synthesising positions instead of requiring user-defined roles

In other organisation generators [5, 10–12], 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" [5, 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

[38] Although the studies presented by Horling and Lesser [11] and Sierra et al. [5] do not focus on planning resources, it may be possible to estimate resources need using roles' behaviours, which are inputs of the corresponding models.

in 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 to available agents to play such a role.

An a priori user-defined roles approach presumably produces a smaller number of solutions compared to a model that synthesises positions.[39] 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 a priori user-defined roles approach may struggle when the system conditions do not match with the user's design assumptions.[40]

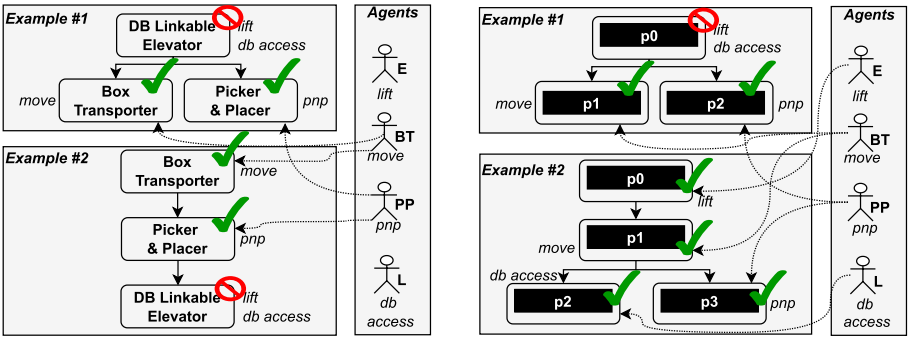
For the example illustrated by Fig. 8, one can define the following roles: *DB Linkable Elevator*, *Box Transporter* and *Picker & Placer*. Models based on a priori defined roles generate different structures for these given roles. Figure 24 presents examples that can be generated by each of these approaches.[41] In Fig. 24a there are two of the possible structures of a priori 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 regarding the exemplified condition. As a result, a design model based on those a priori defined roles cannot find a feasible solution without a user intervention redefining the roles and running the process again. Figure 24b, illustrates two solutions that *GoOrg4Prod* generates from synthesised positions. The example #1 is a solution presenting the same limitation of Fig. 24a examples, exemplifying that *GoOrg-4Prod* 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 Fig. 24b.

Besides bringing biases and limiting 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 comparing to a priori definitions. For instance, it was not defined a priori that a sector manager role/position should exist in the DSN situation illustrated in Fig. 17, but the solution presented in Fig. 18 has generated

[39] At run time, the number of combinations can be enlarged using many-to-many relationships between agents and roles. However, 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. Alternatively, the user may set roles with few requirements, increasing occurrences of agents enacting multiple roles, which can make the coordination and planning of the system more complex.

[40] Using the concept of roles and allowing many-to-many relationships between roles and agents, a method that defines a role for each given goal can achieve a higher number of combinations. However, besides complicating resource planning, such an approach makes the design less relevant since it delegates what is arguably a design task to another system (possibly the running system that decides on agent enactments). For instance, when assigning goals to a position, a method is already defining what the performer of the position should perform or not. In case of many-to-many relationships and roles with little job to do, it is complex for the design process to define that the performer of the goal $g0$ should not perform the goal $g1$ (which could be a goal to assemble something and another to check its quality, which would be better if they are performed by different agents). Besides, it would not allow a user to define preferences at design time, as GoOrg does.

[41] 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.



(a) Structures of a priori defined roles. (b) Structures of synthesised positions.

Fig. 24 Comparing examples of a priori defined roles and synthesised positions structures

(a) Roles and behaviours as inputs. (b) Goals and features as inputs.

Fig. 25 Comparing generators with behaviours and goals as input

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 Sect. 6.4.

6.4 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' expected behaviour. GoOrg is based on this assumption.

Instead of requiring goals explicitly, the organisation generators presented by Horling and Lesser [11] and Sierra et al. [5] require roles' behaviours expressed by equations. These definitions should be provided by the user (engineer) as input to the generator, and they are used to constrain the search for possible organisational structure descriptions. Whether the user definitions are correct, they can generate precise and coherent organisations for a MAS. In such a case, the search is supposed to be more constrained and faster when comparing to generators that tend to create wider search spaces such as GoOrg, DeLoach and Matson's [10] and Sims et al.'s [12] models. Although it is not explicit, one can say that generators which require roles' expected behaviours are also based on goal definitions. Indeed, roles are 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 25 illustrates sets of user definitions that should be provided as input to different generators. Figure 25a represents an organisation generator model such as Horling and Lesser's and Sierra et al.'s models in which goals are defined by the user (even if not

explicitly), then roles' behaviours are defined and given as inputs. Figure 25b represents an organisation generator model based on GoOrg such as *GoOrg4DSN* in which goals and features are user-defined and given as inputs.

In the approach of Fig. 25a, since roles' behaviour definitions depend 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. In this hypothetical less hierarchical approach, 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*. With such a change to the system goals, for a generator that requires roles' behaviours 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 Fig. 25 are also different. The input *goals* is 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 run time behaviour, which brings an extra concern when using models that require such inputs.

7 Conclusion

Organisation design has been refined continuously over the years. Many studies in the Administration Research Field propose theories and frameworks for this task. In the 2000 s, 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. GoOrg has great concern for simplifying its parametrisations, in which the automated position synthesising approach is highlighted.

Besides, GoOrg is extendible for dealing with the specificity and complexity of each domain. GoOrg has only the fundamental elements that are found in any organisational design. Domain-specific and even common elements, such as "is superior of" relationships, are not part of the generic model, which makes GoOrg more concise.

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.

Finally, this paper presented two extensions of GoOrg. These extensions 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 choose the "best" organisation. The proposed algorithms can solve simple problems in a satisfactory time. For both extensions, distinguishing processes were defined.

As future work, it is planned to: (i) test organisation run time adaptations in situations that require simple reallocations to complex redesigns; (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 [62]; (iii) implement more extensions of the model to test its applicability in other domains, to design other kinds of structures, and propose a methodology for selecting features and constraints for a domain; (iv) define a generic method for generating structures for different domains; and (v) test organisational aspects such as power, span of control, accountability, and trust.

Author Contributions Amaral wrote the main manuscript which is a result of a 4 years research supervised by both Hübner and Cranefield. Hübner and Cranefield actively proposed, discussed and revised the parts of this work.

Declarations

Competing Interests The authors declare that there are no competing interests that might be perceived to influence the results and/or discussion reported in this paper. This paper is the result of PhD research performed by Amaral and supervised by Hübner and Cranefield, who actively worked on the construction of this manuscript. The research was partially funded by Project AG-BR Petrobras and Programme PrInt CAPES-UFSC "Automação 4.0". The results discussed in this paper can be (re)generated following the instructions provided on the repositories https://github.com/cleberjamaral/GoOrg4Prod and https://github.com/cleberjamaral/GoOrg4DSN. Ethical approval is not applicable. The authors declare no competing interests.

References

1. Bordini, R. H., Hübner, J. F., & Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason* Series in agent technology (1st ed.). UK: Wiley.
2. Rahwan, T., Michalak, T. P., Wooldridge, M., & Jennings, N. R. (2015). Coalition structure generation: A survey. *Artificial Intelligence, 229*, 139–174. https://doi.org/10.1016/j.artint.2015.08.004
3. Hübner, J.F., Sichman, J.S., & Boissier, O. (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, pp. 118–128. https://doi.org/10.5555/645853.669463. Springer.
4. Boissier, O., Hübner, J. F., & Ricci, A. (2016). The JaCaMo framework. *Governance and Technology Series*. https://doi.org/10.1007/978-3-319-33570-4_7
5. Sierra, C., Sabater, J., Augusti, J., & Garcia, P. (2004). The SADDE methodology: Social agents design driven by equations. In: Methodologies and software engineering for agent systems. Springer, Boston.
6. Gasser, L. (2001). *Perspectives on organizations in multi-agent systems* (pp. 1–16). Berlin: Springer.
7. Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., & Santi, A. (2013). Multi-agent oriented programming with JaCaMo. *Science of Computer Programming, 78*(6), 747–761. https://doi.org/10.1016/j.scico.2011.10.004
8. Cardoso, R. C., & Ferrando, A. (2021). A review of agent-based programming for multi-agent systems. *Computers, 10*, 1–15. https://doi.org/10.3390/computers10020016
9. Galbraith, J. R. (1995). *Designing organizations: an executive briefing on strategy, structure, and process*. San Francisco, USA: Jossey-Bass Publishers.
10. DeLoach, S.A., & Matson, E. (2004). An organizational model for designing adaptive multiagent systems. In: The AAAI-04 workshop on agent organizations: Theory and practice (AOTP 2004), pp. 66–73. AAAI Press, San Jose, USA.
11. Horling, B., & Lesser, V. (2008). Using quantitative models to search for appropriate organizational designs. *Autonomous Agents and Multi-Agent Systems, 16*(2), 95–149. https://doi.org/10.1007/s10458-007-9020-y

12. Sims, M., Corkill, D., & Lesser, V. (2008). Automated organization design for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*. https://doi.org/10.1007/s10458-007-9023-8
13. So, Y.-P., & Durfee, E. H. (1998). Designing organizations for computational agents. *Computational Organization Theory (Simulating Organizations), 2*, 47–64. https://doi.org/10.1007/BF00127275
14. Slade, S. (2018). Going horizontal: Creating a non-hierarchical organization, one practice at a time, 1st edn., pp. 1–204. Berrett-Koehler Publishers, Inc., Oakland, CA.
15. Kühne, T. (2006). Matters of (meta-) modeling. *Journal on Software and Systems Modeling, 5*, 369–385.
16. Stoner, J. A. F., & Freeman, R. E. (1992). *Management* (1st ed.). New Jersey, USA: Prentice-Hall.
17. Burton, R. M., Obel, B., & Desanctis, G. (2011). *Organizational design: A step-by-step approach* (p. 272). Cambridge, UK: Cambridge University Press.
18. De Pinho Rebouças De Oliveira, D. (2006). Estrutura organizacional: Uma abordagem para resultados e competitividade. Editora Atlas, São Paulo, Brazil.
19. Wu, Z., Deng, S., & Wu, J. (2015). Chapter 7 - service composition. In Z. Wu, S. Deng, & J. Wu (Eds.), *Service computing* (pp. 177–227). Boston: Academic Press.
20. Amaral, C. J., & Hübner, J. F. (2019). Goorg: Automated organisational chart design for open multi-agent systems. In F. De La Prieta, A. González-Briones, P. Pawleski, D. Calvaresi, E. Del Val, F. Lopes, V. Julian, E. Osaba, & R. Sánchez-Iborra (Eds.), *PAAMS* (pp. 318–321). Cham: Springer.
21. Amaral, C. J., & Hübner, J. F. (2020). From goals to organisations: Automated organisation generator for mas. In L. A. Dennis, R. H. Bordini, & Y. Lespérance (Eds.), *Engineering multi-agent systems* (pp. 25–42). Cham: Springer.
22. Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research, 7*, 83–124. https://doi.org/10.1613/jair.433
23. Ferber, J., & Gutknecht, O. (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. https://doi.org/10.1109/ICMAS.1998.699041
24. Hübner, J.F., & Sichman, J.S. (2003). Organização de sistemas multiagentes. III Jornada de Mini-Cursos de Inteligência Artificial JAIA03 **8**, 247–296.
25. Hatch, M. J. (1997). *Organization theory: Modern, symbolic, and postmodern perspectives*. Oxford, UK: Oxford University Press.
26. Sims, M., Corkill, D., & Lesser, V. (2004). Separating domain and coordination in multi-agent organizational design and instantiation. In: Proceedings - IEEE/WIC/ACM international conference on intelligent agent technology. IAT 2004, pp. 155–161. https://doi.org/10.1109/IAT.2004.1342938
27. Decker, K.S. (1995). Environment centered analysis and design of coordination mechanisms, p. 219. PhD Thesis, University of Massachusetts, Massachusetts, USA.
28. Sleight, J. (2014). Agent aware organizational design (doctoral consortium). In: Proceedings of the 2014 international conference on autonomous agents and multi-agent systems. AAMAS '14, pp. 1739–1740, Paris, France. https://doi.org/10.5555/2615731.2616153
29. Cardoso, R.C., & Bordini, R.H. (2019). Decentralised planning for multi-agent programming platforms. In: AAMAS'19: Proceedings of the 18th international conference on autonomous agents and multiagent systems, pp. 799–807 . https://doi.org/10.1007/978-3-642-02377-4_4
30. Fink, S. L., Jenks, R. S., & Willits, R. D. (1983). *Designing and managing organizations* (1st ed.). Ilinois, USA: Irwin Series in Financial Planning and Insurance. R.D. Irwin.
31. Ye, D., Zhang, M., & Vasilakos, A. V. (2016). A survey of self-organisation mechanisms in multi-agent systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems,*. https://doi.org/10.1109/TSMC.2015.2504350
32. Sleight, J.L., Durfee, E.H., Baveja, S.S., Cohn, A.A.E.M., & Lesser, E.V.R. (2015). Agent-driven representations, algorithms, and metrics for automated organizational design. PhD thesis, University of Michigan..
33. Ishida, T., Gasser, L., & Yokoo, M. (1992). Organization self-design of distributed production systems. *IEEE Transactions on Knowledge and Data Engineering, 4*(2), 123–134. https://doi.org/10.1109/69.134249
34. Decker, K., Sycara, K., & Williamson, M. (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. https://doi.org/10.1007/BFb0030082
35. Shehory, O., Sycara, K., Chalasani, P., & Jha, S. (1998). Agent cloning: An approach to agent mobility and resource allocation. *IEEE Communications Magazine, 36*(7), 58–6367. https://doi.org/10.1109/35.689632

36. Kamboj, S., & Decker, K. S. (2007). Organizational self-design in semi-dynamic environments. *AAMAS*. https://doi.org/10.1145/1329125.1329370
37. Labella, T. H., Dorigo, M., & Deneubourg, J.-L. (2007). Division of labor in a group of robots inspired by ants' foraging behavior. *ACM Transactions on Autonomous and Adaptive Systems, 1*(1), 4–25. https://doi.org/10.1145/1152934.1152936
38. Kota, R., Gibbins, N., & Jennings, N. R. (2012). Decentralized approaches for self-adaptation in agent organizations. *ACM Transactions on Autonomous and Adaptive Systems, 7*(1), 1–28. https://doi.org/10.1145/2168260.2168261
39. Ye, D., Zhang, M., & Sutanto, D. (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*(4), 887–897. https://doi.org/10.1109/TPDS.2013.120
40. Ohta, N., Iwasaki, A., Yokoo, M., Maruono, K., Conitzer, V., & Sandholm, T. (2006). A compact representation scheme for coalitional games in open anonymous environments. *Proceedings of the National Conference on Artificial Intelligence, 1*(1994), 697–702.
41. Krausburg, T., Dix, J., & Bordini, R.H. (2021). Computing sequences of coalition structures. In: 2021 IEEE symposium series on computational intelligence (SSCI), pp. 01–07. https://doi.org/10.1109/SSCI50451.2021.9660127.
42. Pattison, H. E., Corkill, D. D., & Lesser, V. R. (1987). Chapter 3 - instantiating descriptions of organizational structures. In M. N. Huhns (Ed.), *Distributed Artificial Intelligence* (pp. 59–96). San Francisco: Morgan Kaufmann.
43. McAuley, J., Duberley, J., & Johnson, P. (2007). *Organizational theory: Challenges and perspectives* (1st ed., p. 448). New Jersey: Prentice-Hall.
44. Katz, D., & Kahn, R. (1987). *Psicologia social da organizações* (3rd ed., p. 512). São Paulo, Brazil: Atlas.
45. Hübner, J. F., Boissier, O., Kitio, R., & Ricci, A. (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*(3), 369–400. https://doi.org/10.1007/s10458-009-9084-y
46. Dastani, M., Dignum, V., & Dignum, F. (2003). Role-assignment in open agent societies. In: Proceedings of the second international joint conference on autonomous agents and multiagent systems - AAMAS '03, p. 489. https://doi.org/10.1145/860575.860654.
47. Mintzberg, H. (1983). *Structure in fives* (1st ed.). New Jersey: Prentice-Hall.
48. Daft, R. L. (2009). *Organization theory and design* (10th ed.). USA: South-Western College Pub Centage Learning.
49. Durfee, E. H., Lesser, V. R., & Corkill, D. D. (1987). Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers, C–36*(11), 1275–1291. https://doi.org/10.1109/TC.1987.5009468
50. Kilmann, J., Shanahan, M., Toma, A., & Zielinski, K. (2010). *Demystifying organization design*. (June: Technical report. Boston Consulting Group - BCG White Paper).
51. Burns, T., & Stalker, G.M. (1994). Mechanistic and organic systems of management. In: The management of innovation vol. 21, pp. 96–125. Oxford University Press, Oxford, UK
52. Pettigrew, A. M., & Fenton, E. M. (2000). *The innovating organization* (1st ed.). London, UK: SAGE Publications.
53. Newman, D. A. (1973). *Organization design: An analytical approach to the structuring of organisations* (1st ed.). London, UK: Edward Arnold.
54. Robbins, S., & Coulter, M. (2012). *Management* (11th ed.). New Jersey, USA: Prentice-Hall.
55. Uez, D. M., & Hübner, J. F. (2014). Environments and organizations in multi-agent systems: From modelling to code. In F. Dalpiaz, J. Dix, & M. B. van Riemsdijk (Eds.), *Engineering multi-agent systems* (pp. 181–203). Cham: Springer.
56. Matson, E.T., & Deloach, S.A. (2005). Autonomous organization-based adaptive information systems. In: 2005 international conference on integration of knowledge intensive multi-agent systems, KIMAS'05: Modeling, exploration, and engineering **2005**, 227–234.
57. Deloach, S.A., Oyenan, W.H., & Matson, E.T. (2008). A capabilities-based model for adaptive organizations. In: Autonomous agents and multi-agent systems, pp. 13–56. https://doi.org/10.1007/s10458-007-9019-4.
58. Horling, B., & Lesser, V. (2004). A survey of multi-agent organizational paradigms. *Knowledge Engineering Review, 19*(4), 281–316. https://doi.org/10.1017/S0269888905000317
59. Seidewitz, E. (2003). What models mean. *IEEE Software, 20*, 26–32. https://doi.org/10.1109/MS.2003.1231147
60. Mintzberg, H., & Van der Heyden, L. (1999). Organigraphs: Drawing how companies really work. *Harvard Business Review, 77*(5).

61. Lesser, V., Ortiz, C.L., & Tambe, M. (eds.) (2003). Distributed sensor networks: A multiagent perspective, pp. 1–376. Springer, USA.
62. Dean, T., & Boddy, M. (1988). An analysis of time-dependent planning. In: Proceedings of the seventh AAAI national conference on artificial intelligence. AAAI'88, pp. 49–54. AAAI Press, Saint Paul, Minnesota.