RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ MOHAMED KHIDER - BISKRA

N° d'ordre :.....
Serie :.....

– **FACULTÉ DES SCIENCES EXACTES ET DES SCIENCES DE LA NATURE ET DE LA VIE** –
— **DÉPARTEMENT D'INFORMATIQUE** —

THÈSE

présentée pour obtenir le diplôme de

DOCTORAT EN SCIENCES

SPÉCIALITÉ : **INFORMATIQUE**

A Graph Transformation Approach for Dynamic Reorganization in Multi-Agent Systems

---

Par

Fayçal GUERROUF

Soutenue le : / /
Devant le jury composé de :

Pr. Okba KAZAR,	Professeur à l'Université de Biskra, Algérie	Président
Pr. Allaoua CHAOUI,	Professeur à l'Université Constantine 2, Algérie .	Rapporteur
Pr. Hammadi BENNOUI,	Professeur à l'Université de Biskra, Algérie	Examinateur
Pr. Laid KAHLOUL,	Professeur à l'Université de Biskra, Algérie	Examinateur
Dr. Elhillali KERKOUCHE,	MCA à l'Université de Jijel, Algérie	Examinateur
Dr. Toufik MAAROUK,	MCA à l'Université de Khenchela, Algérie	Examinateur

To my dear parents with all my love and gratitude ....

To my lovely wife and my little angels Soumia, Hiba, and Sohayb ....

To my brothers and my sisters ....

To my friends ....

I dedicate this thesis.

Acknowledgment

I would like to express my sincere gratitude to my supervisor Allaoua Chaoui for his constant guidance, insightful comments, and considerable encouragement to complete this thesis.

Special thanks to Prof. Kazar Okba, who always offers his unlimited support to the LINFI laboratory members.

I would also like to express my gratitude to the jury members who give me the honor by accepting to evaluate and review this work.

Abstract

Nowadays, complexity and high distribution are considered as the main properties of large real-world systems such as airports and manufacturing systems. The Organization Centered Multi-Agent System (OCMAS) approach is advocated as an appropriate solution to handle the complexity and the distribution of such systems. These systems are often open, and execute under a dynamic environment with unpredictable interaction. In fact, the concept of organization that is explicitly defined in OCMAS provides a key feature to the designed system which is the "*stability*". Despite that, when the Multi-agent system operates in dynamically changing environments and often unreliable communication resulting in various events originating from its external environment and/or its internal elements which *de-stabilize* the system state. To overcome this kind of undesirable situation, the Multi-Agent System has to reorganize its behavior and structure to continue fulfilling its overall objectives. In this dissertation, we proposed a formal approach for the reorganization of a multi-agent system statically at design time or dynamically at run-time. Indeed, we proposed to formulate the reorganization using graph transformation. In particular, we have defined a type graph to represent the multi-agent system organization and a set of rules that define the different actions that can be performed to reorganize the system. We implemented our approach using AGG. We evaluated our approach on a case study related to a manufacturing system implemented as a Multi-Agent System. The obtained results show the efficiency and the effectiveness of our approach.

Keywords: Multi-Agent System, Organization Centered Multi-Agent System, Reorganization., Complex System, AGG, Graph Transformation

Résumé

De nos jours, la complexité et la distribution sont considérées comme les principales propriétés des grands systèmes du monde réel tels que les systèmes des aéroports et de fabrication. L'approche "Organization Centered Multi-Agent System" (OCMAS) est considéré comme solution appropriée pour gérer la complexité et la distribution de ces systèmes. Ces systèmes sont souvent ouverts et s'exécutent dans un environnement dynamique avec une interaction imprévisible. En fait, le concept d'organisation qui est explicitement défini dans OCMAS fournit une caractéristique clé du système conçu qui est la "*stabilité*". Malgré cela, lorsque le système multi-agent fonctionne dans des environnements changeants de manière dynamique et souvent des communications non fiables qui résultent de divers événements provenant de son environnement externe et / ou de ses éléments internes peuvent *déstabiliser* l'état du système. Pour surmonter ce genre de situation indésirable, le système multi-agents doit réorganiser son comportement et sa structure pour continuer à remplir ses objectifs généraux. Dans cette thèse, nous avons proposé une approche formelle pour la réorganisation d'un système multi-agents statiquement au moment de la conception ou dynamiquement au moment de l'exécution. En effet, nous avons proposé de formuler la réorganisation en utilisant la transformation de graphe. En particulier, nous avons défini un type graph pour représenter l'organisation du système multi-agents et un ensemble de règles qui définissent les différentes actions qui peuvent être effectuées pour réorganiser le système. Nous avons implémenté notre approche en utilisant AGG. Nous avons évalué notre approche sur une étude de cas liée à un système de fabrication implémenté en tant que système multi-agents. Les résultats obtenus montrent l'efficacité de notre approche.

Mots clés:

Système Multi-Agents, Système Multi-Agents centré sur l'Organisation, Réorganisation, Système Complexe, AGG, Transformation de Graphes

ملخـص

في الوقت الحاضر ، يعتبر التعقيد والتوزيع العالي من الخصائص الرئيسية للأنظمة الموجودة في العالم الحقيقي مثل أنظمة المطارات و التصنيع. نهج النظام متعدد الوكلاء المتمركز على المنظمة يعتبر كحل مناسب للتعامل مع تعقيد وتوزيع مثل هذه الأنظمة. غالبًا ما تكون هذه الأنظمة مفتوحة ويتم تنفيذها في بيئة ديناميكية مع تفاعل غير متوقع. في الواقع ، يوفر مفهوم " النظام متعدد الوكلاء المتمركز على المنظمة " ميزة أساسية للنظام المصمم وهي " الثبات " .على الرغم من ذلك ، عندما يعمل النظام متعدد العوامل في بيئات متغيرة ديناميكيًا وغالبًا ما تكون الاتصالات غير موثوق فيها تؤدي إلى أحداث مختلفة تنشأ من بيئتها الخارجية و / أو عناصرها الداخلية التي تزيل استقرار و ثبات النظام. للتغلب على هذا النوع من المشاكل غير المرغوب فيها ، يتعين على النظام متعدد العوامل إعادة تنظيم سلوكه وهيكله لمواصلة تحقيق أهدافه العامة. في هذه الأطروحة ، اقترحنا نهجًا رسميًا (شكلي) الإعادة تنظيم نظام متعدد الوكلاء في وقت التصميم و في وقت التشغيل. في الواقع ، اقترحنا صياغة إعادة التنظيم باستخدام تحويل الرسوم البيانية. على وجه الخصوص ، قمنا بتعريف نوع رسم بياني لتمثيل مؤسسة النظام متعدد الوكلاء ومجموعة من القواعد التي تحدد الإجراءات المختلفة التي يمكن تنفيذها لإعادة تنظيم النظام. قمنا بتنفيذ نهجنا باستخدام *AGG* . قمنا بتقييم نهجنا في دراسة حالة تتعلق بنظام تصنيع يتم تنفيذه كنظام متعدد العوامل. تظهر النتائج التي تم الحصول عليها كفاءة وفعالية نهجنا.

الكلمات المفتاحية: نظام متعدد الوكلاء ، نظام متعدد الوكلاء المتمركز على المنظمة ، إعادة التنظيم ، النظام المعقد ، تحويل الرسوم البيانية.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Context

Nowadays, complexity and high distribution are considered as the main properties of large real-world systems. The latter are mainly composed of many aspects, such as distribution of control and implication of a significant number of components. Such systems are often open, and execute under a dynamic environment with unpredictable interaction. Examples of these systems are airports, manufacturing systems, etc. Besides, the classical life cycle of software engineering is not the suitable paradigm for developing such systems (they require a nontraditional approach). Indeed, agent-oriented software engineering (AOSE) is considered as one of the most popular paradigm that is used to develop complex and distributed systems [Jen99; DeL09]. It is applied in a variety of domains, such as: social sciences [Saw03], information retrieval, distributed data mining, robotics, e-commerce, networks, virtual reality, biological simulations, etc [Woo09]. Moreover, AOSE uses concepts of a multi-agent system such as agents, and organization (societies) of agents which allow to abstracting the complexity of systems. In addition, the convenient choice of multi-agent systems for the development of complex systems over classical software engineering arises from the fact that the manifested complexity in a system is presented naturally with a multi-agent system [Jen01b]. Furthermore, systems that are developed with multi-agent paradigm are in general, fast and efficient thanks to their asynchronous and parallel computation nature. They are also scalable and flexible due to the simplicity of adding and removing agents from the system [FGM04].

In the literature, two viewpoints of AOSE are proposed [FGM04; Pic+09] : i) Agent-Centered Multi-Agent System (ACMAS); ii) Organization-Centered Multi-Agent System (OCMAS). ACMAS focuses on the micro-level of a Multi-agent System, i.e., at the level of the agent's states itself and their relationship to its overall behavior. ACMAS considers the agents as the force that generate and drive the organization. This latter is created implicitly at run-time with no prior design. The collective behavior of the cooperation pattern between agents shapes the organization's structure following a bottom-up approach (start at the agent level). Agents must figure out how to organize amongst

themselves, and the organization emerges as an observable phenomenon.

Unfortunately, ACMAS suffers from two significant drawbacks when designing large systems, namely; unpredictability and uncertainty. Predicting the overall behavior of the system from the interaction and cooperation of its agent is extremely difficult and uncertain due to the probability of unwanted emergent behaviors [FGM04]. Additional drawbacks are related to the security of applications, modularity, uniqueness of framework or approach, etc. The drawback of ACMAS has forced to the second type of design where the importance is given to the use of organizational concepts within MAS such as "organizations", "groups", "communities", "roles", "functions", etc [FG98; Jen00; PO01; CD96; ZV02].

Ferber et al. consider OCMAS as the modern design of MAS, and it allows to eliminate the drawbacks of ACMAS [FGM04]. Indeed, it starts from the opposite direction of ACMAS, in which the organization is explicitly defined before running the system. So, the organization is a first-class citizen that holds all the system elements in addition to agents. In other words, it is a top-down approach where the designer defines the organization and the cooperation pattern in order to specify or constrain the agent's behaviors. Thus, agents who play in the system have to comply with the rules that are imposed by the organization. In this type of design, the organization's agents are aware of the system's structure and state, which gives them the capability to manipulate primitives to alter their social environment.

Several pioneers [FGM04; Gas01; OPF03; HL04; Van+05; HVB08; Dig09] in the field of MAS have advocated OCMAS as a solution for mastering the complexity of systems. In addition, they argue that using the concept of organization in MAS allows to increase the efficiency and improves system scalability [HL04]. Furthermore, the system's abstraction with an organization decreases or controls its uncertainty and unpredictability. Also, it allows formalizing global goals that require the awareness of a set of agents instead of a single one. Moreover, organization structure helps to impose rules on agents' behaviors and improve the achievement of coordination effectively [Dig09].

1.2 The problem studied in the thesis

In this thesis, we are interested in studying the following general question:

How to ensure the stability of a complex system even in a dynamic environment?

Despite complex systems that are developed with an OCMAS approach are supposed to be stable, but unfortunately, these systems can be subject to destabilization caused by the fact that organizations are operating under uncertainty, in dynamically changing environments, and often unreliable communication which result in a multitude of events. Therefore, these characteristics make organizations dynamic where they can evolve, disappear, or expand. An example of the events in a manufacturing system is when a product is no longer in demand, and it causes the need of a producing a better product with the best features. Hence, the organization can evolve to realize this objective, or simply disappear. When there is an over-demand for a particular product, the organization can be expanded by creating a production line. In the context of MAS, events can be a change in organization objectives or when agents leave the system for any given reason. Besides, some undesirable events can negatively affect the organization, which causes degradation to its performance and effectiveness. For instance, production can be suspended when a machine is broken. It is obvious that this can affect negatively the performance. Therefore, flexibility and adaptation (a.k.a. reorganization) are essential proprieties for an organization to acquire for achieving its overall objective. Without these properties, the system goes into a state of failure.

We distinguish two types of adaptation (reorganization) : *static reorganization* (at design time) and *dynamic reorganization* (at run-time). i) *Static reorganization*: carried out at design time by the system designer using different tools such as prototyping, model checking, and simulation tools. In this case, the designer has to perform the adaptation by changing the system's model statically. Unfortunately, not all undesirable situations can be identified at this stage, but they can emerge at run-time. ii) *Dynamic reorganization*: refers to the modification of an organization during the execution of the system. The

reorganization can affect its two dimensions; structural (static) and behavioral dimensions such as substitution, addition, removal of elements. Therefore, the reorganization process should include the definition of situations changing the organization's behaviors, such as the departure or the arrival of agents, as well as the organization's structure, such as defining new goals. [Pic+09]

The problem tackled in this thesis is to find a way to precisely define the reorganization in complex systems that are developed using OCMAS. Indeed, we focus on how to accurately describe the effect of the dynamics of a given complex system. We are interested in finding a manner to describe the reorganization unambiguously. The reorganization can affect either the behavioral or the structure of the system. In fact, we want to describe the series of system reactions it must takes in response to undesirable events occurring in the system.

1.3 Contributions

In this dissertation, we propose a formal approach for the reorganization of MAS at design time and run-time to overcome most of the predictable and unpredictable events, which keeps the system in a functional and stable state. Our approach consists of providing a formal definition of the process of reorganization. First, we describe the MAS organization and the reorganization process as three main components, namely i) MAS monitor, ii) MAS organization, iii) Reorganization Manager, where each one represents a particular concept of the MAS organization. The three components interact with each other in a specific manner. Besides, we use as formal tool the Graph Grammar, which has a solid mathematical foundation that allow proofing properties such as termination and consistency. Moreover, the Graph Grammar is used to specify and verify a variety of complex and distributed systems [Ehr+15a]. Additionally, the existence of tools support such as AGG [RET12], GROOVE [Ren04], facilitate the operation of edition and offer different techniques for analysis purposes such as model checking, critical pair analysis, etc. Therefore, we use in our approach graph grammar concepts to define a Multi-Agent

System and describe the process of reorganization. The main contributions of this thesis are : i) MAS organization type graph definition, ii) MAS organization rules definition, and iii) An evaluation of the proposed formal definition. These contributions can be summarized as follow:

1. **MAS organization type graph definition** : We have defined a type graph that acts as a meta-model for representing the structure (state) of a given MAS. The system state is represented as an instance of such meta-model.

2. **MAS organization rules definition**: We have defined a set of rules representing possible actions that can be performed to reorganize the system. These rules are used to monitor and describe the behavior of a Multi-Agent System. In our approach, the static reorganization is expressed with the different mechanisms that a designer (or an administrator with privileged access) can modify not only at design time but also at runtime. These mechanisms can be used for the behavioral part (such as restricting the organization to some types of agents) and the structural part (such as defining priority between goals, preventing a role from being executed more than a defined amount of time). The dynamic reorganization is expressed through the application of different graph transformation rules. These rules are equipped with three mechanisms: i) negative application conditions to prevent them from execution in specific contexts. ii) context conditions to enforce a needed requirement, such as the condition put on agent capabilities to satisfy role requirements. iii) rule priory to prioritize the execution of a rule before others. These mechanisms allow defining policies for reorganization that suits different situations and different types of organization. Hence, our approach can be applied in various MAS with a slight modification to the provided mechanism.

3. **An evaluation of the proposed formal definition**: In this evaluation, we use a case study related to a manufacturing system to see the effectiveness of our approach. Indeed, we use a set of scenarios of different sizes. For each scenario, we show how our specification is used and which rules

can be applied. In the end, we perform a semi-automatic check to see how the system still consistent and stable.

1.4 Thesis Outline

The rest of this thesis is organized as follows:

Chapter 2: gives an overview of graph transformation system and Multi-Agent System. We first introduced the standard definition of simple graphs and then the concepts of graph transformation systems using typed graphs. Second, we go through the concepts of multi-agent systems, in which we focus on organization and what relates to it, such as the elements of an organization, and the process of reorganization, etc.

Chapter 3: provides an overview of the state-of-the-art of formalizing Multi-agent systems using different techniques. In fact, we divided the proposed contributions in literatures into three categories which are; i) approaches that use semi-formal notation such as the unified modeling language to design and describe the system, ii) approaches that use graph transformation to describe the different aspects of the system, and finally, iii) approaches that use formal languages to specify the system.

Chapter 4: presents the proposed approach for formalizing reorganization in MAS using graph transformation. It comprises a general picture of the approach as well as detailed explanations of its different components. After that, we present the defined type graph and the MAS organization rules. At the end of this chapter, we present the mathematical notation and details some properties of the formalized MAS organization.

Chapter 5: introduces the details of our evaluation. We present the selected case study and the different scenarios that are used to validate our approach.

Chapter 6: concludes the dissertation and draws some future directions.

Part I

State of the Art

CHAPTER 2

Background

2.1 Introduction

In this chapter, we give a background about the context of this work, which helps in the understanding of the concepts that are used in this thesis. We begin (in Section 2.2) with an overview of graph transformation. Subsequently, we introduce simple graphs, typed graphs, and typed attributed graphs. In section 2.3, we present a Multi-Agent System in general and, more specifically, the organizational point of view in Multi-Agent System. We start by defining what an organization is and the different types of agent organizations. Then we present what motivates a change in an organization and its types. Finally, we detail the concept of reorganization and give a general process of it.

2.2 Graph Transformation System

Graph transformation systems allow transforming a given graph by application of the rules of a given graph grammar. A rule is applicable whenever a match to its left-hand side is found in the working graph. Meaning an inclusion from the left rule side into the working graph can be found such that the application conditions of the rule are fulfilled. Besides, In an attributed graph, each graph object of the source graph (left rule side) has to have a matching graph object in the target graph (working graph), such that also the object's attributes match.

The application of graph grammar rules in the graph rewriting approach allows deriving a graph from another.

The algebraic approaches [Cor+97] [Ehr+97], node replacement [ER97], and edge replacement [DKH97] are examples of various approaches to graph transformation, and the most prominent to algebraic graph transformation are the single pushout (SPO) approach and double pushout (DPO).

In our work, we used the algebraic graph transformation according to the double pushout approach [EPS73]. These approaches are based on graphs and graph morphisms. In this section, we review the basic definitions of this approach. These definitions are standard in this field, and more details can be

found in [Ehr+15b].

2.2.1 Graph and Graph Morphism

Essentially, a graph comprises edges and nodes (called vertices also). An edge is a link between two nodes. Formally a graph is defined as follows:

Definition 2.1 (Graph).
A graph $G = (V, E, s, t)$ where

- *V is a set of nodes.*
- *E is a set of edges.*
- *$s : E \to V$ is the source function.*
- *$t : E \to V$ is the target function.*

The purpose of graph morphisms is to match a graph or sub-graph to another graph, in which all the elements of one graph (edges and vertices) are mapped into the corresponding elements of another graph while preserving the structure of a graph. (i.e., if an edge n_1 is mapped to an edge n_2, the source and target vertices of n_1 must be accordingly mapped to the source and target of n_2). Formally, it is defined as follows:

Definition 2.2 (Graph Morphism).
Let G, H be two graphs. A graph morphism $f : G \to H, f = (f_V, f_E)$, consists of two functions $f_V : V_G \to V_H, f_E : E_G \to E_H$ such that $s_H \circ f_E = f_V \circ s_G$ and $t_H \circ f_E = f_V \circ t_G$.

Figure 2.1 illustrates The category $Graphs$ formed with graph and graph morphisms.

2.2.2 Typed Graph and Typed Graph Morphism

A typed graph is defined with a type graph and a type graph morphism. The nodes and edges of the type graph are types that can be used to assign types

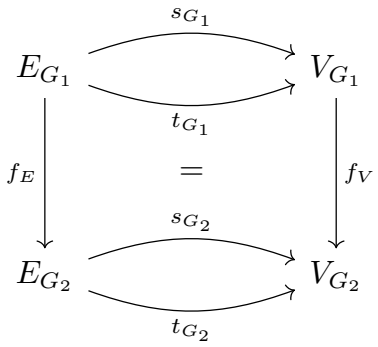


Figure 2.1: $Graphs$ Category Diagram

to the nodes and edges of the typed graph. This typing is done using a typed graph morphism between the type graph and the typed graph.

The relation between typed graph and type graph can be seen as the relation between model and meta-model where the model must conform to the meta-model. To formally define a typed graph and typed graph morphism we start by the definition of a type graph:

Definition 2.3 (Type Graph).
A type graph is a distinguished graph $T_G = (V_{T_G}, E_{T_G}, s_{T_G}, t_{T_G})$. where

- *V_{T_G} is a set of nodes type alphabets.*
- *E_{T_G} is a set of edges type alphabets.*
- *$s_{T_G} : E \to V$ is the source function.*
- *$t_{T_G} : E \to V$ is the target function.*

Using this definition of a type graph, we continue to define a typed graph as follow:

Definition 2.4 (Typed Graph).
a typed graph is defined as a tuple $(G, type)$ where :

- *G is a graph.*
- *$type : G \to T_G$. is a graph morphism called the typing of G*

a typed graph morphism is no different to ordinary graph morphism as they are used to match a graph or sub-graph to another graph except that the

used graphs are typed and typed with the same type graph. the formal definition of typed graph morphisms is as follows:

Definition 2.5 (Typed Graph Morphism).
Given typed graphs $G_T = (G, type_G)$ and $H_T = (H, type_H)$, a typed graph morphism $f : G_T \to H_T$ is a graph morphism $f : G \to H$ such that $type_H \circ f = type_G$

Considering a type graph TG, the category $Graphs_{TG}$ formed with typed graphs and typed graph morphisms is illustrated in Figure 2.2.

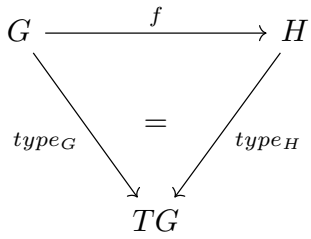


Figure 2.2: $Graphs_{TG}$ Category Diagram

2.2.3 Graph Rule

A rule (production) is an overall specification of local changes that may take place in graphs. Generally, it consists of a left-hand side (LHS), a right-hand side (RHS), and a mechanism that describes how to replace LHS by RHS. Formally it is defined as follows:

Definition 2.6 (Rule).
A typed graph rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ where :

- *L : is a typed graph called the left-hand side.*
- *R : is a typed graph called the right-hand side.*
- *K : is a typed graph called gluing graph.*
- *l, r : are typed graph morphisms.*

2.2.4 Concept of Transformation

A transformation is a sequence of direct transformations. It specifies how a rule is applied to a graph using a match. Formally it is defined as follows:

Definition 2.7 (Transformation).
Given a typed graph rule $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$, a typed graph G, and a typed morphism $m : L \rightarrow G$ called match, a direct typed transformation $G \overset{p;m}{\Longrightarrow} H$ from G to a graph H is given by the pushouts (1) and (2) (Figure 2.3).

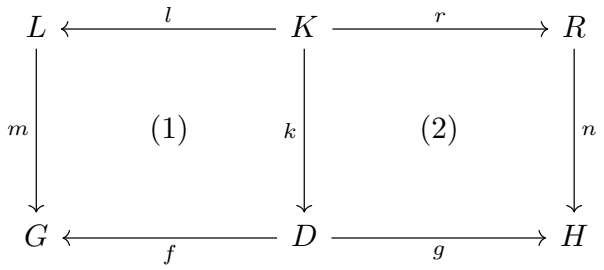


Figure 2.3: Double pushout construction from G to H (Direct derivation)

2.2.5 Negative Application Conditions

A negative application condition (NAC) is used to define a banned context that prevents rule application or to prohibit the application of the same typed graph production infinitely. Formally a NAC is defined by:

Definition 2.8 (NAC).
A simple negative application condition is of the form $NAC(x)$, where $x : L \rightarrow X$ is a (typed) graph morphism. A (typed) graph morphism $m : L \rightarrow G$ satisfies $NAC(x)$ if there does not exist an injective (typed) graph morphism $p : X \rightarrow G$ with $p \circ x = m$. The Figure 2.4 shows the NAC structure to the double pushout construction.

2.2.6 Typed Attributed Graph

many applications require more than a simple graph to represent their complex data structure. Therefore, nodes and edges are enriched with attributes

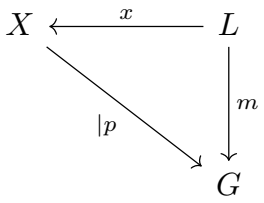


Figure 2.4: NAC Structure

of a given data types (e.g. boolean, integer, string) to stores additional information. Attributes can be used to define guards to restrict the applicability of transformation rules. For example, a rule can only be applied if a certain attribute is above some threshold). Also, it is possible to perform computation on them to deduce useful information for the application.

As in object-oriented languages, an attribute in the context of typed graphs is declared by its name and data type in the type graph. After declaration, each node in the instance graph can have different values in the attribute having the same data type.

To formally define the attributed graphs, the classical notion of graphs is extended to E-graphs to allow nodes and edges to store additional information through attributes. In an E-graph, there are two distinct kinds of nodes representing the traditional graph nodes and data nodes that carry values for attributes. An E-graph also has three kinds of edges as well, the normal graph edges and special edges used for the node and edge attribution. To formalize the problem of reorganization in a Multi-Agent System, we used a typed attributed graph. However, to simplify our work, we will be using a typed graph as it is also valid. Thus, we gave just this informal definition. The complete formal definition can be found in [Ehr+15b].

2.2.7 Typed Graph Transformation System

simply, a typed graph transformation system is composed of a type graph and a set of typed rules.

Definition 2.9.

A typed graph transformation system $GTS = (T_G, P)$ consists of a type graph T_G and

a set of typed graph productions P.

2.2.8 Typed Graph Grammar

A typed graph grammar is a combination of a typed graph transformation system and a typed start graph.

Definition 2.10.
A typed graph grammar $GG = (GTS, S)$ consists of a typed graph transformation system GTS and a typed start graph S.

2.2.9 Graph Transformation Tools

Many graph transformation tools such as AGG [Tae99], $AToM^3$ [DV02], VIATRA [Cse+02], and FUJABA [NNZ00] are available for many different purposes such as model transformation, model checking, rapid system prototyping, and state-space exploration.

The main capacity of these software tools is to automate the process of graph transformation. They are fed with a graph transformation system with a set of initial graphs and return the resulted transformation.

Many of these tools have a graphical user interface that can be used to graphically create and edit graphs and transformation rules. Others are terminal-based and require the user to specify graphs and graph transformation rules textually.

They are equipped with additional functions to fulfill the different purposes for which they are created.

In our work, we used AGG to implement our approach. In the next section, we give a brief description of this tool.

2.2.9.1 Attributed Graph Grammar System (AGG)

The development environment for attributed graph transformation systems *AGG* [Tae99] targets prototyping and rapid specification of applications with

complex, graph-structured data. It allows :

- The edition of graphs and rules.
- Attribution of graphs and rules with basic java data types and object classes.
- Additionally, Rules may be attributed with Java expressions, which are evaluated during rule applications and conditions that are boolean Java expressions.
- Analysis of graph grammar using critical pair analysis and consistency checking.
- Simulation.
- Controle the applicability of Rules using negative application conditions and layers.

2.3 Agent-Oriented Software Engineering

In this section, we will introduce the main concepts related to the agent-oriented software engineering.

2.3.1 Concept of Agent

Despite the progress made in the last decades, there is no agreement about what an agent is [Woo09]. Hence, many definitions of agents have been proposed in the literature. The following are some of the most notable definitions:

Starting from a linguistic point of view, The term agent comes from the Latin word *"agere"* which means to do and refers to the capacity of an entity to do or to act.

Wooldridge [Woo09] define "*an agent as a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives*".

For Franklin and Graesser [FG97] "*an autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future*".

Similarly, Knapik and Johnson [KJ98] define an agent as "*a piece of software which performs a given task using information gleaned from its environment to act in a suitable manner so as to complete the task successfully. The software should be able to adapt itself based on changes occurring in its environment, so that a change in circumstances will still yield the intended result*".

Also, for Ferber and Weiss [FW99] "*an agent is an entity that perceives its environment and acts autonomously in accordance with the information gathered*".

Finally, the definition which we will use in this thesis is that of Russell, Norvig, and Davis [RND10]. they define an agent as "*anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators*". This definition is illustrated in Figure 2.5, which depicts the interaction of an agent with its surrounding environment.

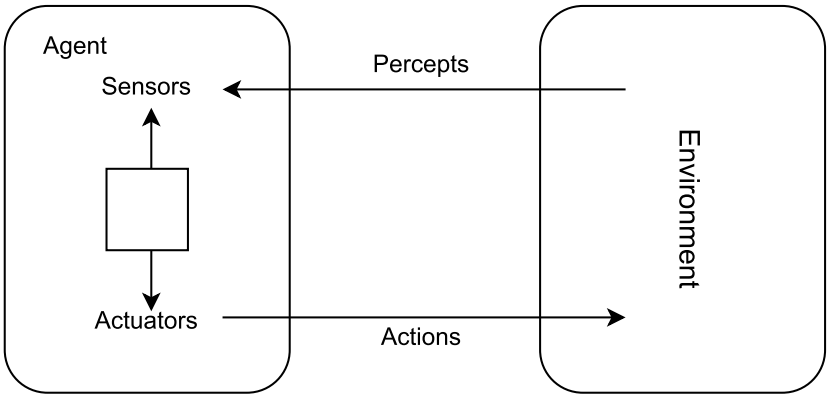


Figure 2.5: Agent interacting with its environment [RND10].

Although this variety of definitions, a classification of proprieties by Wooldridge [Woo09] and Russell [RND10] have been widely accepted. The following general properties can be attributed to an agent:

– ***Autonomy***: Agents are capable of performing autonomously without the direct intervention of a third party (human or agent) and control their own actions as well as their internal state.

- ***Reactivity***: Agents exist in an environment that may be the physical world, a collection of other agents, etc. They are able to perceive it and respond in a timely manner to changes that occur in it.
- ***Proactiveness***: In addition to the capability of responsiveness to their environment, they are also able to perform goal-directed behaviors in a proactive manner.
- ***Social ability***: In order to achieve goals in competitive or cooperative behavior, agents must be able to interact with other agents (human or software).

Many other agent properties exist and can influence their engineering. However, they are not the scope of this thesis. The reader can find further details about agents in [RND10].

2.3.2 Multi-Agent System

In order to solve a complex problem, multiple agents must cooperate and coordinate together towards a common goal instead of one agent that is limited in terms of its resources such as computing power, knowledge, and perspective. This manner of grouping agents is called Multi-Agent System. Consequently, a Multi-Agent System is composed of a number of decentralized, autonomous agents. The characteristics of a MAS are:

1. Each agent has partial capabilities and knowledge required for solving the problem. Consequently, it has a limited viewpoint.
2. An agent shares a global goal with other agents in a multi-agent system and its own goal at the same time.
3. Each agent acts towards its own goal; there is no global control system.
4. The computation in a multi-agent system is asynchronous, and the agents are decentralized.

Agents in a MAS are perceiving, reasoning, and acting by collaborating with each other towards a global goal, even though every agent works toward its own goals to find a solution to a certain problem aspect in a MAS environment.

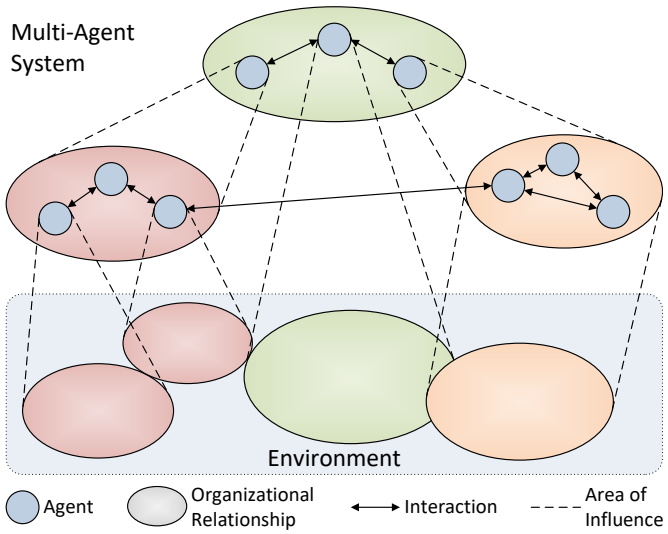


Figure 2.6: General structure of a multi-agent system [Jen01a].

Two-point of view exist for designing a Multi-Agent System: i) The Agent-Centered Multi-Agent System (ACMAS), ii) The Organization-Centered Multi-Agent System (OCMAS).

2.3.3 Agent Centered Multi-Agent System

An agent-centered multi-agent system (ACMAS) is designed with regard to the mental states of agents. Therefore, the designer sees agents as individuals and focus on agents' local behaviors and their interactions without concerning the global structure of the system.

We can say that the agent is the engine that drives the organization. The existence of this latter is the result of the emergent global behavior of agents' individual behaviors and their interactions in a common shared and dynamic environment [FGM04]. This type of Multi-agent system is designed according to a bottom-up approach, which means the designer start by defining the agent interactions' rules (with other agents and the environment) and its behavior.

Consequently, the organization emerges out as a result of the application of these rules.

The bottom-up approach is used to produce systems that are autonomous, scalable, and adaptable, often necessitating minimal (or no) communication [CGL08], such as the control of robotic systems [KZ96; AB97; HM99], embedded systems, sensor networks [IGE00], and information agents [Cha+01], etc.

In order to simplify the design of an ACMAS based system and make them compatible with other systems, the designer of the system should respect the following points [FGM04]:

- No restriction is imposed on the communication between agents. An agent may communicate with any other agent.
- Any agent can access the provided services by any other agent in the system.
- Only the agent itself can constrain its accessibility from other agents.
- Every agent has an ID used to access it from the outside. Thus, agents are presumed to be autonomous, and no constraint is placed on the way they interact.

Drawbacks of ACMAS

What makes a system an ACMAS based system is also what makes its weakness when engineering large systems [Jen00]. According to Jennings, the two major drawbacks are: first, the unpredictability of the interaction patterns and their outcomes. Second, the difficulty or the impossibility to predict the global behavior of the system based on its composing elements because of the high probability of emergent (and unwanted) behavior.

To overcome these drawbacks, an Organization Centered Multi-Agent System is proposed. In the next section, we will see the basic concept of this design model.

2.3.4 Organization Centered Multi-Agent System

The agent-oriented software engineering (AOSE) and social reasoning have influenced this approach, where both the designer and agents use the organization. The former one to specify the desired system, and the latter one to perform organizational acts and possibly modify the organization [Pic+09].

In contrast to the ACMAS approach, the organization in the organization centered multi-agent system (OCMAS) approach is explicitly defined by the designer and exist as an explicit entity of the system [Pic+09]. In this type of system, the cooperating patterns are specified by the designer(or by agents themselves) following a top-down approach. Therefore the designer starts by defining the organization and then the agents' behavior according to organization's imposed rules or norms [Pic+09].

Agents in the OCMAS approach are characterized by:

i) The awareness of the organization they are taking part in.
ii) They are provided with a representation of the organization or some part of it.
iii) They can reason about it and make interactions and relationships to reach their objectives using this knowledge.

In this thesis, we are interested in The OCMAS approach. The rest of this chapter will be dedicated to the concept of organization and what relates to it.

2.3.5 Concept of Organization

In this section, we depict the concept of organization and what relates to it, starting from the basic definition of an organization.

2.3.5.1 Definition

In reality, the term organization is a multidisciplinary term. Its definition differs from one area to another. In the following, we will give definitions related to organization theory and Multi-Agent Systems.

Starting from a linguistic point of view, the Cambridge[1] dictionary defines an organization as: "*a group of people who work together in an organized way for a shared purpose*".

Several definitions were proposed for the concepts of an organization due to the complexity of its meaning. In the following, we present examples of these definitions.

Definition 1: Gasser [Gas92] proposed the following definition of an organization: "An organization provides a framework for activity and interaction through the definition of roles, behavioral expectations and authority relationships (e. g. control)". This definition is very broad and gives no hint as to how organizations should be designed.

Definition 2: a more practical definition proposed by Jennings and Wooldridge in [WJK00]: "We view an organization as a collection of roles, that stand in certain relationships to one another, and that take part in systematic institutionalized patterns of interactions with other roles".

Definition 3: Organization theory defines an organization as an entity that enables an element inside (It can be a person or a moral element) to identify its role as well as the other element's roles to achieve a common goal.

Organizations are created for a particular goal (goals), either statically by a designer, or dynamically by emerging from the collective behavior of several agents. The decision of an agent to join an organization or not is based on how it will contribute to (some of) their goals. They essentially have two purposes [DD12]:

1. Minimize the complexity of decision making.

2. Support coordination across the part of the organization.

In the following section, We will introduce the main components of an organization.

[1] https://dictionary.cambridge.org/dictionary/english/organization

2.3.5.2 Components of organization

the three main components of an organization are

1. **Environment**: In organizational theory, the environment is the space outside the organization exerting all kinds of forces on an organization, and that can impact it. Neither The organization can fully control the changes that may happen over time, nor the individual agents populating it [DD12]. For agents inside the organization, the environment represents [MN10]:

   - The space for their interactions and operation with respect to the constraints (e.g. boundaries) it imposes.
   - The provider of resources they use and consume.
   - The source of events they perceive and consider.

   Different software platform exists as environments for Multi-Agent System that offers services (e.g, communication, life cycle management, or advertisement of agent's services) such as JADE [BCG07] or EVE [JSP13].

2. **Structure**: It describes the different relationship that exists inside the organization and its intended strategy. It is defined in terms of organizational goals and roles.

   i) **Goal**
   a goal descript a desirable situation, such as defining city walls or producing a piece in a manufacturing system. Also, a goal is considered as the objective of a computational process. Furthermore, the goal defines the purpose of the existence of an organization. This latter only exists to fulfill its overall objective.

   ii) **Role**
   generally, a role describes a function inside an organization, and they are used to achieve a particular goal. Hence, they can also be defined as the set of responsibilities required to fulfill a goal. They are similar to the roles played in a real-world organization such as chief the executive officer (CEO).

3. **Agent**
   They are capable entities fulfilling different roles in the organization. We already defined in detail in section 2.3.1 what an agent is.

2.3.6 Types of Agent Organization

It is generally accepted that there is no one type of organization that is appropriate for all situations [IGY92; CL98; Les91; CG99]. The obvious is that what makes a type of organization suitable or not for a particular problem is the set of characteristics and properties it provides.

A variety of organizational types has been proposed in the literature. Namely, hierarchical, Holarchies, Coalitions, Teams, Congregations, Societies, Federations, Markets, Compound Organizations, etc. In the next sections, we summarize some of these Multi-Agent System organizational models.

2.3.6.1 Hierarchical Organization

One of the earliest used to capture the structure of the organizational design in a multi-agent system [MD93]. Simply, the concept of this organization is to arrange the agent in a treelike structure as depicted in Figure 2.7. The interaction between agents is only allowed between entities connected directly. Thus, agents at an elevated level have a more global view than those below them. The flow of information going from the low level to the higher one is to provide a broader view, while the opposite direction is to control agents below.

This hierarchy is easily generated by the natural decomposition possible in many different tasks environments. As a result, the larger groups of agents are used more efficiently and allows to address larger-scale problems [YKO03].

2.3.6.2 Holonic Agent Organization

The concept of holonic was first introduced by Arthur Koestler, in his book "The Ghost In The Machine" [Koe68] where he made the following observation. i) To satisfy a continuously complex and changing needs, a simple system

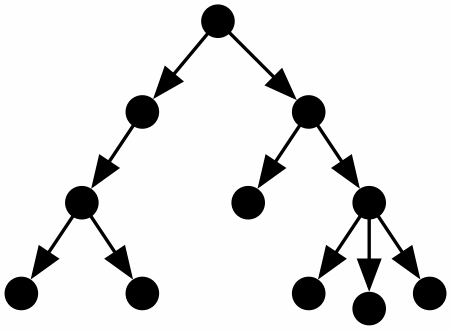


Figure 2.7: A hierarchical organization [HL04].

evolves and grows by creating a better and more capable version of itself. ii) in social organization and biological organisms, it is hard and confusing to make a distinction between wholes and parts in an absolute manner, because of an element can be at once an autonomous whole and an integrated section of a larger, more capable body.

The structure of These wholes is the basic unit of the holonic organization. They are named by Koestler "holons", , derived from the Greek word holos, meaning "whole", and on, meaning "part". Every Holon can be seen at the same time as a distinguishable entity made up of a collection of subordinates and as part of a larger entity. A Holon is described as being i) stable, ii) cooperative but most importantly is iii) autonomous [Tia07]. Figure 2.8 represents an example of a holonic organization where the directed edges represent hierarchical relationships, and holon boundaries are represented by circles.

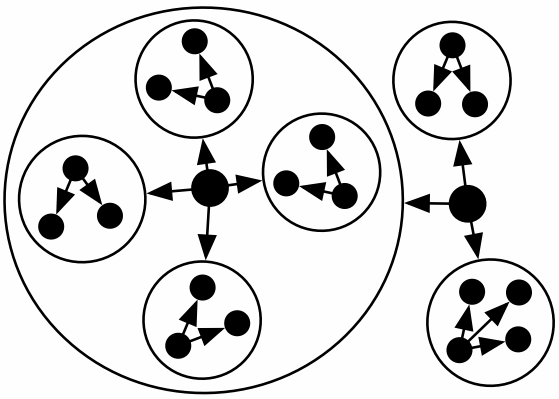


Figure 2.8: A holarchical organization [HL04].

2.3.6.3 Agent Federation

A variety of agent federation (federated systems) exist. However, all share the same concept of a group of agents who gives over more autonomy to a single delegate agent (called a facilitator, mediator, or broker) to represents the group [Gen97]. This type of organization is modeled in a similar fashion to the governmental federation system in which every regional province has some amount of local autonomy while operating under a single central government [DSW97; HCY99]. All interaction between members of the group and the outside world goes through the agent delegate as depicted in Figure 2.9. In that figure, every grouping of agents represents a federation and the agent delegate is designated as a white agent. The intermediary must have the capabilities to communicate and understand the members of its federation and other intermediary agents. Generally, this is done using a declarative communication language [Gen97].

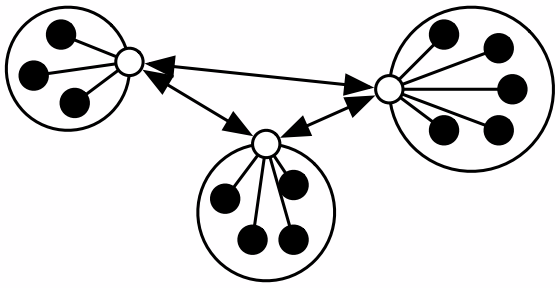


Figure 2.9: An agent federation [HL04].

2.3.6.4 Coalitions

The concept of the coalition has been used both in multi-agent systems and in real-world economic scenarios as it has been demonstrated to be a useful strategy. A coalition can be seen as a subset of a bigger set of agents population [HL04]. Generally, coalitions are: goal-directed, short-lived, i.e., a coalition appears to satisfy a need and disappear if that need no longer exists or can no longer satisfy the need built for or lost its population [HL04]. Multiple coalitions can be formed iteratively in response to a dynamic task en-

vironment [MW04]. They may form in populations of both cooperative and self-interested agents.

The organizational structure inside a coalition is typically flat Even though, there may be a "leading agent" to act as a representative and intermediary for the group as a whole [KG02]. Coalitions may be dealt with as a single, atomic entity once formed. It is possible to form a hierarchy of coalition by nesting one inside another. It is also possible to overlapping coalitions [SK98]. Figure 2.10 represents a population of agents organized into coalitions.

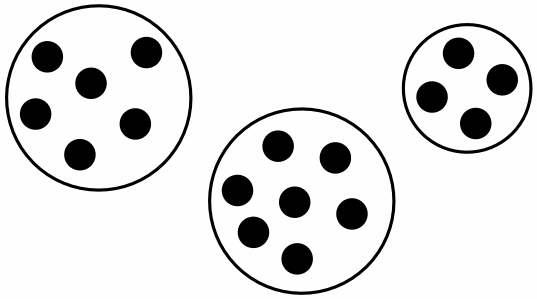


Figure 2.10: A coalition-based organization [HL04].

2.3.6.5 Teams

An agent team-based organization consists of a set of agents that have agreed to work together cooperatively to achieve a global goal [Tam97; BH01].

In contrast to coalitions, teams prioritize the overall objective (goal) of the organization rather than that of the individual members [HL04]. Consequently, agents' actions coordinating together must be aligned together and in favor of the team's goal. Inside a team-based organization, the interaction pattern can be entirely random, as depicted in Figure 2.11. Yet, generally, each agent will enact as many roles as required to achieve the team's goal. Those roles are subject to planned or unplanned events, whilst, the global goal itself continues to be the same. The primary benefit of this organization style is, a larger problem can be addressed by a group of agents than a single agent [GS88].

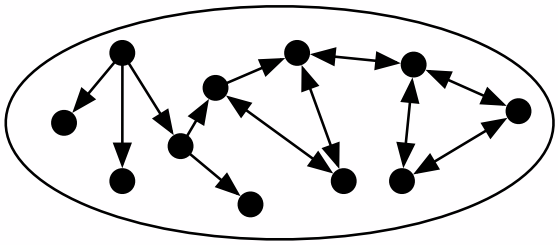


Figure 2.11: A team-based organization [HL04].

2.3.6.6 Congregation

The congregations-based organization is similar to teams and coalitions where a set of individuals agents are grouped together (in order to derive additional benefits) into a typically flat organization. In contrast to these other paradigms, congregations i) are long-lived ii) are shaped based on complementary or similar characteristics to ease the task of finding convenient collaborators. Figure 2.12 depicts an example of congregations. In this figure, the potentially heterogeneous goal behind each grouping is represented with different shadings. In contrast to coalitions (Figure 2.10) where it is typically more homogeneous. What drives the need to congregate is not the single or fixed goal of agents but, their set of stable capabilities or requirements [BDA00; Gri03].

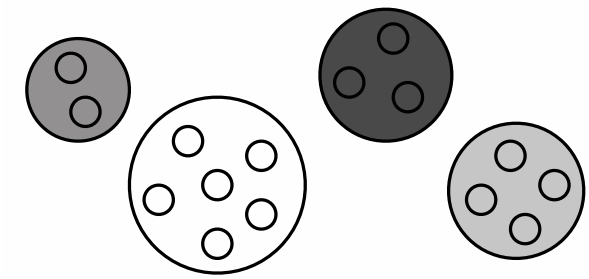


Figure 2.12: Congregations of agents [HL04].

2.3.7 Organizational Change Motivation

Generally, an organizational change is the result of the application of forces originating from different sources, which can be classified as i) internal forces, ii) external forces. The former concerns the environment where the organi-

zation resides. Here the source forces are the other element that populates the same environment, such as other organizations that may be competing together. The latter concerns the organization itself. Here the sources of forces are the internal elements of the organization, such as agents populating it. The impact of these forces on an organization depends on its vulnerability. For instance, an organization that has diffuse objectives is more vulnerable which means more aptitude to change. In the following, we will detail each type.

2.3.7.1 External Forces

It can be defined as the results of a change in the environment that may push for a change inside an organization. Many external forces can be found, of which we mention the following:

- **Market Forces**:
  Client (internal and external agents) demand for an organization's product and service change over time. This number of demand determines if products and services are enough required, which impacts the way an organization produces products or offers services. If there is no interest in what an organization produces, there is no reason for its existence unless it adapts [Ald08].

- **Technological Changes**:
  An organization can enhance its competitive position by improving its productivity through the adoption of new technologies. However, adopting new technologies over the old ones implies a cost for the organization. The managers have to decide about it considering its pro and cons for the organization [BC95].

- **Demographical Features**:
  Agents populating an organization are diverse and heterogeneous. Hence an organization has to control this heterogeneity effectively. On the one hand, by fulfilling the needs of agents .and on the other hand, by avoiding the malicious behavior [MT03]. For instance, by assigning determined roles to these malicious agents, which can limit their actions.

- **Laws and Regulations**:
  External laws might affect the environment of an organization or its neighbor organizations [BC95]. It can force the organization to change its objective, such as the produced product for a manufacturing system.

2.3.7.2 Internal forces

The internal forces are signals originating from the organization's inside itself. They represent signs of a required change inside the organization. In order to perform the required change in the most appropriate form and moment, it is essential to identify these forces and monitor them. Many internal forces exist, of which we mention the following:

- **Growth**:
  An organization grows (in either members or budget) to a point where it will become a requirement to modify its structure to a more hierarchical and bureaucrat organization with specialization of its members [Ald08]. A solution to this force is by decomposing it into smaller organizations.

- **Goal Succession**:
  An organization exists for a reason. Hence, two situations may arise if it achieves its overall objectives; i) The organization disappears, ii) The organization changes its strategy and start to fulfill a new overall objective, which allows it to continue with its existence [Ald08].

- **Crisis**:
  A drop in the efficiency of an organization may put it in a crisis. a potential remedy to such a situation is a profound organizational change in which its structural and/or functional elements are modified depending on the organization's specific needs.

2.3.8 Reorganization

Organization is considered a factor to achieve stability. However, organizations can be subject to different types of force that destabilize them (as stated

in section 2.3.7), which drive the organization to adapt. This adaptation in the context of OCMAS is called reorganization [Pic+09]. Such a mechanism allows an organization to be flexible to face the changing forces to come back to its stable state after being destabilized. Hence, reorganization is the answer to changes in the environment. It gives the organization the ability to do something other than that which was originally intended to survive [DD14].

The reorganization mechanism has two aspects; i) temporal and ii) intentional. For the temporal aspect, it is called *"proactive"* when preparing for an unpredictable future change in advance, and *"reactive"* when an adjustment is made after the occurrence of an event. For the intentional aspect, it is *"offensive"* if the organization's objective is to gain a competitive advantage and *"defensive"* if the objective of the organization is just to survive. These aspects form the Ws of reorganization, which are used to evaluate a reorganization decision. They are as follows:

- **What**: Which aspect of an organization to be reorganized. It can be behavioral or structural (see section 2.3.8.1).
- **Who**: Authority for the decision-making of the reorganization. It can be directive (role-based decision making) or collaborative (consensus-based decision making).
- **When**: Is it proactive or reactive?
- **Why**: What is the strategic reason that drives the reorganization? It can be offensive or defensive.
- **Whether**: It allows defining the reorganization threshold (how likely the reorganization is beneficial). A higher threshold means stability is more desirable than flexibility and vice versa.

In the following subsection, we detail the aspect of reorganization when considering what to change in an organization, and then we present a generic process for the reorganization and the types of changing process.

2.3.8.1 Aspects of Reorganization

Two aspects of reorganization can be identified if we consider what to change in the organization: i) behavioral (dynamical) and ii) structural [Dig09]. In

the behavioral one, the structure of the system stays unchanged, whereas the agents' state changes. Structural changes are what affect the structural elements of the system such as roles. In the rest of this section, we detail these two types [DSD04].

1. **Behavioral Changes**: Change at this level concerns the alteration in the behavioral state of the agent itself and the roles enactments inside the organization such as when agents join or leave the organization, when they change between existing roles, or when they upgrade or downgrade their capabilities or when they fail or succeed in fulfilling a role. Some changes require an assessment before they are accepted, such as when an agent joins the organization, it must agree to the term of the organization or it must be evaluated if it can fit inside the organization. For an agent that tries to enact a role, it must be verified if it has the required capabilities to play the desired role.

2. **Structural Changes**: This type of change concerns the structural elements of an organization. However, it can also influence the behavior of the current and future organization society. Its purpose is to accommodate long-term changes such as new goals or situations. For example, to stay competitive, a manufacturing system sets new goals to shift its production toward new products to accommodate customers' needs. In other cases, the old goals are modified to fulfill customer requests such as using different materials to produce the same product. Goals can also be deleted if they are no longer beneficial to the organization stability.

2.3.8.2 Generic (Re)Organisation Process

As we stated in the introduction, an organization is subject to the effect of its environment. A change in this later can have a negative or a positive effect on the organization, which may result in failure to achieve its purposes of existence. Avoiding such a situation requires an organization to undergo changes. These latter are generally a process that is composed of two phases: monitoring and reparation. In the remainder of this section, we detail this process [Pic+09].

1) **Monitoring Phase**:
   Monitoring means observing and recording the system's external and internal elements continuously and gathering information that is of significance to the organization about the different aspects of these elements. When processed, it allows the detection of problems. The monitoring can be done at the agent level or at the structure of the organization itself. The reported information about the different situations is then used in taking the right action to rectify the problem.
2) **Repairing Phase**:
   This phase is a response to the reported problem by the monitoring phase. It aims to find back the normal state of a system at run-time as optimal as possible. This phase itself can be seen as a process decomposed of three main steps which are:
   (a) Design: In this phase, a set of potential substitutes for the present organization is defined and developed.
   (b) Selection: In this phase, one alternative to the current organization is select to be applied. The selection is based on criteria that determine the best alternative. One important question is how to define the best criteria.
   (c) Execution: In this phase, the select alternative is applied.

2.3.8.3 Types of Changing Processes

Three types of changing processes exist, Namely Predefined, Controlled, and Emergent [Pic+09]. In our work, we are interested in the first two that can occur in the OCMAS type of MAS design. The rest of this section presents a brief description of these types [Pic+09] :

- **Predefined**:
  This case is characterized by prior planning of changes by the designer at design time. Also, the designer specifies the precise moment of application. The execution of this process of the adaptation is simple and clear. An external entity or the agents themselves perform the monitoring.

- **Controlled**:

  This case is characterized by the prior knowledge of the designer of the condition for triggering the change but not of when or how the organization should be changed. Here, the change process is carried out according to a known procedure. In this case, the designer defines monitoring and repair strategies for the organization. If the monitoring phase identifies an undesirable situation, the design phase provides (predefined organizations or created on-demand) a set of the possible alternative organization; then, the selection phase selects the most appropriate one to be executed by the execution phase.

- **Emergent**:

  This process concerns the emergent type of organization. In contrast to the other change process types, the designer does not know global strategies to monitor (time to trigger the process) and repair the organization. Local entities (at the local level of agent) of the system lead the change. In our work, we are interested in the first and the send type of process change.

2.4 Conclusion

In this chapter, we presented the main concepts related to our work. First, we started by defining the basic concepts of graph transformation, such as graph and graph morphism, and then, we detailed the concepts of type graph and transformation rule. Second, we presented the multi-agent system in which we define what an agent and organization is as well as the definition of organization and reorganization process.

CHAPTER 3

Literature Review

3.1 Introduction

In this chapter, we present a state of the art of modeling approaches of agent organization and reorganization. We have classified these approaches into two main categories : semi-formal, formal. In Section 3.2, we present the first category where most of the approaches are based on UML (semi-formal language), which is used as a tool to describe their different aspects. The second category can be divided into two sub-categories: works that use graph transformation (in Section 3.3) and works that use formal notations like logic notation (in Section 3.4).

3.2 Semi-Formal Approaches

Several approaches are proposed in this category. We concentrate on the works that are the most related to our solution.

3.2.1 GORMAS Approach

The authors in [ABJ11] proposed a methodological guideline for MAS modeling based on the Organization Theory and the Service-Oriented approach. This methodological guideline is called GORMAS (Guidelines for ORganizational Multi-Agent Systems). They define a set of activities for the analysis and design of virtual organizations. They also provide a way to model organizational structure and behavior. With this method, the services that are provided and required by this virtual organization are clearly defined.

As illustrated in Figure 3.1, GORMAS proposes to follow a basic sequence-guideline of organizational design, which allows to be integrated in a complete software development process, covering the phases of analysis, design, implementation, installation, and maintenance of the MAS.

To obtain the organizational model, the designer has to follow these steps:

1. *Mission analysis*: it implies the analysis of the system requirements, identification of use cases, stakeholders, and global goals of the system. This

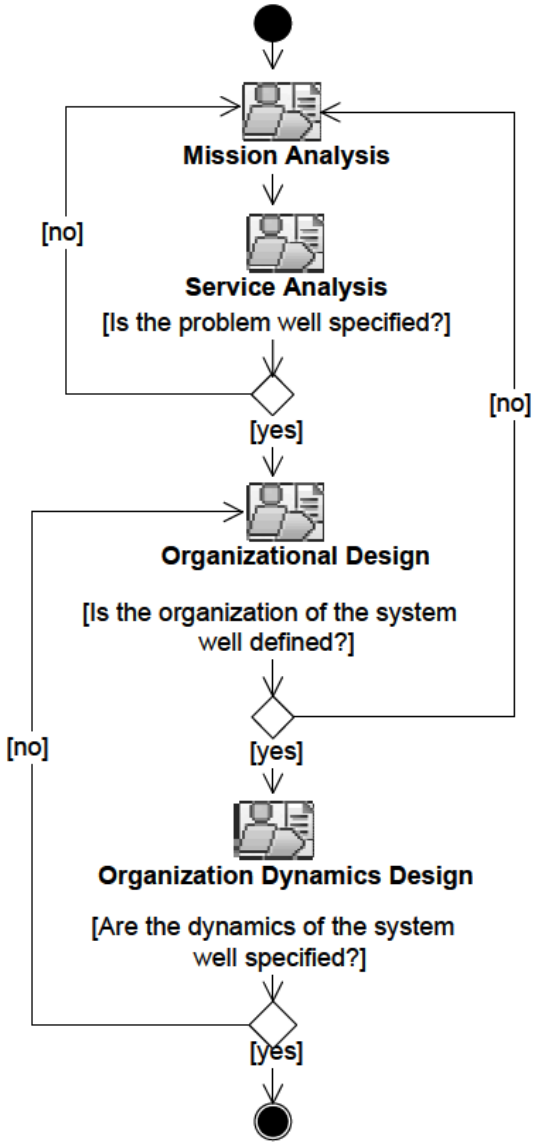


Figure 3.1: GORMAS Activity Diagram [ABJ11].

step answers to the questions: why we create the organization; what are the expected results; and which environment the organization should be located, what are the offered products and/or services.

2. *Service analysis*: in this step, an analysis of the offered services is performed and their requirements and associated processes. In addition, we precise the objectives and tasks that are associated to these services.

3. *Organizational design*: in this step, the most appropriate organizational structure is selected. Organizational models are used to describe roles, interactions that are related to the structure.

4. *Organizational dynamics design*: it identifies interaction for each service. QoS contracts are also specified. Moreover, it quantifies and evaluates tasks and activities in order to see if the system goals are achieved are established.

As we can see in Figure 3.1, the design phase is divided into two other phases: design of the organizational structure and design of the organizational dynamics. The positive point in their approach is that the development process is iterative. Indeed, we can return to a previous phase from any step in the process. This is important from the maintenance point of view. GARMAS is also interesting from the point that it models the dynamics. In contrast to our approach, the dynamic aspect in GORMAS is modeled only at the design time and cannot be changed at run-time, which is not the case in our approach.

3.2.2 OMACS Framework

Scott A. DeLoach in [DeL09] proposed a framework for Adaptive, Complex Systems called Organization Model for Adaptive Computational Systems (OMACS). The framework allows the system to design its own organization at run-time. The key component of the framework is a model (depicted in Figure 3.2) that allows to reorganize the system at run-time thanks to the defined knowledge in this model. This knowledge is about a system's structure and capabilities. The OMACS model allows being applied on a variety of systems thanks to a set of supported methodologies, techniques, and architectures.

Formally OMACS defines an organization as a tuple:
$O = (G, R, A, C, \Phi, P, \sigma, oaf, achieves, requires, possesses)$ where:

- G represents the set of organization's goals.
- R represents the set of roles.
- A: represents the set of agents.
- C: represents the set of capabilities.
- Φ : is a function that defines a relation over $G \times R \times A$. It represents the current set of agent, role, and goal assignments.

– P : is a set of constraints on Φ.
– σ : domain model used to specify environment objects and relationships.
– oaf : function $P(G \times R \times A) \rightarrow [0, \infty]$ defining quality of a proposed assignment set.

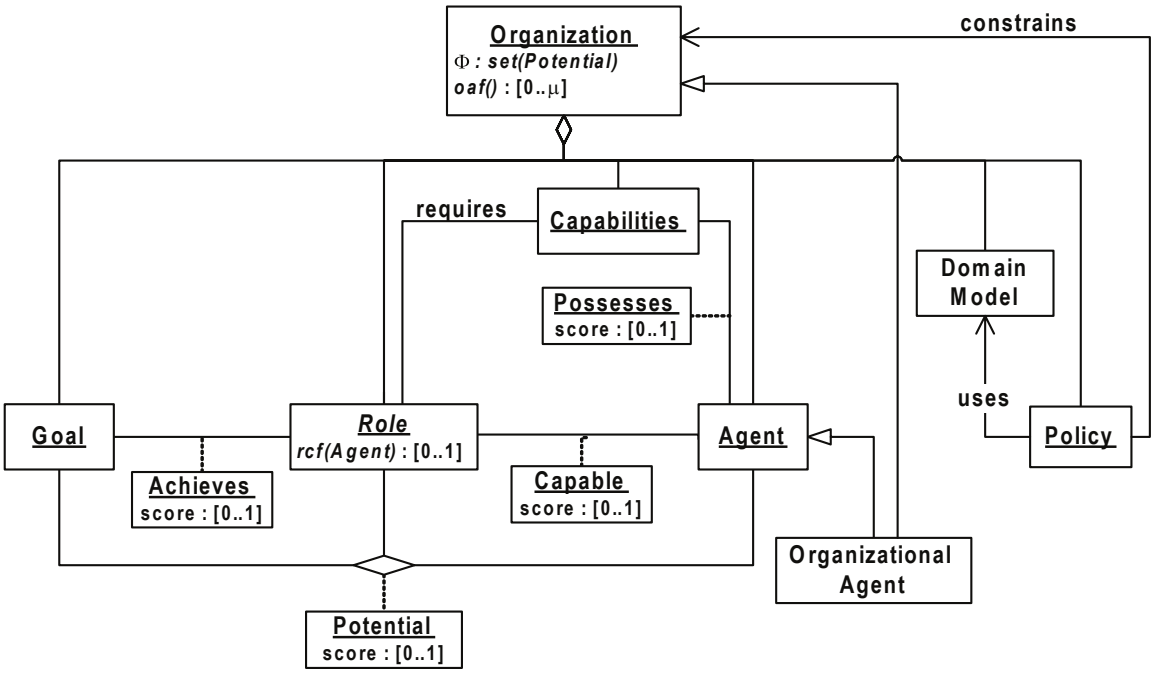


Figure 3.2: OMACS model [DeL09].

Besides, OMACS is related to the Organization-based Multiagent Systems Engineering (O-MaSE) methodology [Gar+07]. O-MaSE allows to create processes during the development of OMACS-based systems. The goal of the O-MaSE methodology is to allow process engineers to customize the construction of agent-oriented processes. It is based on a meta-model (in Figure 3.3), a set of method fragments, and a set of guidelines.

The meta-model (in Figure 3.3) defines the main concepts used in O-MaSE to design MAS organization. The O-MaSE meta-model extends the OMACS meta-model [DeL09] (featuring elements like Organization, Agent, Role, Goal or Domain Model) by adding new elements like Protocols or Environmental objects and properties.

Regarding method fragments, O-MaSE defines three main activities: i) requirements engineering, ii) analysis, and iii) design. The requirement engineering activity translates the system requirements into system-level goals.

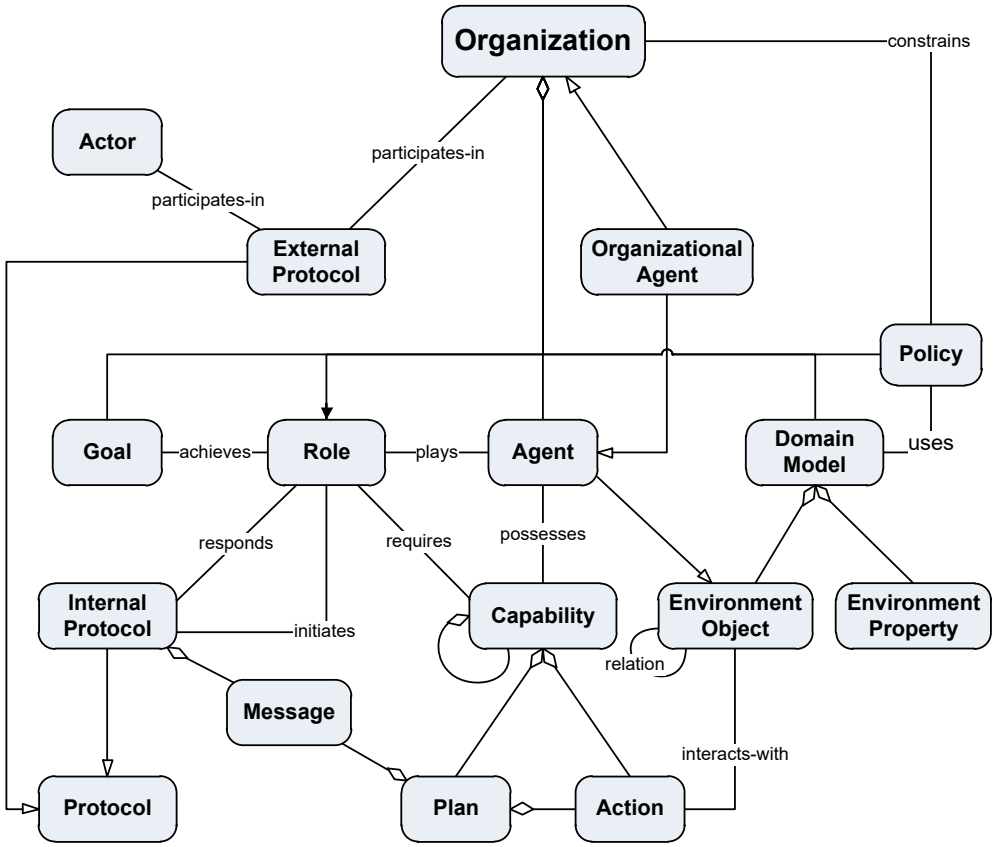


Figure 3.3: O-MaSE meta-model [DG14].

The analysis activity focuses on modeling the relationships between the organization and its environment. The design activity defines the entities that build the system, such as agents.

3.2.3 MOISE

Hannoun et al. proposed an organizational model for multi-agent systems called MOISE (Model of Organization for multI-agent SystEms) [Han+00]. MOISE model is structured into three levels: i) individual level: definition of the set of tasks that agent is responsible ii) aggregate level : allows to aggregate agents in a large structure and iii) society level: global structuring and interconnection of the agents and structures with each other.

The organization in MOISE is viewed as a set of normative rules that control the agents behaviors. MOISE looks for identifying the rights and duties of the agents inside a society from four points of view: structural, functional,

contextual, and normative.

The authors in [HSB02] propose the MOISE+ model which is an extension of MOISE. The main aspect of this extension is to clearly distinguish the structure, the functioning, and the *deontic* organizational aspects. The objective is to create an organization centered model. MOISE+ represents a good approach to organizational change, where new roles join the system to carry out the adaptation process.

MOISE [Han+00] and its extension MOISE+ [HSB02] are considered as the most popular methodologies for designing an OCMAS. Indeed, agents inside MOISE+ designed systems are organized following groups. When a reorganization process starts, a set of roles (the reorganization group) is created in order to carry out with this process. After that, reorganization scheme is created in response to the assignment of the roles from the reorganization group to agents.

3.3 Graph Transformation Based Approaches

Few works have used graph transformation for the formalization of system reorganization such as [WLZ06; MFC13; Buc+15; RGR15].

3.3.1 Multi-level graphs for System Reorganization

In [WLZ06], the authors presented a model based on graph transformation to describe the process of reorganization in the context of organizational structures. They consider the following aspects of organizations: the social structure (roles and their inter-relations), the coordination relations between agents and the role enactment for agents.

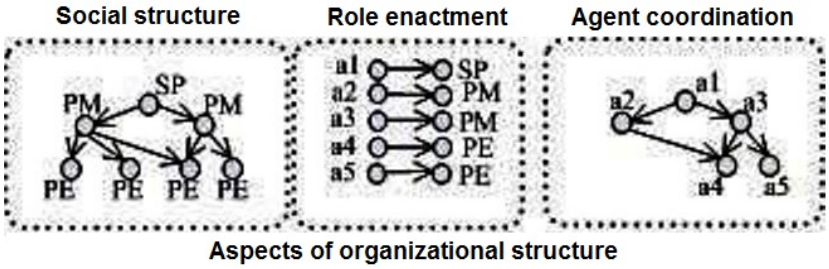


Figure 3.4: A diagram consisting of three aspects of an organizational structure [WLZ06].

For modeling the organizational structure elements, the above three aspects are described as a multi-level graph model (in Figure 3.4):

- the top-level: represents the role-graph that describes the social structure of an organization. The social structure corresponds to the certain goal hierarchies of the organization.
- the middle level: represents a connection graph for each agent which role is enacted. As shown in Figure 3.4, a1, a2, ..,a5 represent agents and SP, PM, and PE represent enacted roles.
- the bottom level: represents the agent-graph. It models the inter-relations between agents. In the agent-graph, the relations between agents are considered as the instantiation of the relations of roles.

Figure 3.5 depicts an example of the application of this approach.

Both role-graphs and agent-graph are labeled, directed acyclic graph and agent-graphs labeled, directed acyclic graph of the form:

$G = (N, E, s, t, l, m)$ where:

- N and E are two finite sets of nodes and edges with $N \cap E = \Phi$
- $s, t : E \rightarrow N$ are two functions mapping each edge to its source and target node respectively.
- $l : N \rightarrow \Sigma, m : E \rightarrow \delta$ are the node and edge labeling functions respectively.

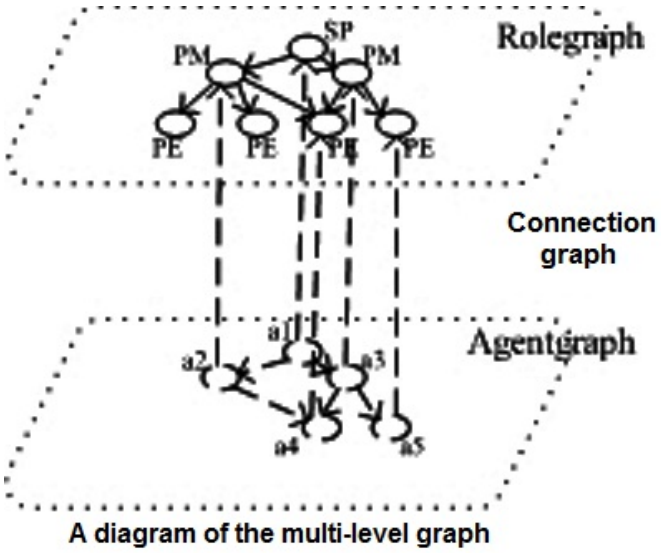


Figure 3.5: An exemplar of the multi-level graph [WLZ06].

3.3.2 A Model for MAS with Dynamic Organizations

The authors in [MFC13] introduce basic definitions that will be the basis of a framework for the specification of different levels of MAS using Graph Grammars. They are based on Population-Organization Model (PopOrg) (published in [DC96]), which is introduced as a minimal and formal model for a multi-agent system with dynamic organizations. The populational level of a multi-agent system is modeled by a set of agent graph grammars. Each agent graph grammars is defined by a graph grammars and a behavioral function. Their graph grammar is defined as follows: i) *a type graph*: it is considered as meta-model of agents and all its possible actions ii) *a set of rules* which define the behaviors of agents. A behavioral function is defined to control the application of the rules for each agent. All rules follow the schema depicted in Figure 3.6. iii) *initial graph*: represents the initial state of a given agent.

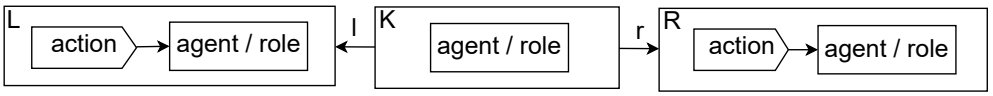


Figure 3.6: Rule schema for agents [MFC13].

The organizational level of a MAS has the same structure as the populational level. Only that, the organizational level gives a more abstract view of a system, where the agents are classified based on their roles

The organizational model is defined by a set of role graph grammars, which are defined as agent graph grammars. The interactions between pairs of roles are given by micro-links. Micro-links and link capability are analogous to exchange processes and exchange capacity, respectively. A graph grammar for the organization is given by a union of role graph grammars and the link capabilities of these roles.

Compared to our approach, their formal model is specific to the PopOrg model. Our approach is generic and can be applied to a variety of MASs, including PopOrg model.

3.3.3 Rule-Based Modeling and Static Analysis of Self-adaptive Systems

In [Buc+15], a typed attributed graph grammars based approach has been proposed to model and analyze self-adaptive systems. The system is modeled as a typed graph and its behavior is specified as a set of rules. The type graph proposed in their approach is related to a car Logistics System (see Figure 3.7). It contains types used for modeling the "normal" aspects of the car logistics scenario, as well as the "context" types used for adaptation.

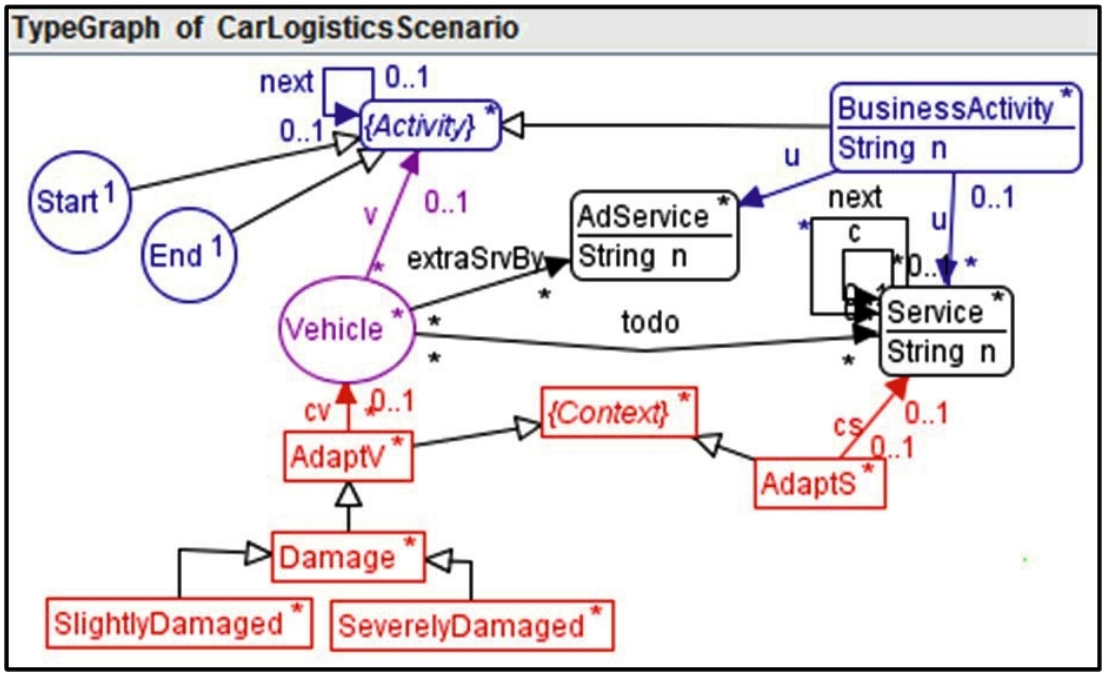


Figure 3.7: Type Graph of the Car Logistics [Buc+15]

They proposed three categories of graph transformation rules to describe the behavior of the system: i) *normal behavior rules*, ii) *context rules* applicable at any time to simulate unforeseen system changes by creating adaptation hooks, iii) *Adaptation rules* represent the adaptation performed in case of a change in the context.

The proposed graph grammar in [Buc+15] is used to model a specific system that is a "Car Logistic System". Their solution cannot be easily applied in another kind of systems such as Multi-Agent systems. They need to redefine the proposed type graph and all the rules, which is not the case in our approach. We have defined a generic approach for any kind of Multi-Agent system organization. In fact, we have modeled most of the concepts (such as Agent, Role, State, Goal, etc.) that can exist in a MAS.

3.4 Formal specification approaches for Multi-Agent Systems

Several approaches have been proposed to specify the Multi-agent System formally. We present in this section the works that are related (re)organization such as [Dig+05; DD14; KKS19; LMS17; BKC18; FK18; Mey14]

3.4.1 Formal Semantics Framework

Dignum et al. [Dig+05; DD14] proposed a theoretical framework to represent both organizational performance and the reorganization itself. They present a generic formal model to specify the MAS organizations and the organizational changes.

This approach is applied at design time. Reorganization consists of two activities: i) formalize the organization evolution and compare the actual state with the desired state. ii) specify the components of the reorganization strategies in order to trace a path to the desired state.

They define a function on the environment that allows to establish the cost of achievement of a given state of affairs, by giving the current state and the group of agents. The cost of reorganization plus the cost of achieving the new state is used to decide the strategy of reorganization.

Reorganization activities ir their approach can be classified in three groups:

- Staffing: update the set of agents, add a new agent, delete an existing agent.
- Structuring: alter the structure order of the organization.
- Strategy: update the objectives of the organization such as : add or delete a desired state.

3.4.2 Rewriting Logic for the Specification of MAS

Another approach has been proposed in [LMS17], which has as objective, specifying formally the elements of an organization (Agent, Group and Role). The authors proposed to transform organizational models given in "AgentUML" into Maude. In their approach, they simulate the system behavior which, allows to get feedback about the suitability of the solution. The proposed framework (depicted in Figure 3.8) is composed of several Maude modules:

- functional modules: describes actions that can be performed by an agent in order to go from a state to another. It is also used to define the basic concept of Agent-Group-Role model which role those agents can play within groups.
- object-oriented modules: Provide easy syntax for object-oriented rewrite theories. The basic concept of group is defined.
- timed object-oriented modules: Support object-oriented specification of real-time systems.

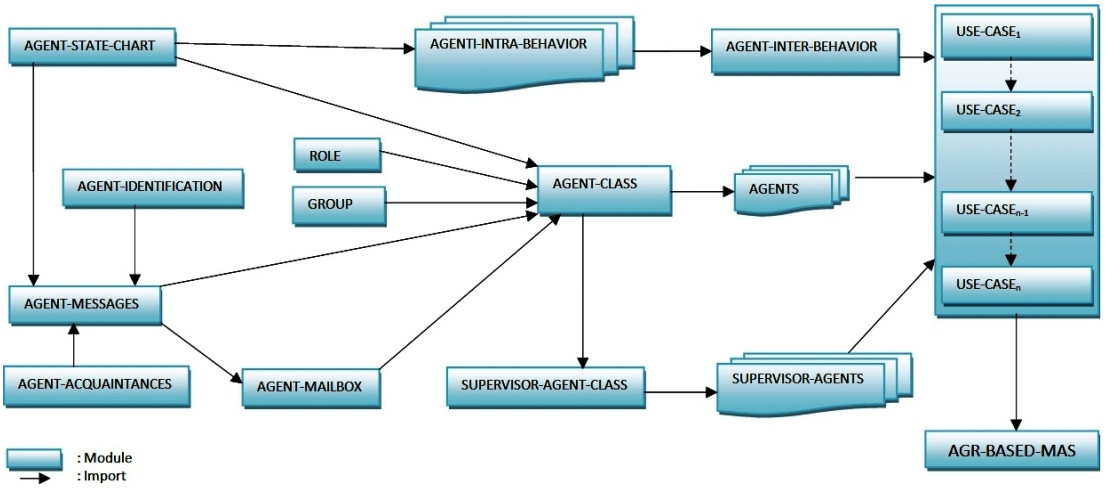


Figure 3.8: Formal framework modules [LMS17]

3.4.3 Automatic generating algorithm of rewriting logic for multi-agent system

In [BKC18], the authors proposed an algorithm to automate the generation (Figure 3.9) of rewriting logic specification for multi-agent system models. A Multi-agent system specified as a Petri net model is given as input to this algorithm. As a result, a Maude specification is generated as output which is used later to verify different proprieties of the candidate system.

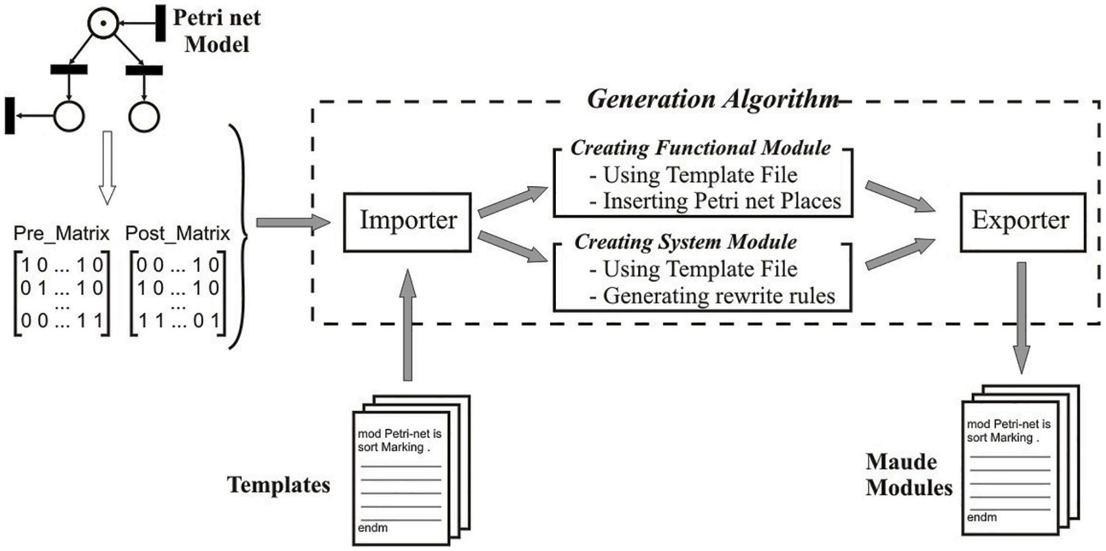


Figure 3.9: illustration of generating algorithm [BKC18].

3.5 Conclusion

This chapter sums up some of the existing works in the state-of-the-art in three main categories: semi-formal approaches which are based on UML language, Graph transformation-based approaches, and formal notations based approaches. The first category represents the easiest and intuitive choice. However, the lack of formal verification makes it prone to errors (made by a human designer). Our work can be classified in the second category as we also use graph transformation. In contrast to our work, the existing approaches in this category are in general more specific to one type of system.

In the third category, The Maude language and multi-modal logic that are

used in these works to describe a MAS can be considered an added difficulty to the learning curve. They require that the designer must be familiar with these notations. On the contrary to these works, in our approach, we use graphs and graph transformation to formally describe a Multi-Agent System. We assume that graphs are intuitive and easy to use. They do not require any additional effort from the designer.

Part II

Contributions

CHAPTER 4

Graph Transformation Approach for the Reorganization in Multi-Agent Systems

4.1 Introduction

This chapter covers the heart of this thesis, where we present our formal solution for specifying a multi-agent system (re)organization based on the graph transformation approach. First, in Section 4.2, we present an overview of the approach in which we present the components of our process. After that, we describe our MAS Organization Type Graph (in Section 4.3), which is considered as the static part. The MAS Organization Rules that we have proposed are presented in Section 4.4. It represents the dynamic part of our approach. Our formal definition of MAS Organization is presented in Section 4.5.

4.2 Approach Overview

Our approach consists of the formalization of the system organization and the reorganization process. First, we describe the MAS organization and the reorganization process as three main components (see Figure 4.1), namely i) MAS monitor, ii) MAS organization, iii) Reorganization Manager, where each one represents a particular concept of the MAS organization. The three components interact with each other in a specific manner. Second, the main objective of our approach is the use of graph transformation to formalize the three components: *"MAS Monitor"*, *"MAS Organization"*, and *"Reorganization Manager"*. As we mentioned above, graph transformation is suitable to represent complex systems, where the dynamic part is modeled as a set of rules, and the static part is modeled as a type graph. The *"MAS Organization"* component represents the static part of the system, which is formalized as a type graph. The *"MAS Monitor"* and *"Reorganization Manager"* components represent the dynamic part of the system. Hence, we formalize them as a set of rules called respectively: *"Monitor Rules"* and *"Reorganization Rules"*.

Before presenting our formalization as a type graph, we illustrate in the next sub-sections the role of each component in our process.

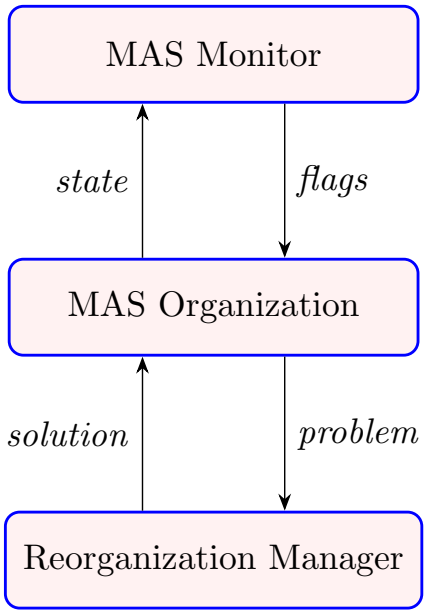


Figure 4.1: Approach Overview

4.2.1 MAS Monitor

Generally, the concept of "monitoring" means capturing properties of the environment, whether they are virtual or physical. Sensors (software or hardware) are the tools used to perform monitoring. The monitored properties in our approach can be related to:

- The capacity of agents: a change in the capacity of an agent can be of two types; degradation and augmentation. The first one can be caused by losing a capability, for example, by losing or damaging physical resources it possesses. In contrast to the first one, the second one can be caused by acquiring new capabilities, either physical such as new tools or mental knowledge. The intention of agents regarding leaving or staying in the organization.

- Agents intention: An agent is free to stay or leave the organization as it suits its objectives. Hence, monitoring its intention can prevent situations such as understaffing in the organization.

- Status of the system structure: we mean by the system's structure the sets of goals and roles defining the system organization. Monitoring

these properties means detecting the change occurring on these elements, whether positive (such as finishing a role, achieving a goal) or negative (such as a goal or role failure).

– Organization's environment: It can affect the system organization by creating new goals or terminate other to adapt to the environment change.

This component monitors the system continuously in order to identify any undesirable change. Indeed, it provides a set of flags describing different kinds of changes that could occur in the system. It is defined as a set of rules. By applying these rules, the *MAS Monitor* component marks the system to trigger the reorganization.

4.2.2 MAS Organization

This component describes the state of a MAS at a given time. In fact, the state of a MAS represents its condition, which is identified by the set of flags provided by the *MAS Monitor* component. Therefore, we have two types of system states: a *normal state* (a stable and functional state) and a *reorganizing state*. A reorganizing state may be composed of several reorganizing sub-states for a system with a significant number of Goals, Roles, and Agents.

4.2.3 Reorganization Manager

The main role of this component is to re-stabilize the system's state which is marked as unstable by the *MAS monitor* component. It specifies how the system should react in case of such an event. First, it assesses the system's undesirable state, which results in; i) the state does not require a change in the system, and the event is ignored. ii) the state requires a change in the system as a reorganization. If it is the second case, this component provides a list of actions to execute in order to bring the system back to its normal state. Essentially, this component is defined as a set of rules. The details of the formalization of these rules is presented below.

4.2.4 Basic Elements Life Cycle

The application of different rules provided by *MAS Monitor* or *Reorganization Manager* components can modify the state of the basic elements (Goal, Role, and Agent) that are used to represent the MAS organization. The different changes that can be applied to these elements should follow a well-defined set of state transitions. All these transitions for a given element allow what we call the element *life cycle*, which starts from the element's existence until its removal from the MAS organization. In the following, we detail the life cycle of each kind of element.

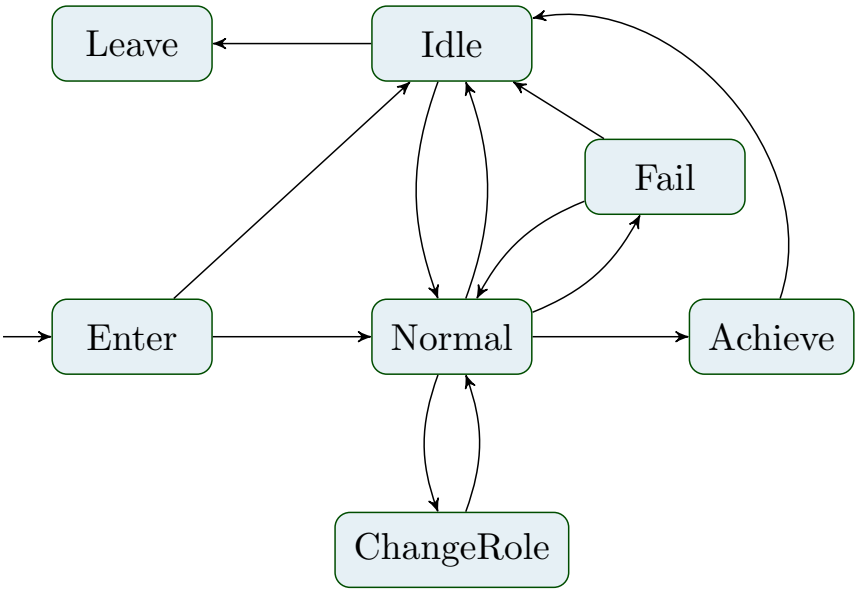


Figure 4.2: Agent Life Cycle

4.2.4.1 Agent Life Cycle

Figure 4.2 presents the life cycle of an agent in the system. This cycle can be summarized as three different phases:

- Agent entering the organization.
- Agent life within the organization.
- Agent leaving the organization.

In the following, we will detail each phase.

1. ***Agent Entering the Organization***

   An agent entering an organization must assess and answer several questions before choosing to enter the organization and play a role within it. Firstly it should consider the reasons for its entering and what it will gain from entering the organization. What are the resources that are allowed for it to access? Moreover, what are the capabilities that it will lose and what the organization is expecting from it? After considering these questions, an agent's life in the organization begins with their entry into the system.

2. ***Agent Inside the Organization***

   If it succeeds in enacting a role, it goes to a normal and functional state; else it goes to an "Idle" state. During its execution to fulfill the enacted role (when it is in "normal" state), the agent may go to a state of failure. In this case, it can go to an "Idle" state, so it would be possible to replace it with another agent.

   Besides, the designer [1] can wait [2] for the agent to return to its "normal" state and continue its role. The waiting is not always the right solution as it depends on the nature of the system. For instance, in a critical system, the waiting could put the system in a devastating state.

   When the agent is in a "normal" state and wants to change its newly enacted role, so, it should go first to a "ChangeRole" state. After that, it returns to its "normal" state, either accorded the wanted new role or not. When an agent is in an "Idle" state, two cases are possible, either it waits for a new role to be enacted, or it leaves the system. If the agent is in an "achieve" state, it can only go to an "Idle" state.

3. ***Agent Leaving the Organization***

   Different reasons may push the agent to leave an organization. Considering these reasons, it has to decide whether to leave or not. For example,

[1] Responsible for the management of the organization and reorganization
[2] The duration depends to the chosen policy or type of the system

if the agent has achieved the goal for which he entered in the first place, it will be very reasonable to want to leave. After deciding to leave the organization, the agent must first free itself from any obligation toward the organization. To do so, the agent must at first go to the state “Idle” and then leave the organization.

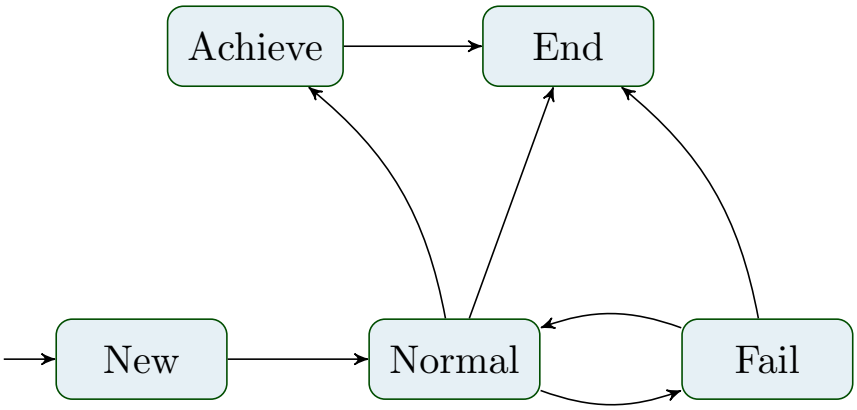


Figure 4.3: Role and Goal Life Cycle

4.2.4.2 Role and Goal Life Cycle:

Figure 4.3 shows the life cycle of goal and role elements in an organization. For each one of them, its life starts with its creation. After that, it goes to a “normal” state to be fulfilled.

While the system is running, if the goal (or role) is achieved, it changes its state to an "achieved" state and then to an “End” state. If it is not achieved, it is put in a "fail" state, where we have two choices:

- Wait until the failure is resolved, and the state is back to “normal”.
- End the goal as it is impossible to recover from this state, or the waiting cost is expensive. For example, when a manufacturing system tries to produce new pieces, and this system has not (cannot acquire in the future) knowledge to perform this goal.

4.3 MAS Organization Type Graph

We use the type graph to capture the structural aspect of the MAS organization. Figure 4.4 depicts our MAS organization type graph. This type graph contains a set of types used for modeling the *MAS Organization* elements, their states, and the triggering of the reorganization. Here, we detail each type in our type graph.

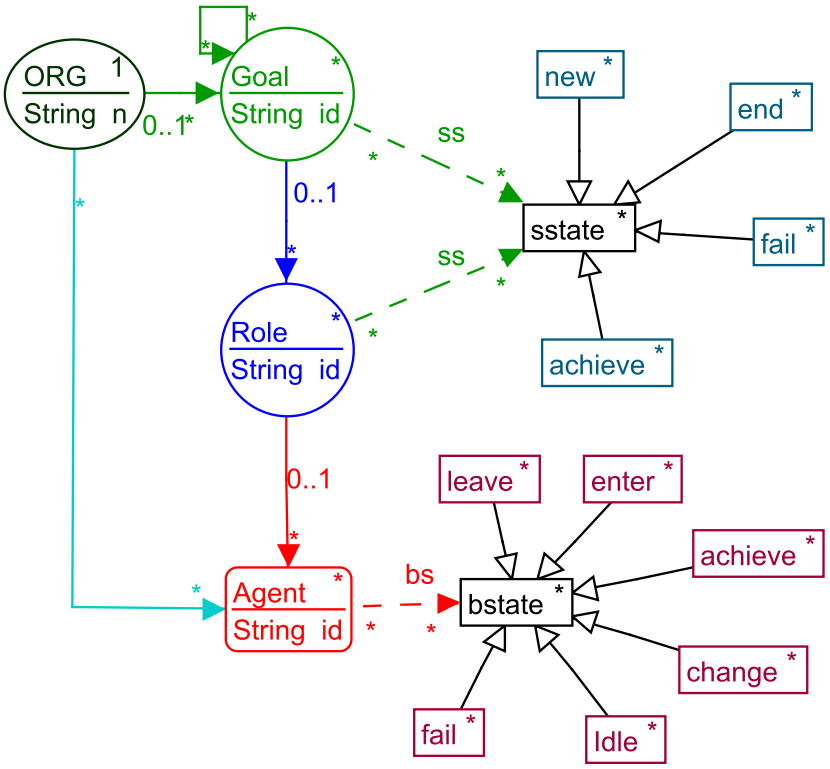


Figure 4.4: Type Graph.

4.3.1 ORG

It is a unique and main element (root) in a type graph representing a single MAS organization. All the other elements are attached directly or indirectly to this root. This element is an abstraction of the existence of a MAS Organization. The multiplicity of 1 to 1 is used to model the fact that this root element is unique. Each organization is characterized by a set of attributes which are explained in the following points:

- n: It is of string type. It holds the name of the organization.

- *Magent*, *Mrole*, *Mgoals*: These three attributes are of integer types. Each one of them holds a value that represents the maximum number of agents, roles, and goals an organization can have at an instant "t".

4.3.2 Goal

This node is an abstraction of an organization Goal (green circle in Figure 4.4). This node type can be connected to other nodes types using three types of edges, which are:

1. gg: represented by a green and continuous line. It can connect an ORG type to a goal representing the global goals of the organization, which can be of any number required by the organization. This property is expressed as a multiplicity of "*".

   Additionally, it can connect a goal to another goal with a multiplicity of "*". Hence a goal can be *solo* or be *composed* of several other sub-goals. This decomposition defines a hierarchy of levels of goals which has an important impact on how the organization progress.

2. gr: represented by a continuous blue line. It can connect a goal to a role with a multiplicity of "*", which means that a goal can have zero or multiple Roles. We note that even a parent goal divided into several other goals can also have roles.

3. ss: represented by a green dashed line. It can connect a goal to the type node *sstate*. This connection if it exists, it represents the current state of the connected goal; otherwise, it is in a normal state.

In our type graph, the control flow between Goals is expressed implicitly; this is because it represents only existing elements of an organization at runtime. Here we give some examples of such control flow. If we have two Goals g_1 and g_2:

(i) If g_1 and g_2 must be achieved sequentially, then g_1 must be created at first. If it is achieved, we can create g_2.

(ii) If there is a choice between g_1 and g_2. Only one goal must be created.

(iii) If g_1 and g_2 must be achieved together, they are created at the same level.

Moreover, different attributes are defined in the Goal type to represent different concepts such as priority. In the following, we present the most important:

- ***Id***: It is of string type, and it is used to identify each goal uniquely in the organization. All the identifiers of Goals are started by the letter "G".
- **P**: This attribute represents the concept of priority. It is an integer that takes its value in an Interval starting from the value "0" to a value defined by the designer. The smallest value represents the biggest priority. It allows the distinguishing between Goals by priority to pursue those that are more important than the others. It is only significant between Sub-Goals that are at the same level and attached to the same parent.
- **C**: The letter "C" stands for critical. It is a Boolean attribute that takes the values true or false. The value *true* means that it is critical, and if it fails, its parent Goal will automatically fail. The value false means that it is not critical, and its failure does not imply the failure of its parent goal.

4.3.3 Role

It is an abstraction of an organization's Role (the blue circle in Figure 4.4). This node type also can be connected to other nodes types using three types of edges (some of them are already mentioned before. Hence we explain the multiplicity only):

1. gr: It connects a goal to a role with a multiplicity of "0.1", which means that a role can only be connected to one goal.
2. ra: represented by a continuous red line. It can connect a role to an agent with a multiplicity of "*", which means that a role can have zero or multiple agents. The number of agents connected to a role determines the number of agents involved (they collaborate together) fulfilling this role.

3. ss: The same as in goal edges.

In addition, this node type has multiple attributes that represent different types of information. These attributes can be used in different situations, such as in the enactment of Roles. The most important ones are:

- ***Id***: It represents a unique identifier of a role. It is of string type. All identifier of Roles starts with the letter "R".
- ***P***: It represents the notion of priority between Roles, and it is of integer type. A Goal may be fulfilled by a different number of roles that may not be of the same priority. This attribute allows defining the biggest priority starting from the number "0" to the smallest priority defined by the designer.
- ***R***: It is used to store the set of required resources to fulfill a role. It is a list of strings where each string is a resource itself. An Agent that is trying to enact a role R_1 must possess all the required resources by R_1.
- ***T***: It holds an approximate amount of time that is required to complete the Role.
- min_A: It represents the minimal number of agents required to fulfill a role. It is of integer type. A number of agents below the minimum can still enact this role; however, it can not be achieved without all the minimal required number of agents. By default, this number is "1" and can't be "0".
- max_A: As opposed to min_A this attribute defines the maximum number of agents required to fulfill a role. An agent cannot enact a role where the number of agents already enacted this role is greater or equal to this number.
- ***C***: This attribute defines the importance of a role compared to other roles fulfilling the same Goal. In many situations, we could find that a Goal may be fulfilled with a variety of roles. However, one role may be critical to the achievement of this goal while other roles are not and do not pose

a threat to the achievement of this goal in case of failure. A failed critical Role implies the failure of the fulfilled Goal. It should be noted that many failed non-critical Roles may also imply the failure of the fulfilled goal.

4.3.4 Agent

This node is an abstraction of an Agent (red rounded rectangle in Figure 4.4). It uses the following edges to connect to other node types:

1. ra: same as mentioned before; however, the multiplicity of its part is "0,1", which means that an agent can be connected to zero or one role to fulfill the role functionality.
2. bs: represented by a red dashed line. It can connect an agent to the type node bstate. This connection, if it exists, represents the current state of the connected agent; otherwise, it is in a normal state.
3. og: represented by a continuous cyan line. It connects an agent to the ORG node with a multiplicity of "*" that allows us to connect every available agent (not fulfilling a role) to the ORG node.

Similarly to the other types, an agent contains the following attributes:

- ***Id***: It represents a unique identifier of an Agent. It is of string type. By convention, all agents' identifiers are started with the letter "A".
- ***R***: Similarly to the attribute "R" of a Role, It stores the set of capabilities that an agent possesses. It is also a list of strings where each string is a capability itself.

4.3.5 Structural State (sstate)

It allows describing the state of the structural part of the system. It includes the following types:

– *"new"* to denote the creation of a new Role or Goal.

- *"end"* to denote that a Goal or a Role has been ended.
- *"fail"* to denote that a Goal or a Role has failed.
- *"achieve"* to denote that a Goal or a Role has been achieved.

4.3.6 Behavioral State (bstate)

It allows describing the state of the behavioral part of the system. It includes the following types:

- *"leave"* to denote an agent leaving the system. Therefore, to be removed from the organization.
- *"enter"* to denote that an agent is entering the system.
- *"change"* to denote that an agent wants to change its enacted role. It has an attribute named "IdRole" of type string used to denote the identifier of the role desired for the change.
- *"achieve"* to denote that an agent has achieved its enacted role.
- *"fail"* to denote that an agent has failed to achieve its role.
- *"Idle"* to denote that an agent is in an "Idle" state.

It is important to note that an element (Goal, Role, Agent) that is not connected to any of the sub-types of *sstate* and *bstate* is considered in a "normal" state.

4.3.7 Extension Mechanism

Our types (Goal, Role, and Agent) can be easily extended to provide further information or put a restriction on a certain operation. This extension takes the form of attributes. For instance, the type *Agent* has the attribute "capabilities" to determine its skills. The type *Role* has the attributes "min Agent", "max Agent" to determine the maximum and the minimum number of agents that collaborate together in order to achieve this Role. Also, it can have the attribute "required skills" to determine the minimum skills that an agent must have to enact a Role.

4.4 MAS Organization Rules

We model the dynamic part of the MAS organization as a set of graph transformation rules (see Section 4.2). Two types of rules exist; *Monitor Rules* (R_m) and *Reorganization Rules* (R_{reo}). A monitor rule is applied if a change (event) has occurred in the system at run-time (such as removing a goal from the system). A response to an event to reorganize the system occurs by applying the reorganization rule if it is necessary.

The idea here is to represent every possible event (according to the life cycle of the different elements of an organization) with a rule which marks the element subject to the event with the corresponding flag to trigger the required response. For example, for removing a goal, we mark the node "Goal" by connecting it to a node "end" (of type sstate).

Furthermore, an event that affects one node type can propagate to other types as well. This propagation can be ascending to the node parent until the global goal or descending to the children nodes. Consequently, there could be more than one rule doing the same thing. The only difference is that one represents the event, and the other represents a reaction to the event. The application of the rules according to the events occurring in the system makes the different elements of the organization (Role, Goal, and Agent) subject to transition from state to state. The more the states change, the bigger the number of rules involved in this change.

According to the type of change (structural or behavioral) that may occur in the system, two types of rules exist:

- Behavioral rule.
- Structural rule.

In the following, we present the main rules that can be applied to demonstrate the most pertinent states that can be taken by the system elements.

4.4.1 Behavioral Rules

The life cycle of agents inside an organization determines the behavior of the system. In the following, we present the most important rules related to different states and changes that may occur to an agent during its life, starting from its entering to its leaving.

4.4.1.1 Agent Entering the System

This rule represents an agent that desire to enter the system. It is modeled by the rule *AgentEnter* as in Figure 4.5. The application of this rule marks the agent with the flag "enter" (see Figure 4.5), which will trigger other rules to put it in an appropriate state according to the global state of the system (enact new role or put the Agent in an "Idle" state). This rule belongs to the *MAS monitor* component.

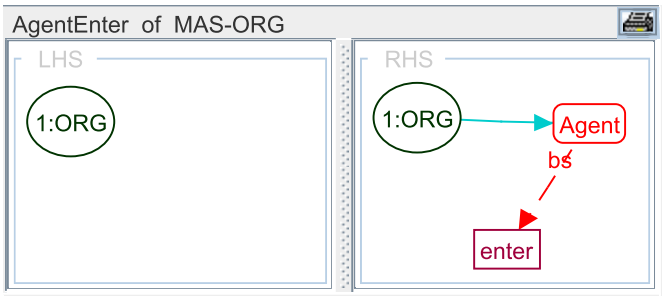


Figure 4.5: AgentEnter Rule

The designer can put an additional restriction on the agents entering the system to allow only those that have certain features or skills. For example, it is possible to restrict the set of agents to those with some capabilities that allow them to operate a certain type of machine.

Other restrictions can be applied to the way agents enter the organization. For example, in an organization where Agents must enter the system one by one, the designer can put a negative application condition represented as an agent with the flag "enter". It will restrict other agents from entering the system as long as there is an agent with the flag "enter" in the system.

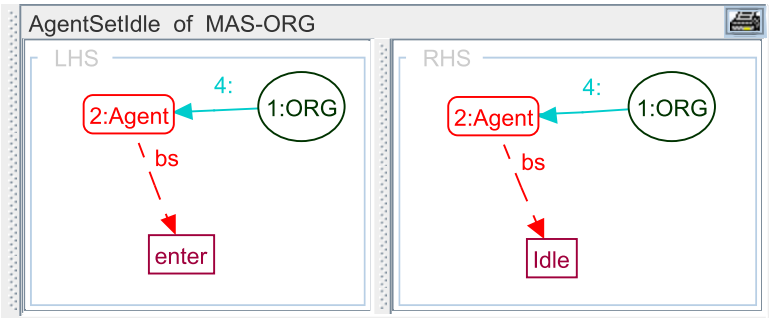


Figure 4.6: AgentSetIdle Rule

After entering the organization, an agent tries first to enact a role as described in Section 4.4.1.2. If the agent does not succeed, it is put in an Idle state by applying the rule "*AgentSetIdle* " (Figure 4.6)

4.4.1.2 Agent Enacting a Role

This event represents an agent enacting a Role in the system. Its state must be an "Idle" or an "enter" state to do so. Two rules that belong to the reorganization manager component are available to address the two states, which are:

- *AgentEnactRole_enter Rule*: In this rule, for each agent with the flag "enter", we create a link between the role and that agent. After that, we remove the flag "enter" and the direct link to the organization node (see Figure 4.7).

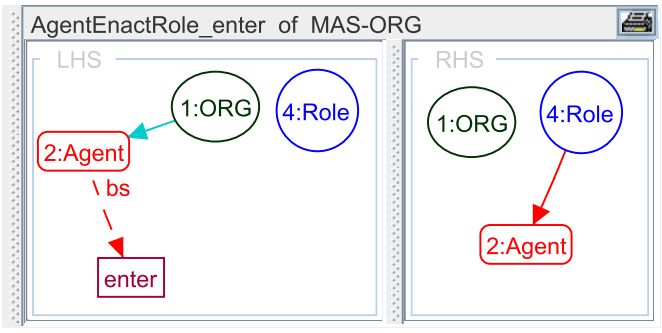


Figure 4.7: AgentEnactRole_enter Rule

- *AgentEnactRole_idle Rule*: In this rule, for each agent in an "Idle" state,

we act as the previous rule by deleting the flag "Idle" and creating a link between the role and that agent (see Figure 4.8).

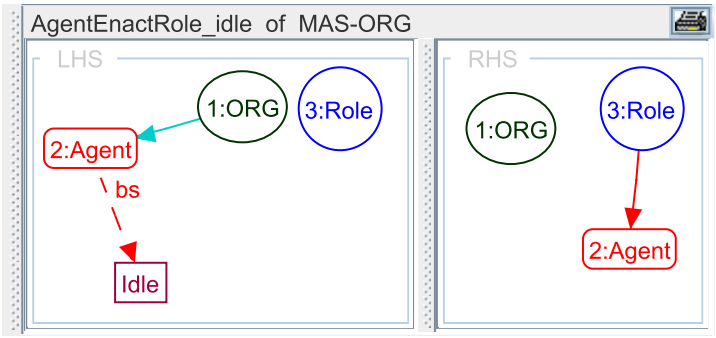


Figure 4.8: AgentEnactRole_idle Rule

The system designer defines the policy of enactment of Roles. This designer can define a priority for the two previous rules. This means, which agent can enact a role first, the ones that just entered the system or those in an Idle state. In other words, choose the ones that had already seen the system and experienced with its different elements or the ones that have no experience inside this organization. This choice is left to the designer who can perceive the significance of this experience for the benefit of the organization.

Furthermore, the designer can define conditions based on the agent capabilities and the role requirements (for instance, what fittest agent can play a given role?).

4.4.1.3 Agent Changing Role

This event represents an agent that desires to change its current Role in the system to another one.

During its life in the system, an agent may lose one or several of its capabilities or acquire other capabilities. These are some of the reasons that push the agent to change its role to upgrade to a more demanding one or downgrade to a less demanding one. Other reasons may relate to its objective itself in the system. Two situations may arise; i) the requested role is not enacted; ii) the requested role is already enacted to another agent.

The first situation poses no problem as the role is just given to the agent. However, the second one is treated as a conflict. The resolution of this conflict depends on the policy chosen by the designer at design time. We identify tow policies;

- Preemptive where the agent changes its role to the requested one by first de-enact it from the other agent (This is may only be possible if the first agent is more fit to the role than the second agent).
- Non-preemptive policy where the agent can change to a role that is not enacted by any other agent.

In the next, we present the rules that are involved in the non-preemptive policy

- *AgentChangeRole*: This rule (see Figure 4.9) models the event of an agent desiring to change its current Role. The negative application condition ensures that the agent is in a normal state. The application of this rule results in an agent attached to the flag "change". The identifier of the requested Role is stored in the attribute *IdRole* of the "change" node. This rule belongs to the *MAS monitor* component.

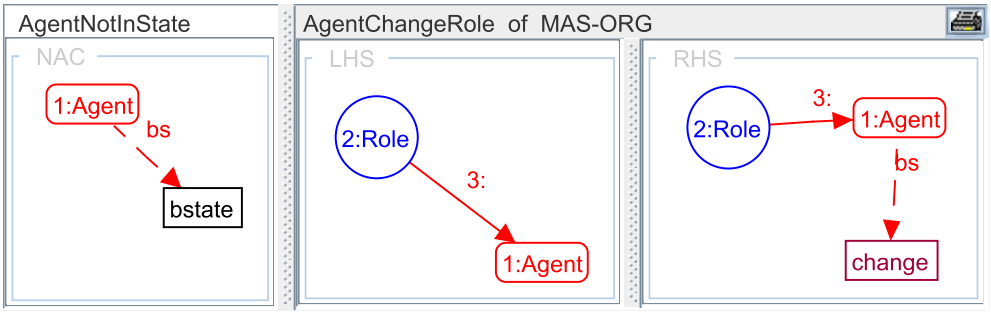


Figure 4.9: AgentChangeRole Rule

- *AgentChangeRole_freeRole*: This is the first rule to apply after triggering the event of changing the current Role. To be applied first, the requested role must be in a normal state, which means it is not connected to any state node. Second, a condition over the attribute context must be satisfied. This condition allows verifying that the agent possesses the re-

quired capabilities to fulfill the role and that the number of agents involved in the desired role did not reach the maximum. This rule belongs to the *reorganization manager* component. Figure 4.10 depicts this rule.

The application of this rule results in removing the node "change" connected to the agent.

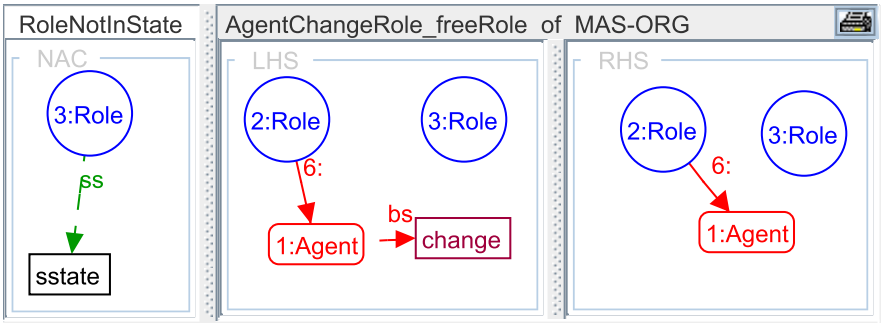


Figure 4.10: AgentChangeRole_freeRole Rule

- *AgentChangeRole_NoFreeRole*: This Rule (see Figure 4.11) is applicable only if the previous rule was not applied. It simply means that the system is denying the request of the agent to change its current role. It removes the node "change" linked to the agent without other changes. This rule belongs to the *reorganization manager* component.

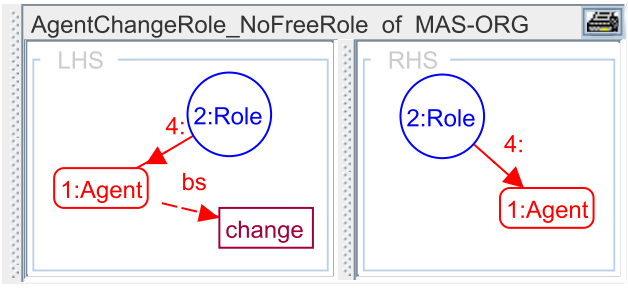


Figure 4.11: AgentChangeRole_NoFreeRole Rule

4.4.1.4 Agent Achieving Roles

After spending the required time and effort, the agent completes the assigned task. This event may be propagated to the goal parent. Hence, this event involves structural and behavioral rules. In this section, we present the second ones, and the others are presented in Section 4.4.2.2.

- *AgentAchieve*: This rule that belongs to the MAS monitor component simply marks the concerned Agent with the flag "achieve". It is only applied if the agent is in a normal state. This condition is expressed as the negative application condition "AgentNotInState" (see Figure 4.12).

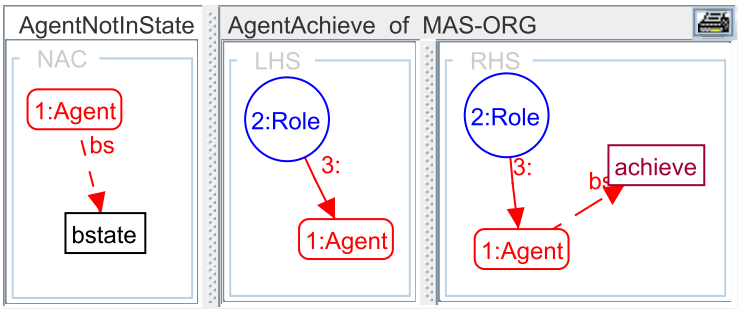


Figure 4.12: AgentAchieve Rule

- *AgentSetIdle_Achieve*: An agent that achieved its role must first go to an idle state before deciding what to do next(leave the organization or enact another Role). Hence, the application of this rule (see Figure 4.13) that belongs to the reorganization manager component. It removes the connection to the achieved role and the flag "achieve" from the agent and create a new connection to the ORG node and the flag "Idle".

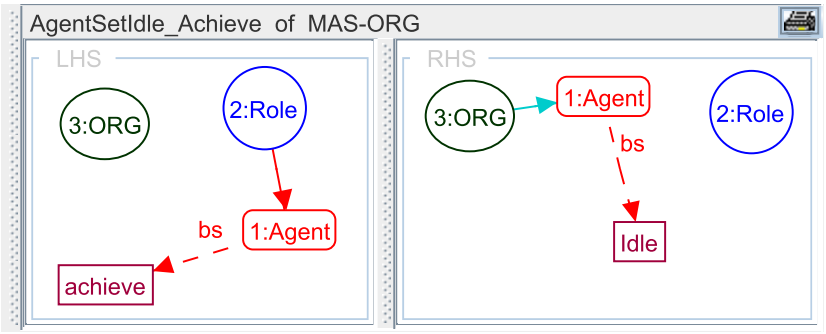


Figure 4.13: AgentSetIdle_Achieve Rule

4.4.1.5 Agent Leaving the System

For each agent that desires to leave the system (whatever its actual state "normal", "fail", or "achieve"), its state must be changed to an "Idle" state first. Afterward, it has the choice to stay or leave the system. We present here the

case of an agent in a "normal" state that wants to leave (other cases are presented in Section 4.4.1.6 and Section 4.4.2.1). In such a situation, the involved rules are:

- *AgentSetIdle_Normal Rule*: This rule (see Figure 4.14) models the event of an agent that is already fulfilling a role, and it is in a "normal" state. For some reason, this agent wants to stop playing this role. In this case, we put the agent in an "Idle" state by creating a new node "Idle" and removing the link between the agent and the role and creating a new link between the "ORG" node and the Agent. In addition, we create another link between the Agent and the new node "Idle".

  The fact that the agent must be in a "normal" state is modeled using a negative application condition (*AgentNotInstate* in Figure 4.14).

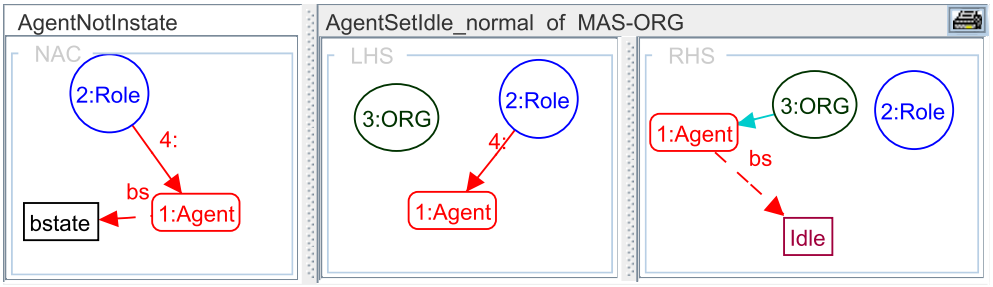


Figure 4.14: AgentSetIdle_Normal Rule

- *AgentLeave Rule*: This rule model the event of an agent wants to leave the system. So, as in Figure 4.15, we replace the "Idle" flag with the "leave" flag.

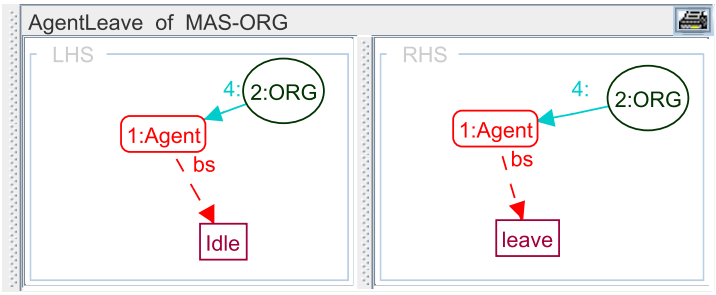


Figure 4.15: AgentLeave Rule

- *AgentLeave_remove Rule*: This rule model the leaving of an agent. In this case, we remove the node Agent and the flag "leave" (see Figure 4.16).

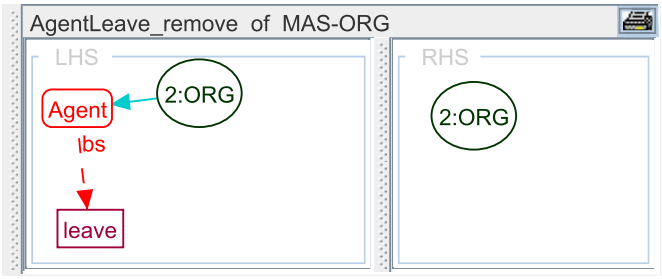


Figure 4.16: AgentLeave_remove Rule

4.4.1.6 Agent being failed

An agent may fail to complete the assigned role for different reasons shus as a lost capability. Agent failure may be permanent or temporary. If the failure is permanent, the agent is de-enacted the role and put to an Idle state. If the failure is not permanent, the policy defined as a response in such a case must answer to questions such as: is this role critical? How much time is required to recover? etc. In the following, we present the rules involved in case of permanent failure:

- *AgentFail*: This Rule (see Figure 4.17) marker the agent with the "fail" flag. However, to fail, the agent must be first in a normal state. This condition is represented as the negative application condition "Agent-NotinState".

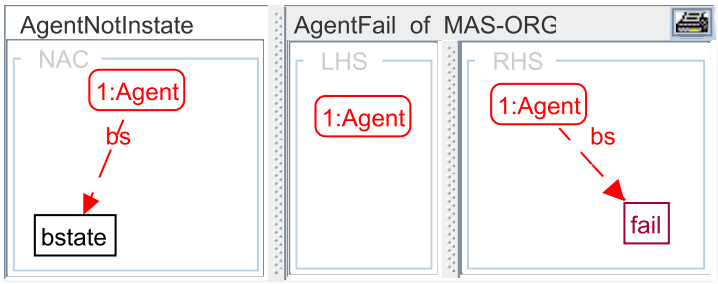


Figure 4.17: AgentFail Rule

- *RoleFail_FromAgentFail*: Similarly to rule *RoleAchieve_FromAgentAchieve*, this rule propagates the failure of an agent to the connected role. Other rules to reproduce the failure of roles to the parent nodes (sub-goals and goals).

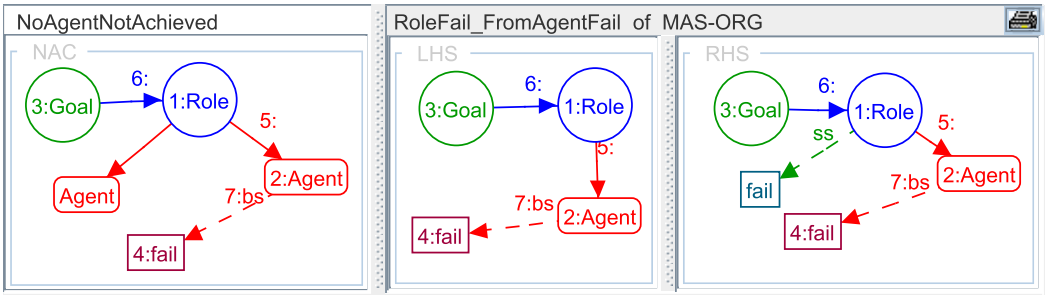


Figure 4.18: RoleFail_FromAgentFail Rule

- *AgentSetIdle_Fail*: This rule (see Figure 4.19) sets the failed agent to an Idle state in the case of unrecoverable failure.

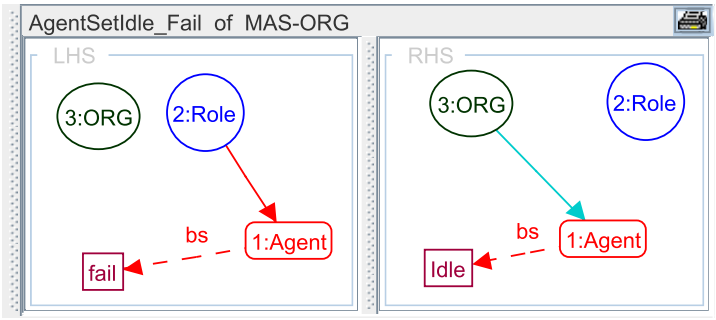


Figure 4.19: AgentSetIdle_Fail Rule

4.4.2 Structural Rule

Structural rules can modify an organization structure (Goals and Roles) in response to an event. Because Goals and Roles are of similar nature, many rules may seem the same with small modifications in the types of elements used. In the next sections, we will illustrate the most important rules and omit similar ones.

4.4.2.1 Goals Being Ended

A Goal can be ended for many reasons, which are related to the overall objective of the system. For example, ending the Goal of producing a piece because it is no longer in demand. For this, we have the following set of rules:

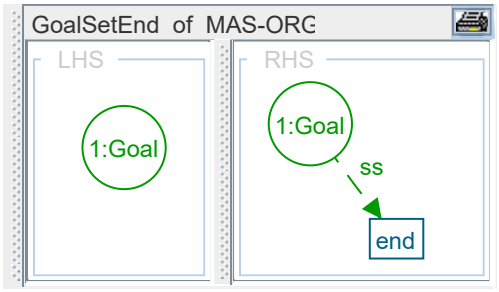


Figure 4.20: GoalSetEnd Rule

- *GoalSetEnd Rule*: In this rule, as shown in Figure 4.20, we mark the Goal to be removed from the system with the flag "end".

  As a response to this event, other rules should be applied to reorganize the system. We resume these rules as follows:

  - Mark all Goals in its hierarchy with the flag "end".
  - Mark all Roles that are attached to the ended goals by the flag "end".
  - Put all Agents that are attached to the ended roles in an "Idle" state.
  - Remove from the system all Goals and Roles that have the flag "end".

- *GoalSetEnd_SubGoal Rule*: This rule is applied for each sub-goals. It enables marking them by the "end" flag (see Figure 4.21).

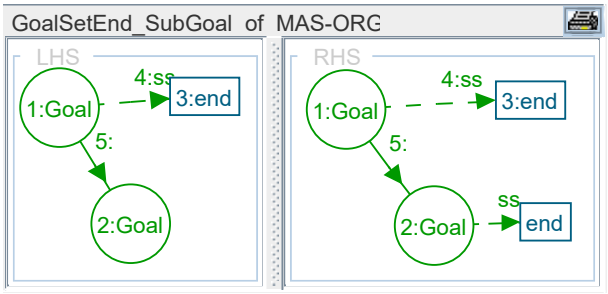


Figure 4.21: GoalSetEnd_SubGoal Rule

- *GoalSetEnd_Role Rule*: This rule enables marking a Role attached to an ended Goal by the flag "end" (Figure 4.22). Besides, a role can be ended without ending its parent goal. The reasons for such a situation are many like it is achieved and no longer required. In this case, we apply other rules.

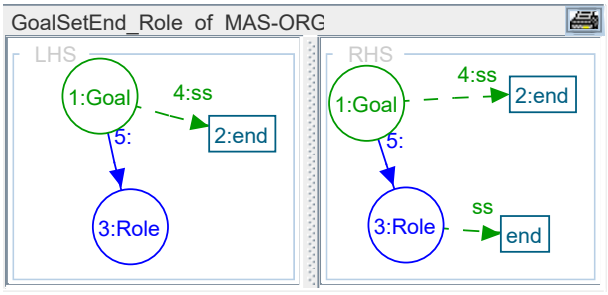


Figure 4.22: GoalSetEnd_Role Rule

- *RoleEnd_AgentIdle Rule*: This rule is applied in response to an ended Role event. It puts the agent in an "Idle" state when the Role is ended (see Figure 4.23).

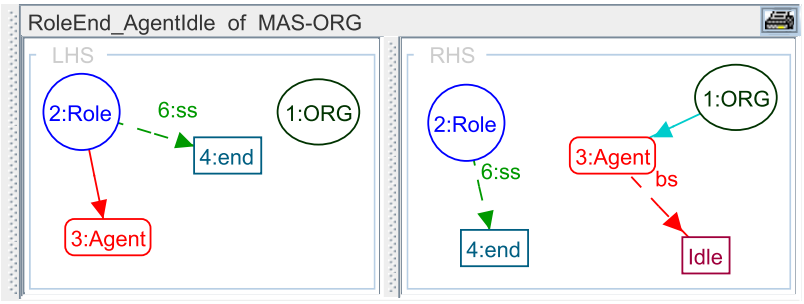


Figure 4.23: RoleEnd_AgentIdle Rule

- *GoalEnd_Remove Rule*: This rule (Figure 4.24) removes all the goals marked by the flag "end" from the system. For goals attached directly to the organization (the ended Goal is attached to the ORG node on the left-hand side), we apply another rule. Indeed, on the right-hand side, we maintain only the ORG node.

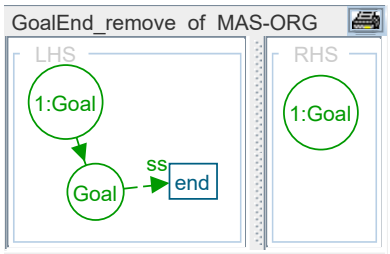


Figure 4.24: GoalEnd_Remove Rule

- *RoleEnd_Remove Rule*: In this rule (see Figure 4.25), each Role that is marked by the "end" flag is removed from the system,

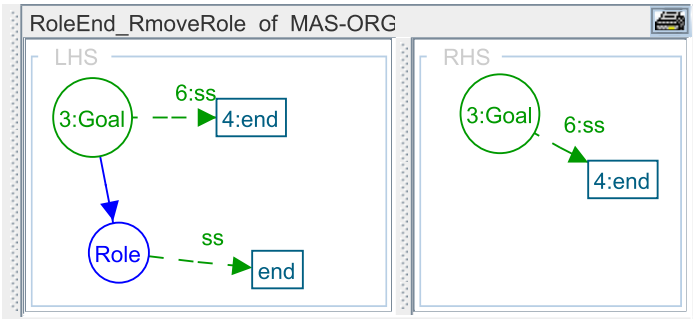


Figure 4.25: RoleEnd_RemoveRole Rule

4.4.2.2 Goals being Achieved

A goal is achieved in two ways:

- Down-Top: All roles of a Goal must first be achieved (or all the critical ones). After that, the achieved goal propagates its achievement to its parent.
- Top-Down: the goal is achieved directly, and then it propagates it to its children node to terminate their roles and free the agents fulfilling the roles.

The first way starts with an agent's event achieving a role as presented in Section 4.4.1.4, which triggers other rules that start to mark the parent role with the flag "achieve". As long as the condition holds for applying these rules, this process continues to the parent goal. The following rules describe this process:

- *RoleAchieve_FromAgentAchieve*: This rule (depicted in Figure 4.26) marks the role connected to the agent with the "achieve" flag. Because the role can have multiple agents fulfilling it, all agents connected to it must be in the state of "achieve". This condition is represented by the negative application condition "NoAgentNotAchieved". Besides, the NAC "RoleNotAchieved" (not shown in Figure) prevents marking an already achieved role with the flag "achieve".

  Othe rules exist similar to this one that propagates the "achieve" event of an agent from roles to sub-goals and goals.

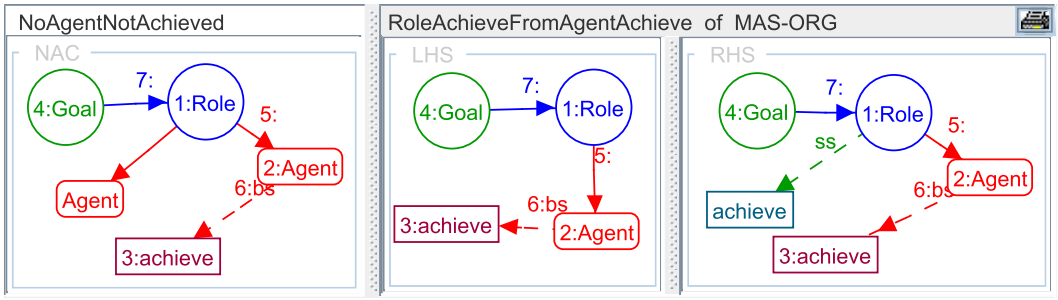


Figure 4.26: RoleAchieve_FromAgentAchieve Rule

- *GoalAchieve_FromRoleAchieve*: This rule (see Figure 4.27) allows marking the goal with the "achieve" flag. It represents the case where all the roles must be fulfilled to achieve a goal. As we did in the other rules, a negative application condition represents this condition. The next step after achieving a goal is to end it. This process is the same described in Section 4.4.2.1.

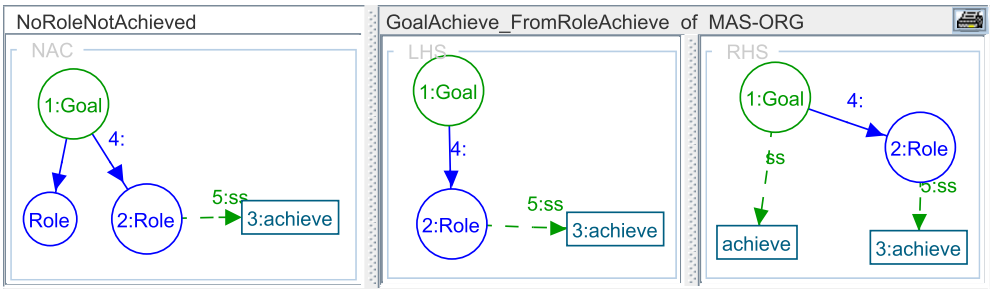


Figure 4.27: GoalAchieve_FromRoleAchieve Rule

Many other cases, such as achieving goals with sub-goals, have rules not presented here. However, they all follow the same process.

Unlike the first way, the second one is a top-down process. It begins with an event that marks a goal as achieved without fulfilling its roles. This event triggers further rules to terminate its roles and release the agent enacting them.

The rule *GoalAchieve* (see Figure 4.28) starts this process by connecting the goal to the achieve flag. After consuming this event in achieving the parent goal, if it exists, it is simply ended following the process described in Section 4.4.2.1.

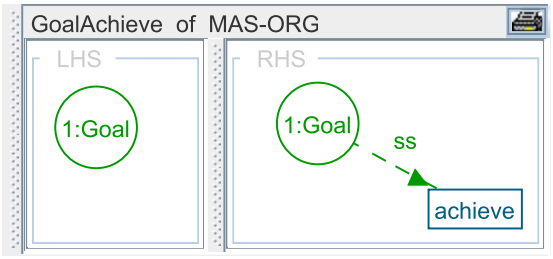


Figure 4.28: GoalAchieve Rule

4.4.2.3 Goals being Failed

Depending on the system's nature, we distinguish two types of failures: recoverable and unrecoverable failure.

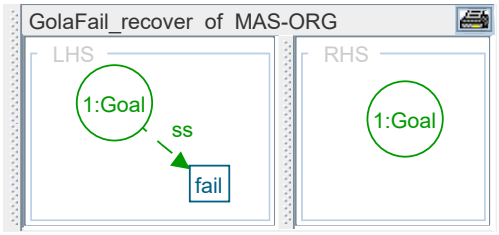


Figure 4.29: GoalFail_Recover Rule

The recoverable failure is represented in the life cycle of a goal as a connection from the "fail" state to the "normal" state. The rule that is applied here removes the flag "fail" from the Goal node (see Figure 4.29). In the unrecoverable failure, the "fail" state is connected to the "End" state. Here, the rule puts the Goal node in the "end" state.

4.5 Mathematical Notation of our MAS Organization

In Multi-Agent Systems, organizations are expected to be subject to various changes where their effect can be local, to a small part of the organization, or global for the whole system. Moreover, multiple small changes can occur simultaneously in multiple places of the organization. Therefore, multiple rules (monitor and reorganization rules) can be applied simultaneously at the same

level or at different levels of the hierarchy. Hence, different sets of rules must not interfere with each other. Therefore they have to be *confluent* and *terminating*. Also, rules have *priorities* that serve different purposes, such as defining the policy of enacting roles by agents. In this section, we provide a mathematical notation of our formal definition of the MAS organization.

Indeed, we define in an unambiguous way the MAS (re)organization using typed graph grammar as follow:

Definition 4.1.
A multi-agent system is given by a triple $MAS = (T_{mas}, G_i, R_{mas})$ *where:*

- T_{mas}*: is the type graph (it introduced in Section 4.3).*
- G_i*: is the initial graph. It represents an organization in its initial state.*
- R_{mas}*: represent the set of typed rules with negative application condition (it is presented in Section 4.4).*

In our approach, the rules that define the behaviors of the MAS are for monitoring or reorganization. Therefore, the R_{mas} can be defined as the union of the two sets R_m and R_{reo} as follow:

$$R_{mas} = R_m \cup R_{reo}$$

where,

- R_m represents the set of monitoring rules.
- R_{reo} represents the set of reorganization rules.

We have defined 61 rules for the monitor component and the reorganization manager that varies from utility rules to those that do the reorganization. We have presented in Section 4.4 the most pertinent rules. Hence R_m and R_{reo} contains only the presented rules in Section 4.4.

R_m = { *AgentEnter, AgentEnactRole, AgentChangeRole, AgentAchieve, AgentLeave, AgentFail, GoalSetEnd, GoalAchieve}*

R_{reo} = {*AgentSetIdle*, *AgentEnactRole_enter*, *AgentEnactRole_idle*, *AgentChangeRole_freeRole*, *AgentChangeRole_NoFreeRole*, *AgentSetIdle_Achieve*, *AgentSetIdle_Normal*, *AgentLeave_remove*, *RoleFail_FromAgentFail*, *AgentSetIdle_Fail*, *GoalSetEnd_SubGoal*, *RoleAchieve_FromAgentAchieve*, *GoalAchieve_FromRoleAchieve*, *GoalSetEnd_Role*, *RoleEnd_AgentIdle*, *GoalEnd_Remove*, *_Remove*, *GoalFail_-Recover*}

In the following, we present the properties: priority, sequential independence, confluence, and termination.

4.5.1 Priority

The rules have priorities allowing them to be executed in a specific order if they can be applied at the same time. Rules from R_m have a higher priority compared to the rules from R_{reo}. Rules in the same set, their priority reflects the process dialing with the event. For example, in response to the event of ending a goal, the rule "GoalSetEnd_SubGoal" has a higher priority compared to the rule "GoalSetEnd_Role" because the process requires to set the end flag to the sub-goal before to set an end to the role linked to the ended goal.

Furthermore, the designer can change the priority at design time according to the policy chosen. For example, to make the agent entered the system enacts a role before the agent already in an Idle state, the designer has to make the rule "AgentEnactRole_enter" have a higher priority than the rule "AgentEnactRole_idle".

Minimal Dependencies

Show

first \ second	1 AgentEnt...	2 AgentLe...	3 GoalSetE...	4 RoleAchi...	5 AgentAc...	6 AgentFail	7 GoalAchi...
1 AgentEnactRole_enter	0	0	0	0	1	1	0
2 AgentEnactRole_idle	0	0	0	0	1	1	0
3 AgentSetIdle	0	2	0	0	0	0	0
4 AgentSetIdle_normal	0	2	0	0	0	0	0
5 AgentSetIdle_Achieve	0	3	0	0	0	0	0
6 AgentSetIdle_Fail	0	3	0	0	0	0	0
7 AgentLeave_remove	0	0	0	0	0	0	0
8 GoalSetEnd_SubGoal	0	0	0	0	0	0	0
9 GoalEnd_remove	0	0	0	0	0	0	0
10 GoalSetEnd_Role	0	0	0	0	0	0	0
11 RoleEnd_AgentIdle	0	3	0	0	0	0	0
12 RoleAchieve_AgentIdle	0	3	0	0	0	0	0
13 RoleAchieveFromAgentAchieve	0	0	0	0	0	0	0
14 RoleFail_FromAgentFail	0	0	0	0	0	0	0
15 GolaFail_recover	0	0	1	0	0	0	0
16 AgentChangeRole	0	0	0	0	0	0	0
17 AgentChangeRole_freeRole	0	0	0	0	1	1	0
18 AgentChangeRole_NoFreeRole	0	0	0	0	1	1	0
19 RoleEnd_RmoveRole	0	0	0	0	0	0	0
20 GoalAchieve_FromRoleAchieve	0	0	0	0	0	0	0

Figure 4.30: Minimal dependence between R_m and R_{reo}

4.5.2 Sequentially Independent

Each pair of rule (R_m, R_{reo}) (see Figure 4.30 is sequentially independent (zero means that there is no dependency) except for some pairs that are considered as a normal dependency. For example, the rule *AgentLeave* is dependent on rules that put an agent in an Idle state such as *RoleAchieve_AgentIdle*.

Figure 4.31 represent the minimal dependencies computed between rules of the set R_m.

Minimal Dependencies

Show

first \ second	1	2	3	4	5	6	7
1 AgentEnter		0	0	0	0	0	0
2 AgentLeave	0		0	0	0	0	0
3 GoalSetEnd	0	0		0	0	0	0
4 RoleAchieve	0	0	0		0	0	0
5 AgentAchieve	0	0	0	0		0	0
6 AgentFail	0	0	0	0	0		0
7 GoalAchieve	0	0	0	0	0	0	

Figure 4.31: Minimal dependence between rules of the set R_m

4.5.3 Confluence

We have used *AGG* to compute the minimal conflict between pairs of different sets of rules. For example, the set R_m and R_{reo}. Figure 4.32 shows that these pairs are confluent (zero mean that there is no conflict). Some computed conflicts are to be ignored, such as "AgentLeave" and "AgentEnactRole_idle" because the rules representing events are applied first. Moreover, an agent leaving cannot enact a role unless it returns to an idle state.

Minimal Conflicts

Show

first \ second	1 Age...	2 Age...	3 Age...	4 Age...	5 Age...	6 Age...	7 Goa...	8 Goa...	9 Goa...	10 Ro...	11 Ro...	12 Ro...	13 Ro...	14 Go...	15 Ag...	16 Ag...	17 Ro...	18 Go...
1 AgentEnter	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2 AgentSetIdle	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3 AgentLeave	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4 GoalSetEnd	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5 RoleAchieve	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
6 AgentChangeRole	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7 AgentAchieve	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8 AgentFail	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9 GoalAchieve	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 4.32: Confluence between rules of the sets R_m and R_{reo}

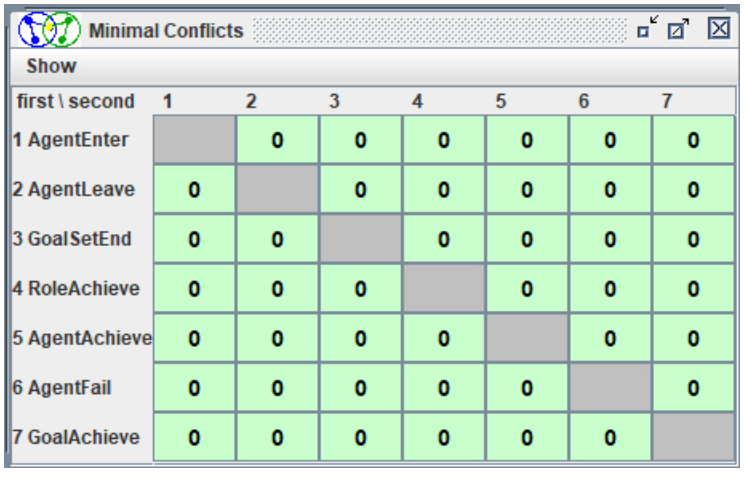
Figure 4.33 represents the minimal conflict computed between the struc-

tural and behavioral rules. Also, in this example, some conflicts are computed, but they are also to be ignored. For example, "RoleEnd_RemoveRole" and "AgentEnactRole_enter" because the reorganization manager will remove an ended role before any agent enacts it. Hence this rule is executed before any other rule of enactment.

first \ second	1 Goal...	2 Goal...	3 Goal...	4 Role...	5 Role...	6 Role...	7 RoleF...	8 GolaF...	9 Role...	10 Goa...
1 AgentEnactRole_enter	0	0	0	0	0	1	1	0	1	0
2 AgentEnactRole_idle	0	0	0	0	0	1	1	0	1	0
3 AgentSetIdle_normal	0	0	0	0	0	0	0	0	0	0
4 AgentSetIdle_Achieve	0	0	0	1	1	2	1	0	0	0
5 AgentSetIdle_Fail	0	0	0	1	1	1	2	0	0	0
6 AgentLeave_remove	0	0	0	0	0	0	0	0	0	0
7 AgentChangeRole_freeRole	0	0	0	0	0	0	0	0	0	0
8 AgentChangeRole_NoFreeRole	0	0	0	0	0	0	0	0	0	0

Figure 4.33: The confluence between rules of the set R_{reo}

Figure 4.34 represents the minimal conflict computed between the rules of the the set R_m.



first \ second	1	2	3	4	5	6	7
1 AgentEnter		0	0	0	0	0	0
2 AgentLeave	0		0	0	0	0	0
3 GoalSetEnd	0	0		0	0	0	0
4 RoleAchieve	0	0	0		0	0	0
5 AgentAchieve	0	0	0	0		0	0
6 AgentFail	0	0	0	0	0		0
7 GoalAchieve	0	0	0	0	0	0	

Figure 4.34: The confluence between rules of set R_m

4.5.4 Termination

In our approach, the problem of termination can arise from two reasons: the number of processed nodes (agent, goal, or role) and the loop generated in applying one or a series of rules repeatedly. In the following, we describe the solution provided by our approach:

1. The number of agents, roles, and goals is controlled via the three attributes of the ORG node presented in Section 4.3.1. Their values are given at design time and are adjustable at run-time. They are used in a condition of the form:

   if $M_{agent} \geq 0$ **then**
   $M_{agent} \leftarrow M_{agent} - 1$
   allow the agent to enter
   else
   agent not allowed
   end if

2. By a loop of rules, we mean, for example, the case of an agent failing and recovering continuously. In this case, the agent is in a loop by failing to fulfill its enacted role and recovering continuously. These two events, represented by several rules, are applied continuously, constructing an infinite loop.

   The monitor component watches and prevents these cases by following a predefined policy at design time. For the example of a failing and recovering agent, we present the most straightforward policy using the number of failures an agent is allowed to have. This number is represented via the attribute "M_{failur}". If this number is exceeded, the agent cannot recover and must be put in an Idle state. Other complex policies can be defined, such as using the time required to recover.

3. Negative Application Condition: It prevents a rule from being applied multiple times for the same nodes if it is not required. For example, the rule *RoleAchieve_FromAgentAchieve* (see Figure 4.26) prevents marking a role with the flag "achieve" if it is already achieved.

4.6 Conclusion

We have presented in this chapter an approach to formalize Multi-Agent Systems using graph transformation. In particular, we have defined a type graph to represent the state of the static part of the system and a set of rules to represent the dynamic part of the system. The application of the proposed rules can modify the state of the basic elements that are used to represent a MAS organization. We have defined a formalization of the life cycle of each kind of these elements. Our approach has been validated in the publication [FC19].

In what follows, we conclude our work with a general conclusion of what has been presented and the perspectives of the present work.

CHAPTER 5

Evaluation: Case Study

5.1 Introduction

In order to evaluate the proposed approach in this thesis, we used a case study related to a manufacturing system. In this chapter, we present how to model the manufacturing system using our specification as an organization-centered multi-agent system. After that, we use a set of scenarios of different sizes to show how the reorganization is performed based on the defined rules in this thesis. At the end of this chapter, we present the results of the semi-automatic check that is performed to see how the system still consistent and stable.

5.2 Case Study Description

In this evaluation, we use a case study related to a manufacturing system. The latter refers to a set of processes and operations used to manufacture a given product. The manufacturing system is also defined as the complex disposition of the physical manufacturing elements (machines, machine tools, people, materials handling equipment, and tooling) that are characterized and controlled by measurable parameters. They can manufacture products with a high degree of automation and many different specifications. Hence, they are more and more applied in factory automation.

Our selection of the manufacturing system as a case study is motivated by the fact that: manufacturers are stumbling into increasing challenges driven by vigorous global competition, well qualified and demanding consumers and fast product and process technological improvements. More precisely [Kor10]:

- Market shifts are increasingly rapid and unpredictable.
- Fast addition of new products and continuously varying demands for products.
- Increasing demands for personalized products, etc.

Hence, to stay competitive, manufacturers have to fast reorganize itself in response to those challenges. For instance, it can stop producing a particular piece no longer in demand to start producing a new one. This change of goal will require a reorganization in the system at hand.

In this case study, we consider a manufacturing system as a multi-agent system. The system is composed mainly of three components: i) a *hardware component*: such as production machines, tools, fixtures, and material handling equipment, ii) *measurable system parameters* such as machines state, production rate/cycle time, inventory, etc. and iii) an *operational component*: it is required to operate the manufacturing ranging from manager and controller to workers stuff.

According to our approach, these components can be seen as an organization defined by a set of goals, roles, and agents fulfilling roles. Here the goals initially vary from production of pieces to management of the manufacturing and marketing of the goods. The manufacture can produce many different pieces. Moreover, one-piece can be the result of the composition of many different smaller pieces. Each agent category has a set of capabilities that satisfy a set of role requirements' to be fulfilled.

As stated in the general introduction, several desirable or undesirable events can occur in a manufacturing system, affecting the system organization negatively or positively. In this chapter, we show examples of these events and how our approach can be applied to re-stabilize the system.

5.3 Planning and execution

In order to perform our evaluation, we have followed these steps:

- Use a set of scenarios of different sizes that represent events occurring in our manufacturing system
- For each scenario, we show how our specification is used and which rules can be applied
- A semi-automatic check is performed to see how the system still consistent and stable
- Describing and discussing the obtained results

Initially, we consider that our manufacturing system is composed of the following roles: a Product Manager (PM), an ENgineer (EN), and two SuPervisors (SP) to supervise the production of two types of pieces. For every piece, there is one machine, which is operated by a maximum number of two workers.

The graph that models the initial state of our system is shown in Figure 5.1. In this initial state, we have one global Goal related to the "Produce Pieces Goal" (*PPG*). This goal is decomposed into two sub-goals: "Produce Piece 1" and "Produce Piece 2". They are labeled respectively, *PP1* and *PP2*. For each sub-goal, there are two Roles attached to it. For example, the sub-goal *PP1* has attached to the roles "supervise product 1" (labeled as *sp1*) and "machine operator 1" (labeled as *mo1*).

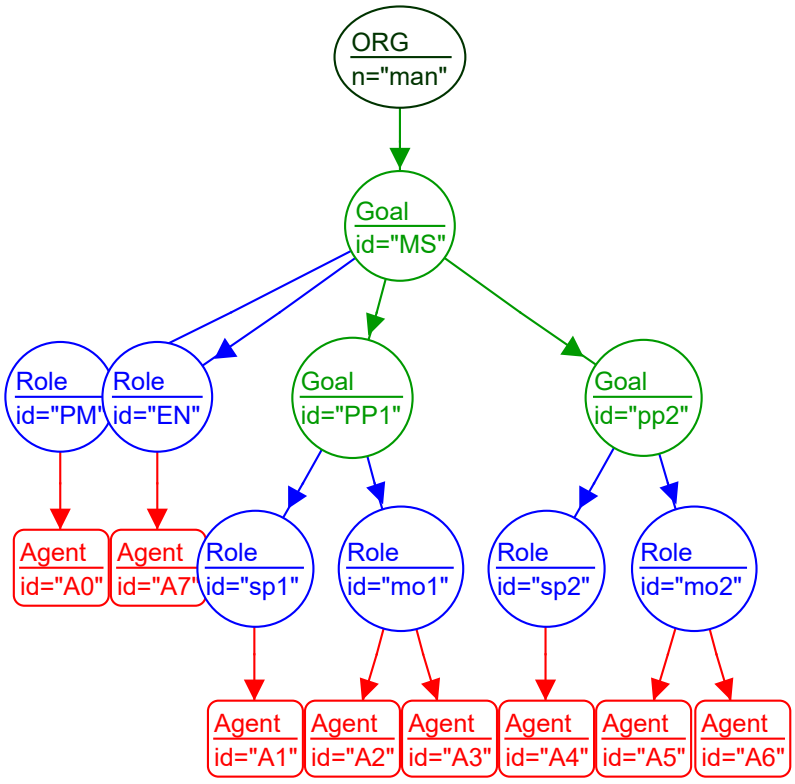


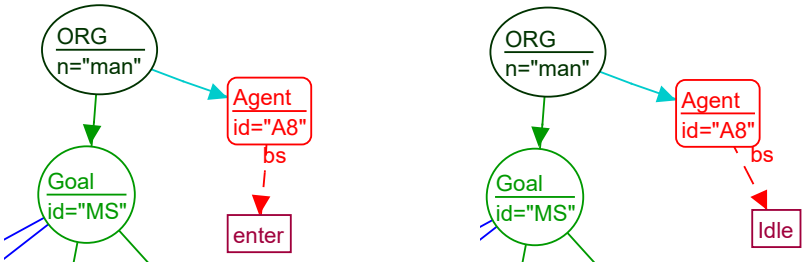
Figure 5.1: Initial Graph of the manufacturing system

We have prepared a set of scenarios that can be applied to this system. We show the results of the application of the corresponding rules that represent the different events and the possible reorganization. The selected scenarios in this case study are: "Agent Entering the System", "Agent Leaving the System",

and "A Goal Being Ended". The result of the applicability of different rules for these scenarios is presented in a way, we show only the changed parts of the graph that represents the state of the system.

5.4 Scenario 1: Agent Entering the System

In this scenario: An agent named $A8$ enters the manufacturing system and would like to enact a Role. However, all the Roles are satisfied. Therefore, it cannot enact a Role. Hence, it is put in an idle state. The application of different rules to handle this scenario is as follows: At first, the rule *AgentEnter* is applied to represent the event of an Agent entering the system, which is depicted by the Figure 5.2a. We can see that the agent $A8$ is connected to the ORG node and is marked with the flag "enter". Because there is no Role available to play, agent $A8$ is put in an "Idle" state by applying the Rule *AgentSetIdle*. We can see (Figure 5.2b) that the flag "enter" is removed and replaced by the flag "Idle".



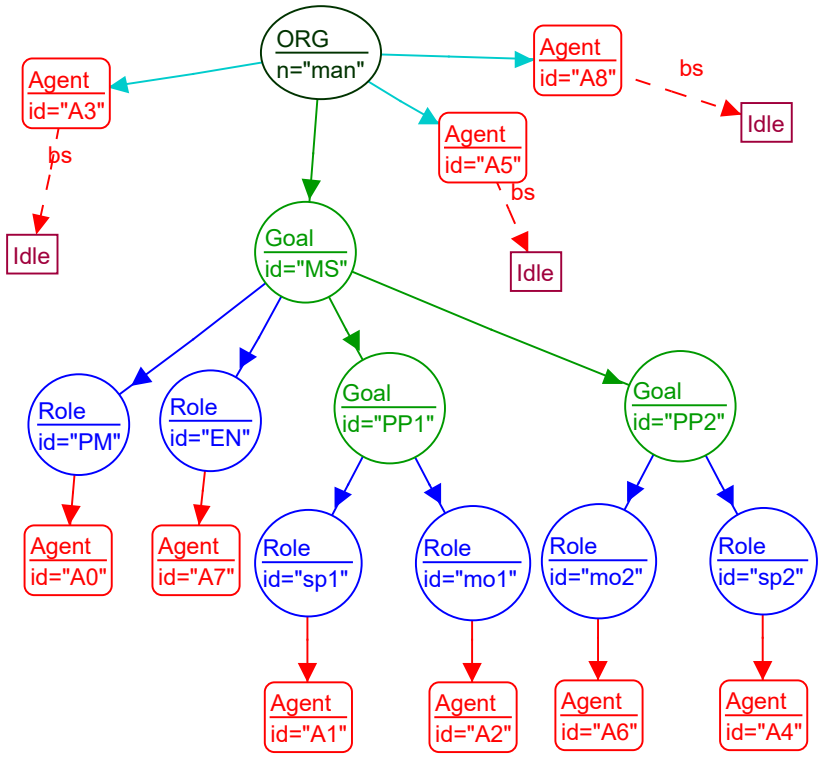
(a) Agent Entered the System (b) Agent in an "Idle" state

Figure 5.2: Scenario of Agent Entering the System

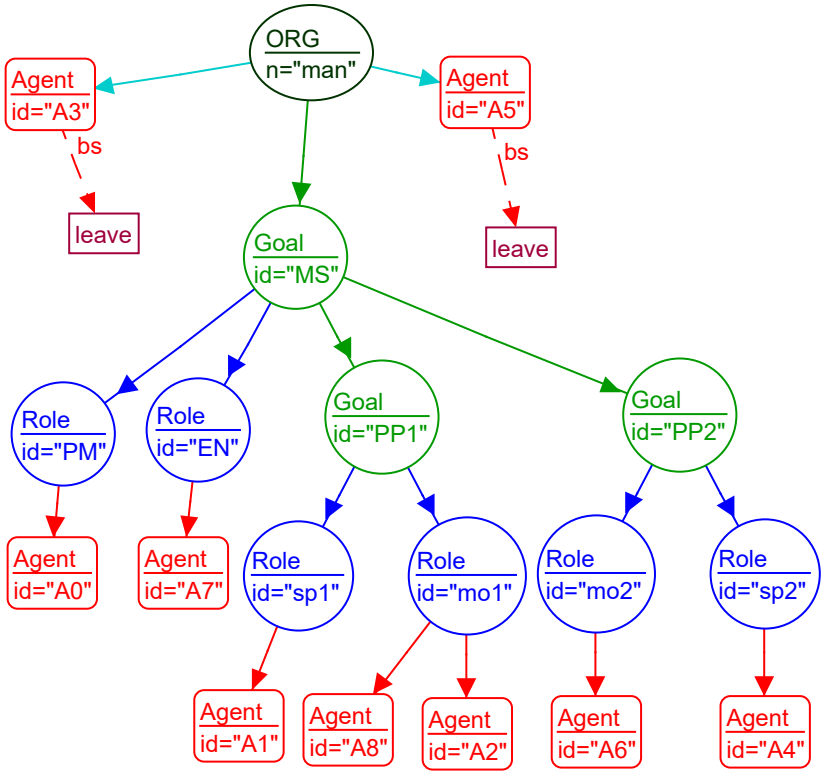
5.5 Scenario 2: Agent Leaving the System

In this scenario, Agents $A3$ and $A5$ want to leave the system. They are first put in an "Idle" state, which will result in Role $mo1$, and $mo2$ being not completely satisfied. Therefore, agent $A8$ enacts one of the available roles. Agent $A3$, and $A5$ flagged with the "leave" flag are removed.

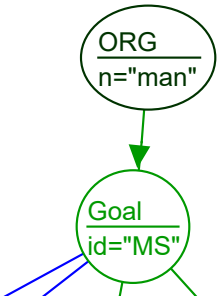
The application of different rules to handle this scenario is as follows: At first, the rule *AgentSetIdle_Normal* is applied to put the agents that want to leave the system in an "Idle" state, which is depicted by the Figure 5.3a. We can see that the two agents, $A3$ and $A5$ are now connected to the ORG node and marked by the "Idle" flag. The Roles $mo1$, and $mo2$ are now not completely satisfied, which trigger the rule *AgentEnactRole_idle*. We can see (Figure 5.3b) that the agent $A8$ have enacted the Role $mo1$ The rule *AgentLeave* is applied to mark the two agents with the flag "leave" to be removed from the system. We can see (Figure 5.3b) that the Idle flag is removed from the two agents and replaced with the flag "leave". Finally, the rule *AgentLeave_remove* is applied to remove the two agents from the system. We can see (Figure 5.3c) that the ORG node now has no direct connection to any agent.



(a) Agent $A5$ and $A3$ put in an "Idle" state



(b) Agent $A5$ and $A3$ marked to leave the system. Agent $A8$ enacted $mo1$

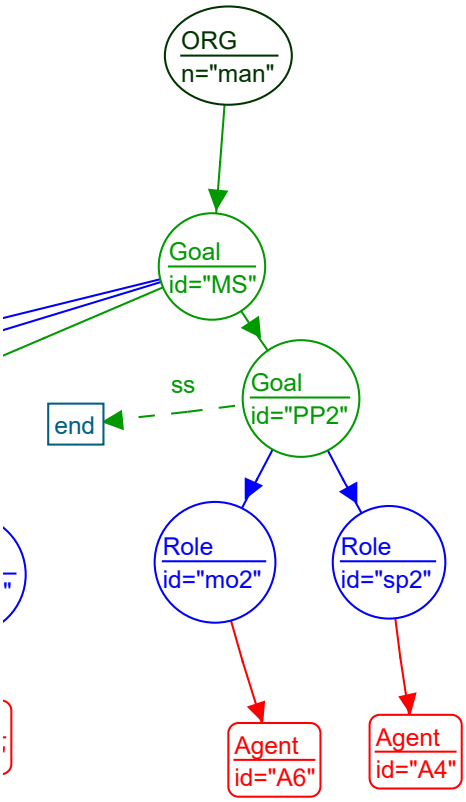


(c) Agent $A5$ and $A3$ Left the System

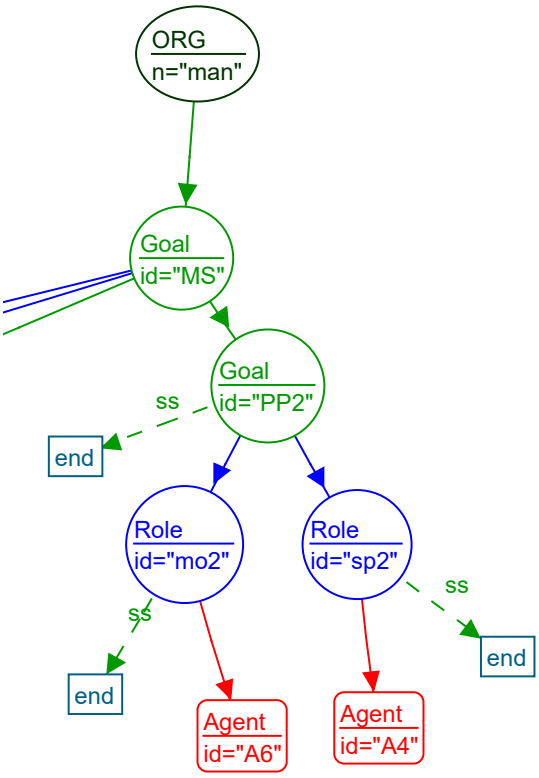
Figure 5.3: Scenario of Agent leaving the system

5.6 Scenario 3: A Goal Being Ended

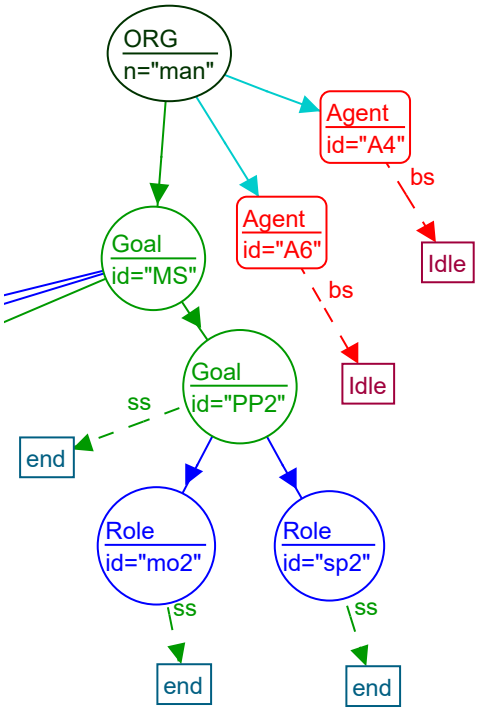
In this scenario, a piece (piece 2) being produced by the manufacturing system is no longer in demand. Therefore, it stops its production by ending the Goal of producing this piece. The application of different rules to handle this scenario is as follows: Firstly, the rule *GoalSetEnd* is applied to mark the Goal $pp2$ with the flag "end". The result of this application is depicted in Figure 5.4a. After that, the rule *GoalSetEnd_Role* is applied to mark all the Roles connected to the Goal $pp2$ with the flag "end". we can see (Figure 5.4b) that the Roles $sp2$ and $mo2$ are marked with this flag. This trigger the rule *AgentSetIdle_Normal* and put the Agents $A6$ and $A4$ in an "Idle" state. The result of this application is depicted by Figure 5.4c. Finally, the rules *GoalEnd_Remove* and *RoleEnd_Remove* are applied to remove the Goals and Roles marked by the "end" flag from the system.



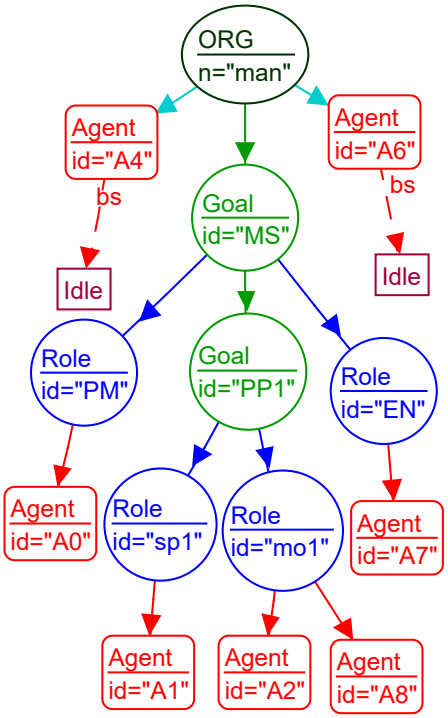
(a) Goal $pp2$ ended



(b) Roles connected to Goal $pp2$ ended



(c) Agents connected to the ended Roles are put in an "Idle" state



(d) The ended Goal and Roles are removed from the system

Figure 5.4: Result of the application of Rules AgentEnter, and AgentSetIdle_enter

5.7 Discussion and limitations

The application of the different rules during the case study shows that our approach is easy to use and allows us to represent the system (and the reorganization) in an expressive way. For instance, agents $A2$ and $A3$ in the initial graph are connected to the same Role $mo1$, which means that they are collaborating in fulfilling this role. In addition, the use of the formal representation allows performing the reorganization in an exact way. Also, our evaluation demonstrates that our formal definition of the MAS (re)organization allows to make the system re-stabilize even for large scenarios (such as the third scenario) of events.

As with any graph transformation approach, our solution suffers from some limitations. In fact, our solution depends on a graph transformation engine. Hence, the performance of our solution depends on the performance of the used engine. So, if we have a large system with an important number of Goals, Roles, and Agents, this can be modeled using a large-sized graph. By consequence, it can slow down the application of rules.

5.8 Conclusion

We have used a concrete example related to a manufacturing system to demonstrate all the aspects of our approach. We have applied a set of scenarios of reorganization that cover all the aspects defined in our rules. The obtained results showed the efficiency of the proposed approach.

In the near future, we plan to develop a simulation tool that integrates our approach, which can be used to check an organization's effectiveness under different circumstances.

CHAPTER 6

Conclusion and Future Work

6.1 Summary

For decades, formal approaches in software engineering are used to develop safety-critical or security-critical software and systems. Nowadays, formal approaches are proposed to deal with the problem of systems reorganization and adaptation. They show that the formalization can preserve certain properties of organizations and are also used to reason about what reorganization actions are required to achieve basic organizational structure [DD14]. In this dissertation, we presented a formal approach to formalize the process of reorganization in MAS (in Chapter 4). We have used the algebraic graph transformation approach to describe and define the different aspects (structural and behavioral) of a MAS organization. In fact, we have split our approach into three components to precisely describe each aspect, which are : i) *MAS Monitor*, ii) *MAS Organization*, and iii) *Reorganization Manager*. MAS Organization: represent the structural aspect of MAS. It is a type-graph (similar to the meta-model concept) that defines its architectural elements (goal, role, and agent) and the different relationships between them. The latter can be as follow: a goal can be decomposed into several goals. A goal can have one or several roles. An agent or several ones can enact a role in order to fulfill it. MAS Monitor: represent the behavioral aspect of MAS. It is a set of graph transformation rules. Specifically, they define the monitoring behavior in a MAS organization. It watches the system continuously for any given change and changes its state from normal to a reorganizing state by applying the corresponding rule. Reorganization Manager: Like the MAS monitor component, it is a set of rules that manages the reorganization response to any change detected by the monitor component. For every change detected, a series of rules can be applied to return the system to its normal state.

The advantage of our approach can be summarized as follow:

- It uses graph transformation as a descriptive tool that benefits the user from the intuitive and easiness of graph notation usage and from the formal foundation of the algebraic graph transformation.
- The small learning curve for a designer to start using our approach as it

is almost a graphical notation.

- The ease of extending our approach using different mechanisms such as attributes and context conditions.

We have implemented our approach using AGG, which is beneficial to us as it facilitates the editing process and provides integrated tools such as critical pairs analysis. We have evaluated our approach using a case study related to a manufacturing system. We have used a set of imaginary scenarios of events occurring to show how our approach reorganizes the system for the purpose of achieving a stable state. The obtained results show the efficiency and effectiveness of our approach.

6.2 Perspectives

Since the use of AGG in the implementation made it coupled with its environment. Hence we cannot use other tools such as GROOVE with our implementation. As a future work, we plan to provide a translation layer that allows us to transform our implemented approach with a particular tool to another.

In the near future, we plan to implement a simulation tool that uses the proposed graph grammar. This tool will allow us to test an organization to find the best configuration that allows it to survive the change in its environment. Consequently, this tool has to show graphically how the reorganization is performed using our rules.

At the conceptual level, we plan to define additional rules that allow transforming the MAS graph into a Maude specification. This latter can be used to verify at run-time that the system preserves its properties after a given reorganization.

Bibliography

[Ald08] H. Aldrich. *Organizations and environments*. Stanford University Press, 2008.

[ABJ11] E. Argente, V. Botti, and V. Julian. "GORMAS: An Organizational-Oriented Methodological Guideline for Open MAS". In: *Agent-Oriented Software Engineering X*. Ed. by M.-P. Gleizes and J. J. Gomez-Sanz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 32–47. ISBN: 978-3-642-19208-1.

[AB97] R. C. Arkin and T. Balch. "Cooperative multiagent robotic systems". In: (1997).

[BC95] W. P. Barnett and G. R. Carroll. "Modeling internal organizational change". In: *Annual review of sociology* 21.1 (1995), pp. 217–236.

[BH01] G. Beavers and H. Hexmoor. "Teams of agents". In: *2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace (Cat. No. 01CH37236)*. Vol. 1. IEEE. 2001, pp. 574–582.

[BCG07] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*. Vol. 7. John Wiley & Sons, 2007.

[BKC18] A. Boucherit, A. Khababa, and L. M. Castro. "Automatic generating algorithm of rewriting logic specification for multi-agent system models based on Petri nets". In: *Multiagent and Grid Systems* 14.4 (2018), pp. 403–418. DOI: 10.3233/MGS-180298.

[BDA00] C. H. Brooks, E. H. Durfee, and A. Armstrong. "An introduction to congregating in multi-agent systems". In: *Proceedings Fourth International Conference on MultiAgent Systems*. IEEE. 2000, pp. 79–86.

[Buc+15] A. Bucchiarone, H. Ehrig, C. Ermel, P. Pelliccione, and O. Runge. "Rule-Based Modeling and Static Analysis of Self-adaptive Systems by Graph Transformation". In: *Software, Services, and Systems: Essays Dedicated to Martin Wirsing on the Occasion of His Retirement from the Chair of Programming and Software Engineering*. Springer International Publishing, 2015, pp. 582–601. ISBN: 978-3-319-15545-6. DOI: 10.1007/978-3-319-15545-6_33.

[CG99] K. M. Carley and L. Gasser. "Computational organization theory". In: *Multiagent systems: A modern approach to distributed artificial intelligence* (1999), pp. 299–330.

[Cha+01] H. Chalupsky et al. "Electric Elves: Applying Agent Technology to Support Human Organizations." In: *IAAI*. Vol. 1. 2001, pp. 51–58.

[CL98] D. D. Corkill and S. E. Lander. "Diversity in agent organizations". In: *Object Magazine* 8.4 (1998), pp. 41–47.

[Cor+97] A. Corradini et al. "Algebraic approaches to graph transformation–part i: Basic concepts and double pushout approach". In: *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific, Feb. 1997, pp. 163–245. DOI: 10.1142/9789812384720_0003.

[CD96] A. d. R. Costa and Y. Demazeau. "Toward a formal model of multi-agent systems with dynamic organizations". In: *Proceedings of the International Conference on Multi-Agent Systems, MIT Press, Kyoto, Japan*. Vol. 431. 1996.

[CGL08] V. Crespi, A. Galstyan, and K. Lerman. "Top-down vs bottom-up methodologies in multi-agent system design". In: *Autonomous Robots* 24.3 (2008), pp. 303–313.

[Cse+02] G. Csertán et al. "VIATRA-visual automated transformations for formal verification and validation of UML models". In: *Proceedings 17th IEEE International Conference on Automated Software Engineering*, IEEE. 2002, pp. 267–270.

[DV02] J. De Lara and H. Vangheluwe. "AToM 3: A Tool for Multi-formalism and Meta-modelling". In: *International Conference on Fundamental Approaches to Software Engineering*. Springer. 2002, pp. 174–188.

[DSW97] K. Decker, K. Sycara, and M. Williamson. "Middle-agents for the internet". In: *IJCAI (1)*. 1997, pp. 578–583.

[DeL09] S. A. DeLoach. "OMACS: A framework for adaptive, complex systems". In: *Handbook of research on multi-agent systems: Semantics and dynamics of organizational models*. IGI Global, 2009, pp. 76–104.

[DG14] S. A. DeLoach and J. C. Garcia-Ojeda. "The o-mase methodology". In: *Handbook on Agent-Oriented Design Processes*. Springer, 2014, pp. 253–285.

[DC96] Y. Demazeau and A. R. Costa. "Populations and organizations in open multi-agent systems". In: *Proceedings of the 1st National Symposium on Parallel and Distributed AI*. 1996, pp. 1–13.

[DD14] F. Dignum and V. Dignum. "A formal semantics for agent (re)organization". In: *Journal of Logic and Computation* 24.6 (Dec. 2014), pp. 1341–1363. DOI: 10.1093/logcom/ext058.

[DSD04] M. V. Dignum, E. Sonenberg, and F. P. M. Dignum. "Dynamic reorganization of agent societies". In: *Proceedings of workshop on coordination in emergent agent societies*. 2004.

[Dig+05] M. Dignum, F. Dignum, V. Furtado, A. Melo, and L. Sonenberg. "Towards a simulation tool for evaluating dynamic reorganization of agents societies". In: *Workshop on socially inspired computing*. 2005.

[Dig09] V. Dignum. *Handbook of research on multi-agent systems: semantics and dynamics of organizational models*. Information Science Reference Hershey, 2009.

[DD12] V. Dignum and F. Dignum. "A logic of agent organizations". In: *Logic Journal of the IGPL* 20.1 (2012), pp. 283–316.

[DKH97] F. Drewes, H.-J. Kreowski, and A. Habel. "Hyperedge replacement graph grammars". In: *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific, Feb. 1997, pp. 95–162. DOI: 10.1142/9789812384720_0002.

[EPS73] H. Ehrig, M. Pfender, and H. J. Schneider. "Graph-grammars: An algebraic approach". In: *14th Annual Symposium on Switching and Automata Theory (swat 1973)*. Oct. 1973, pp. 167–180. DOI: 10.1109/SWAT.1973.11.

[Ehr+15a] H. Ehrig, C. Ermel, U. Golas, and F. Hermann. *Graph and Model Transformation*. Monographs in Theoretical Computer Science. An EATCS Series. 00105. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. DOI: 10.1007/978-3-662-47980-3. (Visited on 01/26/2016).

[Ehr+15b] H. Ehrig, C. Ermel, U. Golas, and F. Hermann. *Graph and Model Transformation*. Monographs in Theoretical Computer Science. An EATCS Series. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. DOI: 10.1007/978-3-662-47980-3. (Visited on 01/26/2016).

[Ehr+97] H. Ehrig et al. "Algebraic approaches to graph transformation–part II: Single pushout approach and comparison with double pushout approach". In: *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific, Feb. 1997, pp. 247–312. DOI: 10.1142/9789812384720_0004.

[ER97] J. Engelfriet and G. Rozenberg. "Node Replacement Graph Grammars". In: *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific, Feb. 1997, pp. 1–94. DOI: 10.1142/9789812384720_0001.

[FK18] M. I. Fakhir and S. A. R. Kazmi. "Formal specification and verification of self-adaptive concurrent systems". In: *IEEE Access* 6 (2018), pp. 34790–34803.

[FC19] G. Fayçal and A. Chaoui. "A graph transformation based approach for multi-agent systems reorganization". In: *Multiagent and Grid Systems* 15.4 (2019), pp. 375–394.

[FG98] J. Ferber and O. Gutknecht. "A meta-model for the analysis and design of organizations in multi-agent systems". In: *Proceedings International Conference on Multi Agent Systems (Cat. No. 98EX160)*. IEEE. 1998, pp. 128–135.

[FGM04] J. Ferber, O. Gutknecht, and F. Michel. "From Agents to Organizations: An Organizational View of Multi-Agent Systems". In: *Agent-Oriented Software Engineering IV*. Ed. by G. Goos et al. Vol. 2935. 00000. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 214–230. DOI: 10.1007/978-3-540-24620-6_15.

[FW99] J. Ferber and G. Weiss. *Multi-agent systems: an introduction to distributed artificial intelligence*. Vol. 1. Addison-Wesley Reading, 1999.

[FG97] S. Franklin and A. Graesser. "Is It an agent, or just a program?: A taxonomy for autonomous agents". In: *Intelligent Agents III Agent Theories, Architectures, and Languages*. Ed. by J. P. Müller, M. J. Wooldridge, and N. R. Jennings. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 21–35. ISBN: 978-3-540-68057-4.

[Gar+07] J. C. Garcia-Ojeda, S. A. DeLoach, W. H. Oyenan, J. Valenzuela, et al. "O-MaSE: a customizable approach to developing multiagent development processes". In: *International Workshop on Agent-Oriented Software Engineering*. Springer. 2007, pp. 1–15.

[Gas92] L. Gasser. "An overview of DAI". In: *Distributed Artificial Intelligence: Theory and Praxis* 9.9-29 (1992), p. 28.

[Gas01] L. Gasser. "Perspectives on organizations in multi-agent systems". In: *ECCAI Advanced Course on Artificial Intelligence*. Springer. 2001, pp. 1–16.

[Gen97] M. R. Genesereth. "An agent-based framework for interoperability". In: *Software agents* (1997), pp. 317–345.

[Gri03] N. Griffiths. "Supporting cooperation through clans". In: *Cybernetic Intelligence–Challenges and Advances, Proceedings IEEE Systems, Man and Cybernetics, 2nd UK&RI Chapter Conference*. 2003.

[GS88] B. J. Grosz and C. L. Sidner. *Plans for discourse*. Tech. rep. BBN LABS INC CAMBRIDGE MA, 1988.

[Han+00] M. Hannoun, O. Boissier, J. S. Sichman, and C. Sayettat. "MOISE: An organizational model for multi-agent systems". In: *Advances in Artificial Intelligence*. Springer, 2000, pp. 156–165.

[HCY99] S. C. Hayden, C. Carrick, and Q. Yang. "A catalog of agent coordination patterns". In: *Proceedings of the third annual conference on Autonomous Agents*. 1999, pp. 412–413.

[HM99] O. Holland and C. Melhuish. "Stigmergy, self-organization, and sorting in collective robotics". In: *Artificial life* 5.2 (1999), pp. 173–202.

[HL04] B. Horling and V. Lesser. "A survey of multi-agent organizational paradigms". In: *The Knowledge engineering review* 19.4 (2004), pp. 281–316. DOI: 10.1017/S0269888905000317.

[HSB02] J. F. Hübner, J. S. Sichman, and O. Boissier. "MOISE+: Towards a Structural, Functional, and Deontic Model for MAS Organization". In: *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*. AAMAS '02. Bologna, Italy: ACM, 2002, pp. 501–502. DOI: 10.1145/544741.544858.

[HVB08] J. F. Hübner, L. Vercouter, and O. Boissier. "Instrumenting multi-agent organisations with artifacts to support reputation processes". In: *International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems*. Springer. 2008, pp. 96–110.

[IGE00] C. Intanagonwiwat, R. Govindan, and D. Estrin. "Directed diffusion: A scalable and robust communication paradigm for sensor networks". In: *Proceedings of the 6th annual international conference on Mobile computing and networking*. 2000, pp. 56–67.

[IGY92] T. Ishida, L. Gasser, and M. Yokoo. "Organization self-design of distributed production systems". In: *IEEE Transactions on Knowledge and Data Engineering* 4.2 (1992), pp. 123–134.

[Jen01a] N. R. Jennings. "An agent-based approach for building complex software systems". In: *Communications of the ACM* 44.4 (2001), pp. 35–41.

[Jen99] N. R. Jennings. "Agent-Oriented Software Engineering". In: *Multiple Approaches to Intelligent Systems*. Ed. by I. Imam, Y. Kodratoff, A. El-Dessouki, and M. Ali. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 4–10. ISBN: 978-3-540-48765-4.

[Jen00] N. R. Jennings. "On agent-based software engineering". In: *Artificial Intelligence* 117.2 (2000), pp. 277–296. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(99)00107-1.

[Jen01b] N. R. Jennings. "An Agent-based Approach for Building Complex Software Systems". In: *Commun. ACM* 44.4 (Apr. 2001), pp. 35–41. DOI: 10.1145/367211.367250.

[JSP13] J. d. Jong, L. Stellingwerff, and G. E. Pazienza. "Eve: A Novel Open-Source Web-Based Agent Platform". In: *2013 IEEE International Conference on Systems, Man, and Cybernetics*. 2013, pp. 1537–1541.

[KKS19] N. Khakpour, J. Kleijn, and M. Sirjani. "A Formal Model to Integrate Behavioral and Structural Adaptations in Self-adaptive Systems". In: *International Conference on Fundamentals of Software Engineering*. Springer. 2019, pp. 3–19.

[KG02] M. Klusch and A. Gerber. "Dynamic coalition formation among rational agents". In: *IEEE Intelligent Systems* 17.3 (2002), pp. 42–47.

[KJ98] M. Knapik and J. Johnson. *Developing Intelligent Agents for Distributed Systems: Exploring Architecture, Technologies, & Applications*. USA: McGraw-Hill, Inc., 1998. ISBN: 0070350116.

[Koe68] A. Koestler. "The ghost in the machine." In: (1968).

[Kor10] Y. Koren. *The global manufacturing revolution: product-process-business integration and reconfigurable systems*. Vol. 80. John Wiley & Sons, 2010.

[KZ96] C. R. Kube and H. Zhang. "The use of perceptual cues in multi-robot box-pushing". In: *Proceedings of IEEE international conference on robotics and automation*. Vol. 3. IEEE. 1996, pp. 2085–2090.

[LMS17] M. A. Laouadi, F. Mokhati, and H. Seridi-Bouchelaghem. "A formal framework for organization-centered multi-agent system specification: A rewriting logic based approach". In: *Multiagent and Grid Systems* 13.4 (2017), pp. 395–419.

[Les91] V. R. Lesser. "A retrospective view of FA/C distributed problem solving". In: *IEEE Transactions on Systems, Man, and Cybernetics* 21.6 (1991), pp. 1347–1362.

[MN10] C. M. Macal and M. J. North. "Tutorial on agent-based modelling and simulation". In: *Journal of Simulation* 4.3 (2010), pp. 151–162. DOI: 10.1057/jos.2010.3.

[MT03] S. McShane and T. Travaglione. *Organisational behaviour on the Pacific Rim*. McGraw-Hill, 2003.

[MFC13] A. M. de Mello, L. Foss, and S. A. da Costa Cavalheiro. "Towards the use of graph grammars for specification of multi-agent system organizations". In: *Simposio Brasileiro de Metodos Formais 2013-Short Papers*. 2013, pp. 42–47.

[MW04] C. Merida-Campos and S. N. Willmott. "Modelling coalition formation over time for iterative coalition games". In: (2004).

[Mey14] J.-J. C. Meyer. "Logics for intelligent agents and multi-agent systems". In: (2014).

[MD93] T. A. Montgomery and E. H. Durfee. "Search reduction in hierarchical distributed problem solving". In: *Group Decision and Negotiation* 2.3 (1993), pp. 301–317.

[NNZ00] U. Nickel, J. Niere, and A. Zündorf. "The FUJABA environment". In: *Proceedings of the 22nd international conference on Software engineering*. 2000, pp. 742–745.

[OPF03] J. J. Odell, H. V. D. Parunak, and M. Fleischer. "The Role of Roles in Designing Effective Agent Organizations". In: *Software Engineering for Large-Scale Multi-Agent Systems*. Springer Berlin Heidelberg, 2003, pp. 27–38. DOI: 10.1007/3-540-35828-5_2.

[PO01] H. V. D. Parunak and J. J. Odell. "Representing social structures in UML". In: *International workshop on agent-oriented software engineering*. Springer. 2001, pp. 1–16.

[Pic+09] G. Picard, J. F. Hübner, O. Boissier, and M.-P. Gleizes. "Reorganisation and self-organisation in multi-agent systems". In: *1st International Workshop on Organizational Modeling, ORGMOD*. 2009, pp. 66–80.

[RGR15] V. Rafe, M. Golparian, and S. Rasoolzadeh. "Using graph transformation systems to formalize Tropos diagrams". In: *Journal of Visual Languages & Computing* 30 (2015), pp. 1–16.

[Ren04] A. Rensink. "The GROOVE Simulator: A Tool for State Space Generation". In: *Applications of Graph Transformations with Industrial Relevance*. Ed. by J. L. Pfaltz, M. Nagl, and B. Böhlen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 479–485. DOI: 10.1007/978-3-540-25959-6_40.

[RET12] O. Runge, C. Ermel, and G. Taentzer. "AGG 2.0 – New Features for Specifying and Analyzing Algebraic Graph Transformations". en. In: *Applications of Graph Transformations with Industrial Relevance*. Ed. by D. Hutchison et al. Vol. 7233. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 81–88. DOI: 10.1007/978-3-642-34176-2_8.

[RND10] S. J. Russell, P. Norvig, and E. Davis. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall series in artificial intelligence. Prentice Hall, 2010. ISBN: 978-0-13-604259-4.

[Saw03] R. K. Sawyer. "Artificial Societies". In: *Sociological Methods & Research* 31.3 (Feb. 2003), pp. 325–363. DOI: 10.1177/0049124102239079.

[SK98] O. Shehory and S. Kraus. "Methods for task allocation via agent coalition formation". In: *Artificial intelligence* 101.1-2 (1998), pp. 165–200.

[Tae99] G. Taentzer. "AGG: A tool environment for algebraic graph transformation". In: *International Workshop on Applications of Graph Transformations with Industrial Relevance*. Springer. 1999, pp. 481–488.

[Tam97] M. Tambe. "Towards flexible teamwork". In: *Journal of artificial intelligence research* 7 (1997), pp. 83–124.

[Tia07] H. Tianfield. "A new framework of holonic self-organization for multi-agent systems". In: *2007 IEEE International Conference on Systems, Man and Cybernetics*. IEEE. 2007, pp. 753–758.

[Van+05] E. L. Van Den Broek, C. M. Jonker, A. Sharpanskykh, J. Treur, et al. "Formal modeling and analysis of organizations". In: *International Conference on Autonomous Agents and Multiagent Systems*. Springer. 2005, pp. 18–34.

[WLZ06] Z.-g. Wang, X. Liang, and Q. Zhao. "A Graph Transformation System Model of Dynamic Reorganization in Multi-agent Systems". In: *Intelligent Data Engineering and Automated Learning - IDEAL 2006, 7th International Conference, Burgos, Spain, September 20-23, 2006, Proceedings*. Ed. by E. Corchado, H. Yin, V. J. Botti, and C. Fyfe. Vol. 4224. Lecture Notes in Computer Science. Springer, 2006, pp. 1182–1190. DOI: 10.1007/11875581_140.

[Woo09] M. Wooldridge. *An introduction to multiagent systems*. 2nd. John Wiley & Sons, 2009. ISBN: 978-0470519462.

[WJK00] M. Wooldridge, N. R. Jennings, and D. Kinny. "The Gaia methodology for agent-oriented analysis and design". In: *Autonomous Agents and multi-agent systems* 3.3 (2000), pp. 285–312.

[YKO03] O. Yadgar, S. Kraus, and C. L. Ortiz. "Scaling-Up Distributed Sensor Networks: Cooperative Large-Scale Mobile-Agent Organizations". In: *Distributed Sensor Networks: A Multiagent Perspective*. Ed. by V. Lesser, C. L. Ortiz, and M. Tambe. Boston, MA: Springer US, 2003, pp. 185–217. ISBN: 978-1-4615-0363-7. DOI: 10.1007/978-1-4615-0363-7_9.

[ZV02] F. Zambonelli and H. Van Dyke Parunak. "From design to intention: signs of a revolution". In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*. 2002, pp. 455–456.