Contents

List of Figures

List of Tables

General Introduction

Systems are becoming more complex, in part due to increased customer requirements and the expectation that applications should be seamlessly integrated with other existing, often distributed applications and systems. In addition, there is an increasing demand for these complex systems to exhibit some type of intelligence as well.

No longer is it „good enough ”to be able to access systems across the internet, but customers require that their systems know how to access data and systems, even in the face of unexpected events or failures.

The goal of our research is to develop a framework for constructing complex, distributed systems that can autonomously adapt to their environment. Multiagent systems have become popular over the last few years for providing the basic notions that are applicable to this problem. A multiagent

system uses groups of self-directed agents working together to achieve a common goal. Such multiagent systems are widely proposed as replacements for sophisticated, complex, and expensive stand-alone systems for similar applications. Multiagent systems tend to be more robust and, in many cases, more efficient (due to their ability to perform parallel actions) than single monolithic applications. In addition, the individual agents tend to be simpler to build, as they are built from a single agent?s perspective.

However, unpredictable application environments make multiagent systems susceptible to individual failures that can significantly reduce the ability of the system to accomplish its goal.

The problem is that multiagent systems are typically designed to work within a limited set of configurations. Even when the system possesses the resources and computational ability to accomplish its goal, it may be constrained by its own structure and knowledge of its member?s capabilities, which may change over time.

In most multiagent design methodologies ,the system designer analyzes the possible organizational structure - which determines which roles are required to accomplish which goals and sub-goals and then designs one organization that will suffice for most anticipated scenarios.

Unfortunately, in dynamic applications where the environment as well as the agents may undergo changes, a designer can rarely account for, or even consider, all possible situations. Attempts to overcome these problems include large-scale redundancy using

homogenous capabilities and centralized/distributed planning.

However,homogenous approaches negate many of the benefits of using a multiagent approach and are not applicable in complex applications where specific capabilities are often needed by only one or two agents. Centralized and distributed planning approaches tend to be brittle and computationally expensive due to their required level of detail (individual actions in most cases).

To overcome these problems, we are developing a framework that allows a system to design its own organization at runtime. In essence, we propose to provide the system with organizational knowledge and let the system design its own organization based on the current goals and its current capabilities.

While the designer can provide guidance, supplying the system with key organizational information will allow it to redesign, or reorganize, itself to match its scenario. This paper presents a key component of our framework, a metamodel for multiagent organizations named the Organization Model for Adaptive Computational Systems (OMACS).

OMACS defines the requisite knowledge of a system?s organizational structure and capabilities that will allow the system to reorganize at runtime and enable it to achieve its goals in the face of a changing environment and its agent?s capabilities.

Chapter 1

Organization Multi-Agent System

Designing and implementing large, complex, and distributed systems by resorting to autonomous or semi-autonomous agents that can reorganize themselves by cooperating with one another represent the future of software systems A set of methodologies a selection of design processes , and a collection of frameworks are available in the literature to provide the basis for constructing sophisticated autonomous multi-agent organizations .

Moreover, a set of metrics and methods have been suggested with the intention of providing useful information about key properties (e.g., complexity, flexibility, self-organized, performance, scalability, and cost) of these multi-agent organizations .

This chapter introduces a suite of technologies for building complex, adaptive systems. It is based in the multi-agent systems paradigm and uses the Organization Model for Adaptive Computational Systems (OMACS). And presents a suite of technologies including the Organization-based Multiagent Systems Engineering (O-MaSE) methodology,

Framework of Modilization MaS

1.1 Organization Multi-Agent System Engineering

1.1.1 Overview of O-MaSE

the Organization-based Multiagent System Engineering (O-MaSE) is a framework that allows designers to create custom agent-oriented development processes.

This custom agent-oriented process is generated following a process metamodel and then instantiated from a set of method fragments and guidelines by using a method engineering approach .

O-MaSE defines a metamodel, a repository of method fragments and a set of guidelines. The O-MaSE metamodel defines general concepts used in multiagent systems along with their relationships and is based on an organizational approach.

The O-MaSE Process Framework is based on the OPEN Process Framework (OPF) and uses the OPF metamodel , as shown in Figure 1.1 .

1. level M2 : which defines processes in terms of Work Units (Activities, Tasks, and Techniques), Producers, and Work Products.

2. Level M1 : contains the definition of O-MaSE in the form of the O-MaSE metamodel, method fragments , and guidelines.

3. Level M0 : level for specific projects (a process instance).

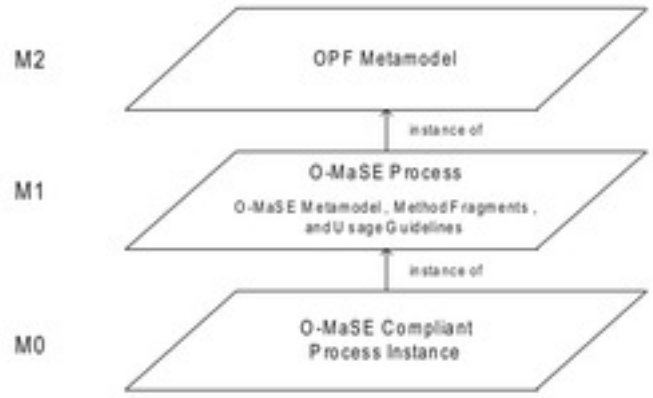

Figure 1.1: O-MaSE Process Framework

1.1.2 The Goal of O-MaSE

The O-MaSE Process Framework is to allow process engineers to construct custom agent-oriented processes using a set of method fragments, all of which are based on a common metamodel.

To achieve this, we define O-MaSE in terms of a metamodel, a set of method fragments, and a set of guidelines.

The O-MaSE metamodel defines a set of analysis, design, and implementation concepts and a set of constraints between them. The method fragments define how a set of analysis and design products may be created and used within O-MaSE. Finally, guidelines define how the method fragment may be combined to create valid O-MaSE processes, which we refer to as O-MaSE compliant processes.

The O-MaSE methodology is supported by the agentTool III [1] development environment 2 , which is designed as a set of Eclipse plug-ins. agentTool includes a plug-in for each O-MaSE model and the agentTool Process Editor (APE), which was developed to support the design and definition of O-MaSE compliant processes.

The APE plug-in is based on the Eclipse Process Framework and provides a process designer the ability to :

1. extend O-MaSE with new tasks, models, or usage guidelines

[1] http://agenttool.cs.ksu.edu/

2. create new process instances by composing tasks , models, and producers from the O-MaSE method fragment library

3. verifying that they meet process guidelines

1.2 Organization Model for Adaptive Computational System

1.2.1 Overview of OMACS

The OMACS is Framework , this model grew from the MaSE metamodel , which was based on the original AGR[2] model .

We realized that while we could define multiagent systems in terms of agent playing roles in order to achieve system goals, we did not have to limit the flexibility of agents by predefining the roles they could and could not play.

Noting that agents could be assigned to roles based on the capabilities Required to play various roles and the capabilities possessed by the agents, we figured the agents could adapt their assignments based on the current set of goals required to be achieved by the system. This basic idea led to the OMACS model as defined in DeLoach, Oyenan, and Matson (2007) and shown in Figure .

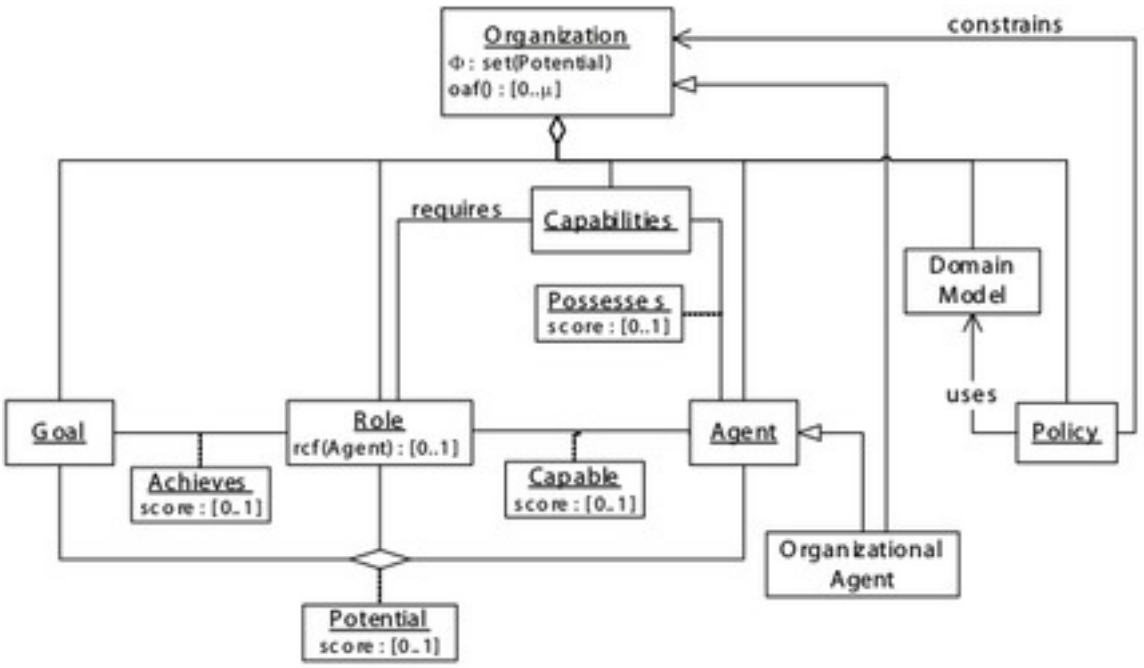

Figure 1.2: OMACS meta-models

OMACS defines an organization as a tuple O = ⟨ G, R, A, C, Φ , P, $\sum$, oaf, achieves, requires, possesses ⟩ where

[2]Agent Group Roles

G : goals of the organization

R : set of roles defines a set of roles (i.e., positions within an organization whose behavior is expected to achieve a particular goal or set of goals)

A : is a set of agents, which can be either human or artificial (hardware or software) entities that perceive their environment

C : set of capabilities which define the percepts/actions at their disposal. Capabilities can be soft (i.e., algorithms or plans) or hard (i.e., hardware related actions)

Φ : relation over G - R - A defining the current set of agent/role/goal assignments

P : set of constraints on Φ formally specifies rules that describe how O OMACS may or may not behave in particular situations

$\sum$: domain model used to specify objects in the environment, their inter- relationships

achieves : function G $\times$ R $\rightarrow$ [0..1] a function whose arguments are a goal in G as well as a role in R that generates an output which is a positive real number greater than or equal to 0 and less than or equal to 1 , defining how effective the behavior defined by the role could be in the pursuit of the specified goal (the extent of achievement of a goal by a role)

requires : function R $\rightarrow$ P(C) a function that assumes a role in R , thereby yielding a set of capabilities required to play that role defining the set of capabilities required to play a role 1

possesses : function A $\times$ C $\rightarrow$ [0..1] a function with an agent in A and a capability in C as inputs yields a positive real number in the range of [0,1] defining the quality of an agent?s capability

OMACS also includes two additional derived functions to help compute potential assignment values: capable and potential.

capable : function A $\times$ R $\rightarrow$ [0..1] function whose inputs are an agent in A OMACS and a role in R OMACS and generates an output, which is a positive real number greater than or equal to 0 and less than or equal to 1 defining how well an agent can play a role (computed based on requires and possesses)

$$\begin{cases} 0 & \text{if} \prod_{c \subset require(r)} posseses(a,c) = 0 \\ \frac{\sum_{c \subset require(r)} posseses(a,c)}{|require(r)|} & \text{else} \end{cases} \tag{1.1}$$

potential : function A $\times$ R $\times$ G $\rightarrow$ [0..1] a function with an agent in A OMACS , a role in R , and a goal in G as inputs yields a positive real number in the range of [0..1] , thus

yielding

$$potential(a, r, g) = achieves(r, g) \times capable(a, r) \tag{1.2}$$

defining how well an agent can play a role to achieve a goal (computed based on capable and achieves)

OAF : function (G × R × A) → [0.. ∞] he selection of φ from the set of potential assignments is defined by the organization?s reorganization function , oaf, that assumes a set of assignments in φ, thereby yielding a positive real number in the range of [0.. ∞] defining quality of a proposed assignment set

$$OAF = \sum_{(a,r,g)\in\Phi} potential(a, r, g) \tag{1.3}$$

1.2.2 Main Element of OMACS

The first eight elements in the organization tuple defined above G, R, A, C, Φ , P, $\sum$ and oaf constitute the main elements of the OMACS model as depicted in Figure 1.2.

1.2.2.1 Goals

are generally defined as a desirable situation (Russell & Norvig, 2003) or the objective of a computational process (van Lamsweerde, Darimont & Letier, 1998).

In OMACS, each organization has a set of goals, G, that it seeks to achieve. OMACS makes no assumptions about these goals except that they can be assigned and achieved by one, and only one agent playing an appropriate role in the organization (i.e., agents do not share goals).

1.2.2.2 Roles

Each OMACS organization also has a set of roles (R) that it can use to achieve its goals. A role defines a position within an organization whose behavior is expected to achieve a particular goal or set of goals. These roles are analogous to roles played by actors in a play or by members of a typical corporate structure. Each OMACS role has a name and a function that defines how well an agent can play the role based on the capabilities possessed by that agent. This function is termed the role capability function, rcf[3]: A × R → [0..1] . The rcf is defined at design time for each role and computed in terms of the capabilities required to play that role. A default rcf (as shown in 1.1) would assume that all the capabilities required to play a role r are equally important, essentially taking the average of all the required possesses values (possesses(a,c)) (with the stipulation that none of those possesses scores are 0).

[3]the same capable function

1.2.2.3 Agents

Each OMACS organization also has a set of heterogeneous agents (A), which are assumed to be computational systems (human or artificial) that sense and act autonomously in a complex dynamic environment in order to realize a set of assigned goals. Thus, we assume that agents exhibit the attributes of autonomy, reactivity, pro-activity, and social ability (Russell & Norvig, 2003). To be part of an OMACS organization, agents must have the ability to communicate with each other (which requires a share ontology and communication language at the organizational level), accept assignments to play roles that match their capabilities, and work to achieve their assigned goals.

1.2.2.4 Capabilities

OMACS uses the capabilities possessed by an agent to determine whether or not it can play a specific role , in OMACS capabilities are atomic entities used to define a skill or capacity of agents. Capabilities can be used to capture soft abilities such as the ability to access certain resources or the knowledge of algorithms, or hard abilities such as sensors and effectors that are often associated with hardware agents such as robots. The set of capabilities, C, known to an organization is the union of all the capabilities required by roles or possessed by agents in the organization.

$$\forall\ \mathrm{c} : \mathrm{C}\ (\ \mathrm{a} : \mathrm{A}\ \mathrm{c}\ \in \mathrm{possese(a,c)}\ \geq 1 \vee\ \mathrm{r} : \mathrm{R}\ \mathrm{c}\ \in \mathrm{requires(r)}) \tag{1.4}$$

1.2.2.5 The Tuple φ

An assignment set φ is the set of agent-role-goal tuples $\langle$ a,r,g $\rangle$, that indicate that agent a has been as-signed to play role r in order to achieve goal g .

φ is a subset of all the potential assignments of agents to play roles to achieve goals. This set of potential assignments is captured by the potential function (see Equation 1.2), which maps each agent-role-goal tuple to a real value ranging from 0 to 1 representing the ability of an agent to play a role in order to achieve a specific goal.

If $\langle$a,r,g $\rangle \in \varphi$, then agent a has been assigned by the organization to play role r in order to achieve goal g .

The only inherent constraints on φ is that it must contain only assignments whose potential value is greater than zero (Equation 1.5) and that only one agent may be assigned to achieve a goal at a time (Equation 1.6)

$$\Phi \subseteq \{(\mathrm{a,r,g}) \nmid \mathrm{a} \in \mathrm{A} \wedge \mathrm{r} \in \mathrm{R} \wedge \mathrm{g} \in \mathrm{G} \wedge \mathrm{potential(a,r,g)} \geq 0\} \tag{1.5}$$

$$\forall \mathrm{a1,a2:A\ r1,r2:R\ g1,g2:G\ (a1,r1,g1)} \in \Phi \wedge \mathrm{(a2\ r2\ g2)} \in \Phi \wedge \mathrm{a1{=}a2} \tag{1.6}$$

1.2.2.6 OAF

In order to select the best set of assignments to maximize an organizations ability to achieve its goals, OMACS defines an organizational assignment function, or oaf , which is a function over the current assignment set, oaf: $\varphi \rightarrow 0..\infty$. As with the rcf , the selection of assignments may be application specific. Thus, each organization has its own application specific organization assignment function, oaf , which computes the goodness of the organization based on φ. As with the rcf , we can define a default oaf , which is simply the sum of the potential scores in the current assignment set φ.

1.2.2.7 Domain Model

An OMACS domain model, $\sum$, is used to define object types in the environment and the relations between those types. The domain model is based on traditional object oriented class and relation concepts . Specifically, an OMACS domain model includes a set of object classes, each of which is defined by a set of attribute types.

The relations between object classes include general purpose associations as well as generalization-specialization and aggregation.

Relations may also include multiplicities to constrain the number of object classes participating in any given relation.

1.2.3 Function of OMACS

There are three major relations/functions and two derived functions between the eight main elements that provide the power of the OAMCS model: achieves, requires, possesses, capable, and potential.

1.2.3.1 The Achieves

function (although somewhat confusingly named) actually defines how effective an agent could be while playing that role in the pursuit of a specific goal. For instance, if one role requires more resources or better capabilities, it can use a different approach and thus yield better results than a second role that requires fewer resources or capabilities. Providing two different roles to achieve the same goal may provide the organization flexibility in deciding how to actually achieve a given goal.

The value of achieves can be predefined by the organization designer or learned before or during system operation (Odell, Nodine & Levy, 2005). Thus, the OMACS achieves function formally captures the effective- ness of a role in achieving a specific goal by defining a total function from the R $\times$ G to a real value in the range of 0 to 1, achieves: G $\times$ R $\rightarrow$ [0..1]. Thus, by definition, a role that cannot be used to achieve a particular goal must have an achieves value of 0, while a role that can achieve a goal would have an achieves value greater than zero.

1.2.3.2 Requires

In order to play a given role, agents must possess a sufficient set of capabilities that allow the agent to carry out the role and achieve its assigned goals.

Thus OMACS defines the requires function, requires: R $\rightarrow$ P(C), which allows an organization designer to define the minimal set of capabilities agents must possess in order to play that role.

1.2.3.3 Possesses

In order to determine if some agent has the appropriate set of capabilities to play a given role, OMACS defines a similar relation, the possesses relation, that captures the capabilities a specific agents actually possesses.

The possesses relation is formally captured as a function over agents and capabilities that returns a value in the range of 0 to 1, possesses: A $\times$ C $\rightarrow$ [0..1]. The real value returned by the possesses function indicates the quality of each capability possessed by the agent; 0 indicates no capability while a 1 indicates a high quality capability.

1.2.3.4 Capable

Using the capabilities required by a particular role and capabilities possessed by a given agent, we can compute the ability of an agent to play a given role, which we capture in the capable function. The capable function returns a value from 0 to 1 based on how well a given agent may play a specific role, capable: A $\times$ R $\rightarrow$ [0..1] .

Since the capability of an agent, a, to play a specific role r , application and role specific, OMACS provides the rcf defined in the previous section that controls how this value is computed. Thus, the capable score of an agent playing a particular role is defined via the designer defined rcf of each role.

$$\forall a : A \text{ r:R } capable(a, r) = r.rcf(a) \tag{1.7}$$

While the rcf is user defined, it must conform to one OMACS constraint. To be capable of playing a given role in the current organization, an agent must possess all the capabilities that are required by that role .

$$\forall a : A, r : Rcapable(a, r) > 0 \Leftrightarrow requires(r) \subseteq c | possesses(a, c) > 0 \tag{1.8}$$

The main goal of OMACS is to provide a mechanism to assign goals to agents in such a way that agents cooperate toward achieving some top-level goal. Intuitively, this mechanism should provide a way to assign the best agents to play the best roles in order to achieve these goals. Thus, OMACS has defined a potential function that captures the ability of an agent to play a role in order to achieve a specific goal.

1.2.3.5 Potential

The potential function maps each agent-role-goal tuple to a real value ranging from 0 to 1, potential: A × R × G → [0..1]. Here, a 0 indicates that the agent-role-goal tuple cannot be used to achieve the goal while a non-zero value indicates how well an agent can play a specific role in order to achieve a goal. The potential of agent a to play role r to achieve goal g is defined by combining the capable and achieves functions.

$$\forall a : A \text{ r:R g:G } potential(a, r, g) = achieves(r, g) * capable(a, r) \tag{1.9}$$

1.3 Organization & Reorganization Dynamic System

$$\forall a : A \text{ r:R g:G } potential(a, r, g) = achieves(r, g) * capable(a, r) \tag{1.10}$$

The constraints above define the legality of the organization structure and its instances. However,we are also interested in whether or not an assignment of agents to roles satisfying all the organizational policies exists that can allow the system to achieve its goals, which we refer to as organizational viability. Although an organization may be structurally valid, there is no guarantee that an instance of that organization exists that can achieve its goals. In actuality, we can never guarantee that the system will ever achieve all its goals due to the dynamic nature of the environment in which the organization operates. To achieve the organizational goals, the system must have the right mix of agents to play the right roles to achieve those goals. Essentially, a viable organization is a valid organization that has been populated with the right types and numbers of agents so that it might potentially achieve its goals.

1.3.1 Organization and Reorganization

Each organization has an implicitly defined organization transition function that describes how the organization may transition from one organizational state to another over the lifetime of the organization. Since agents in an organization as well as their individual capabilities may change over time, this function cannot be predefined, but must be computed based on the current state, the goal set, G, and the current policies. In our present research with purely autonomous systems, we have only considered reorganization that involves the state of the organization. However, we have defined two distinct types of reorganization: state reorganization, which only allows the modification of the organization state, and structure reorganization, which allows modification of the organization structure (and may require state reorganization to keep the organization consistent). We define the state of the organization as the set of agents, A, the possesses, capable, and potential functions, and the assignment set, φ. However, not all these components may actually be under the control of the organization. For our purposes, we assume that agents may enter or leave organizations or relationships, but that these actions are triggers that cause

reorganizations and are not the result of reorganizations. Likewise, possesses (and thus capable and potential as well) is an automatic calculation that determines the possible assignments of agents to roles and goals in the organization. The calculation of possesses is the only calculation totally controlled by the agent; the organization can only use this information in deciding how to make assignments. This leaves one element that can be modified via state reorganization: φ.

1.3.1.1 Goal Set Changes

Any change in G may cause reorganization. There are three basic types of events that can cause achange in G: (1) insertion of a new goal, (2) goal achievement, and (3) goal failure. Each of these is discussed below. The first situation deals with new goals being added to G. However, we cannot say with certainty that reorganization will occur based on a new goal in G. It is possible that the organization will choose to forego reorganization for a number of reasons, the most likely being that it has simply chosen not to pursue any new goals added to G at the present time.The second case deals with goal achievement. When a goal g is achieved, G is changed to reflect that event by (1) removing g from G and (2) possibly adding new goals, which are enabled by the achievement of g, into G. Obviously, the agent assigned to achieve goal g is now free to pursue other goals. The third instance involves goal failure, which really has two forms: agent-goal failure and goal failure. When a specific agent cannot achieve goal g but g might still be achievable by some other agent, agent-goal failure occurs. When agent-goal failure occurs, reorganization must occur to allow the organization to (1) choose another agent to achieve g, (2) not pursue g at the current time, or (3) choose another goal to pursue instead of g. In any of these situations, g is not removed from G since it has not been achieved. In the case where the organization or the environment has changed such that a goal g can never be achieved, then goal failure occurs. In this case, g is removed from G and the organization must attempt to assess whether it can still achieve the overall system goals. Reorganization may occur to see if the agent assigned to achieve g can be used elsewhere. In all cases, the selection of the appropriate strategy is left to the organization.

1.3.1.2 Agent Changes

The second type of change that triggers reorganizations are changes to the set of agents, A, or their individual capabilities. When an agent that is part of φ is removed from the organization, a reorganization must occur, even if only to remove the agent and its assignment(s) in φ. Likewise,when an agent that is part of φ loses a capability that negates its ability to play a role that it is assigned, reorganization must occur as well. In general, when changes occur in an agent?s capability, reorganization may or may not be necessary, based on the agent?s capable relation. We have identified four specific types of changes in an agent?s capabilities that may indicate a need for reorganization: (1) when an agent gains the ability to play a new role, (2) when an agent loses the ability to play

a role, (3) when an agent increases its ability to play a specific role, or (4) when an agent decreases its ability to play a specific role. While case 2 requires reorganization if the agent is currently assigned to play the role for which it no longer has the capability to play, whether or not to reorganize is left up to theorganization when the other three cases (along with 2 when the agent is not currently assigned that role) occur.

1.3.2 Reorganization

Reorganization is the process of changing the assignments of agents to roles to goals as specified in φ. The organization?s oaf function is used to determine the best new φ; however, total reorganization may not be necessary or efficient. (In the absence of any information or policies, an optimal total reorganization would take on the order of 2 A×G×R .) One approach is to take a local view, in which the organization looks at the OMACS state and reorganizes in a locally optimal fashion (i.e. hill climbing). However, when dealing with dynamic environments, it is often desirable to reorganize so that the team can operate more efficiently or effectively in its present situation as well as being adaptable to its changing environment. Thus, we would like to take a long-range or global view. Unfortunately, it has been shown that in the general case globally optimal reorganizations are NEXP-complete and, thus impractical for most applications with any time constraints . Therefore, OMACS provides a mechanism for augmenting the locally optimal algorithm with application specific rules in an attempt to make reasoning more efficient and to enable globally better solutions.

1.3.2.1 General Purpose Reorganization Examples

For general-purpose reorganization, the Developers have developed several reorganization algorithms that give us a default reorganization capability. When a reorganization trigger occurs, general-purpose reorganization algorithms can be used to find appropriate assignments to achieve the organizations goals, if possible.

To compute the best reorganization, an algorithm that simply optimizes the organizations oaf might seem appropriate; however, this approach is short sighted. First, it does not deal with the cost associated with reorganizing and, second, it does not consider the reason reorganizing was initially undertaken. Exploiting reorganizing costs requires a distributed solution since the cost for robots to change roles is not globally known.

For instance, if an agent is required to perform a complex computation, any effort toward that computation would be lost if the agent was reassigned to another role/goal Considering the reason for reorganization may enable less extensive (and less costly) reorganization. If the reason for reorganizing is to fill a single role, then a total reorganization may be a waste of time and resources.

1.4 Conclusion

This chapter describes how to design adaptive multiagent systems using an organizational model, which defines the entities and relationships of a typical organization.

The major elements of the model consist of goals, roles, agents, capabilities, and the relationships between them. By designing a system using the model, and we focus on OMACS framework because it handeling reorganization system more then other framework

In the following chapter will discuss other model we can use it to modeling the multi-Agent System

Chapter 2

PGraph and PNS

In a process system , raw materials are consumed through various transformation to yield desired products . This is usually accompanied bu the generation of wastes .Vassels in which these transformations are carried out are termed operating units of the process , a given set of operating units with the plausible interconnections can be described bu a network The desired products can often be manufactured using some sub networks of this network , thus a given network may give rise to a variety of processes producing the desired products , and each of such process corresponds to sub network which can be considered to be its structure

2.1 PGraph

The so called P graph (Process graph), which is a directed bigraph, has been used for modelling network structures for some time. The vertices of the graph denote the operating units (O operating units) and the materials (M materials). The edges of the graph represent the material-flow between the materials and the operating units.

2.1.1 General Definition of P-Graph

The Pgraph is a bigraph, meaning that its vertices are in disjunctive sets and there are no edges between vertices in the same set. In case of P graphs the assignment of operating units and materials are strictly determined by the tasks given, i.e. an edge can point to an M material type vertex from an O operating unit type vertex, only if M is element of the output set of O, that is O produces M material namely, M $\in$ output O .

An edge can point from an M material type vertex to an O operating unit type vertex, only if M is element of the input set of O, that is O processes M material, namely M $\in$ input O. . Thus, the P-graph can be presented by the pairs of operating unit and the assigned material vertices set like the (M,O) P-graph. The material type vertices can be put into several subsets. There are various subsets like the raw-material type one, which contains the input elements of the whole process, the product-material type subset, which gathers the results of the entire process, the intermediate-material type one, the elements

of which emerge or are used between the processing phases, and finally the by-product-material type set, which contains the non desired results of the process. The applied operating unit and material element notations in the P-graph notation are presented in 2.2.

As an example let us consider a process network with 7 operating units, in which the operating units are 1,2....7 and the materials are A,B, ... L. A,B,C and D are the materials available for the production of L The possible structure is given in .

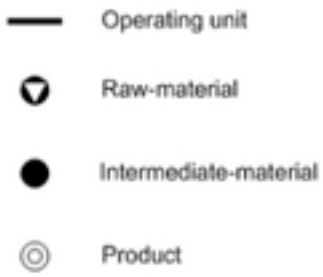

Figure 2.1: notation in pgraph

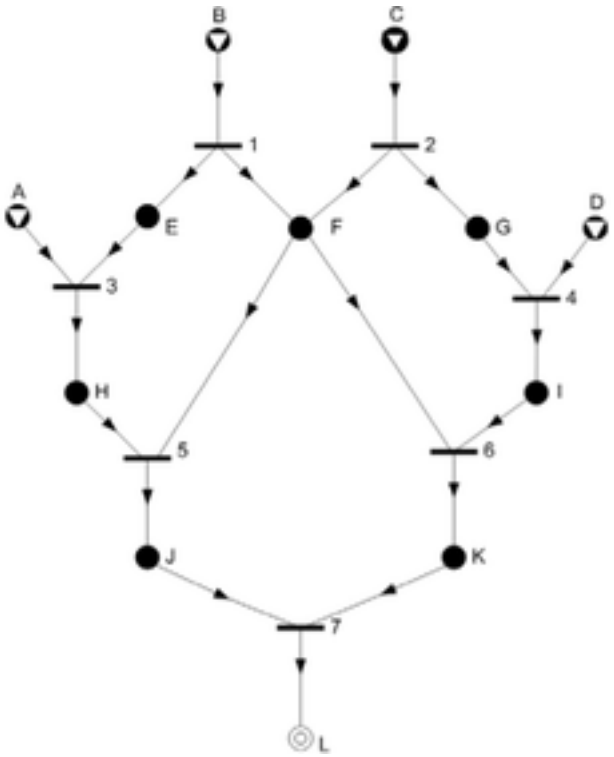

Figure 2.2: example for P-graph

2.2 Process Network Synthesis

In the Process Network Synthesis (PNS for short) problem a set of materials is given and also operating units which are transforming some subset of materials into some other subset. The subsets assigned to the operating unit are called its input and output materials. In the problem two subsets of the materials are distinguished, one is the set of the raw

materials and the other is the set of the desired products. Our goal is to find a minimal cost network of the operating units which can produce all desired products starting from the raw material. These systems can be modeled in the P-graph framework which is based on bipartite graphs.

In these P-graphs we have two sets of vertices, one of them contains the possible materials, the other the operating units. The edges lead to an operating unit from its input materials and from an operating unit to its output materials.

Then the subgraphs satisfying some properties describe the feasible processes which produce the desired products from the raw materials. Thus our goal is to find the least expensive such subgraph. In the structural model the amounts of the material flows are not taken into account thus the cost of an operating unit is a constant, and the cost of a subgraph is the sum of the costs of the operating units contained in it.

2.2.1 Basic Notation

The structural PNS problem can be modeled in the PGraph framework. In the PGraph (Process Graph) we have the set of the materials denoted by M, which contain two special subsets, the set of raw materials and the set of desired products denoted by R and P respectively

The problem also contains a set of possible operating units which can transform some sets of materials.

The set of operating units is denoted by O. An operating unit u is given by two sets, in(u) denotes the set of the input materials out(u) denotes the set of output materials of the operating unit

This means that the operating unit can work in a solution structure if all of its input materials are produced and in this case it produces all of its output materials.

The PGraph of the problem is defined by the sets M and O. It is a directed bipartite graph where the set of vertices is $M \cup O$, and have the following two sets of edges:

1. Edges which connect the input materials to their operating unit
2. Edges which connect the operating units to their output materials

Then some of the subgraphs of this P-graph describe the feasible solutions which produce the required materials from the raw materials. where m and o are the subsets of M and O, represent a feasible solution if and only if the following properties called axioms are valid:

1. m contain all element of P
2. a material from m is a raw material if and only if no edge goes into it in the P-graph (m, o)
3. For each operating unit u from o there exists a path in the P-graph (m,o) which goes into a desired product from u

4. m is the union of the input and output material sets of the operating units contained in set o

2.2.2 Mathematical definition

There is a finite set of material M (which contains the sets of P products and R raw-materials) and the finite set of O operating units. Consequently, the set of P end-products and the set of R raw-materials must be subsets of M and the set of M materials and the set of O operating units are disjunctive. The basic relations between M,P,R and O are as follows :

$$P \subseteq M, R \subseteq M, M \cup O = \emptyset \tag{2.1}$$

As physical processes are defined, each operation unit produces output materials from input materials. Therefore two disjunctive sets can be assigned to each operating unit, i.e. the set of input and the set of output materials.

Let an arbitrary operating unit (α , β), then α is the set of input materials which are processed by the (α , β) unit and $\beta$is the set of output materials, which are produced by the given unit .

Considering the process-network the output materials of each operating unit are the inputs of different operating units. In general, it can be proved that

$$O \subseteq \rho(M) \times \rho(M) \tag{2.2}$$

where O is the set of operating units, M is the set of materials and $\rho(M)$ is the power set, that is the set of subsets of M, and $\rho(M) \times \rho(M)$ represents the set of $\rho(M)$ and $\rho(M)$ pairs

Supposing that there is a finite set m, which is a subset of M, i.e. it is true that m $\subseteq$ M and there is an o finite set, which is a subset of O, i.e. it is true that o $\subseteq$ O and supposing that there is such a material which is an input for one or more operating units, and there is such material which is the output of one or more operating units, then :

$$o \subseteq \rho(m) \times \rho(m) \tag{2.3}$$

The PNS is defined as a bigraph, where the set of V vertices is made of the elements of the union of m and o that is

$$V = m \cup o \tag{2.4}$$

2.2.3 Algorithms MSG, SSG, and ABB

PNS representation of a process network and the set of axioms for solution structures, i.e., combinatorial feasible networks, render it possible to fashion the three mathematically

rigorous algorithms: MSG, SSG, and ABB.

The algorithm MSG (Maximal-Structure Generation) generates the maximal structure (super-structure) of a process synthesis network. Also,

the algorithm SSG (Solution-Structure Generation) generates the set of feasible process structures from the maximal structure,

which leads to the algorithm ABB (Accelerated Branch and Bound) for computing the n-best optimal solution structure

2.2.3.1 Maximal-Structure Generation

The maximal structure of the synthesis problem (P, R, O) contains all the combinatorially possible structures, which make the production of defined products possible from given raw-materials. Therefore, it certainly contains the optimal structure as well.

The first phase is the input phase, in which the synthesis problem is defined (P, R, O) such a way, that the set of M all the plausible materials, the set of P end-products

The second phase is the elaboration of the input structure of the network, which is carried out by the linking of all the similar (same type) material type vertices.

The third phase is the elimination phase, where those materials and operating units are eliminated, which, taking the axioms into account, are not and cannot be linked to the maximal structure for sure

During the fourth phase the vertices are linked again from level to level, starting from the highest, the end-product level.

The maximal structure generated this way contains all the combinatorially possible structures and all of its elements fulfil the axioms.

2.2.3.2 Maximal-Structure Generation

The maximal structure generated by the MSG algorithm contains all such combinatorically possible network structures that are able to produce the end-product from the given raw-materials.

Consequently, it contains the optimal network as well. In most cases the optimalisation means to find the most cost effective solution.

The application of the SSG (Solution Structure Generation) algorithm enables the production of all the solution structures. The SSG is a new mathematical tool which has been developed by Friedler et al.

2.2.3.3 Accelerated Branch and Bound

The branch-and-bound method has been widely used The method is reiterated here to facilitate formalization of the accelerated branch-and-bound algorithm of PNS. to Identify the optimal process from the maximal structure

2.2.4 Comparaison PNS and PetriNet :

Petri Net	**Process Network Synthesis**
Source Place	Raw Material
Normal Place	Intermediate Material
Sink Place	Final Product
Token in Place	Requirement Flow
Transition	Operating Unit
Weight of in or out edges of the transition	producing rate of the operating unit
Modeling Parallel System	Basically use for Chemical Reaction

Table 2.1: Petri Net and PNS

2.3 Conclusion

in this chapter , i presente the definition of second framework i use PGraph and define the Process Network synthesis . with some Algorithme applicable on PNS to find the optimum structure