Multi
Agent
Cooperative
Robotics
Laboratory

Kansas State University
234 Nichols Hall
Manhattan, KS 66506-2302

Phone: (785) 532-6350
Fax: (785) 532-7353
http://macr.cis.ksu.edu/

Guidance and Law Policies in Multiagent Systems[*]

Scott J. Harmon, Scott A. DeLoach, and Robby
{harmon, sdeloach, robby}@ksu.edu

MACR-TR-2007-02

March 17, 2007

Abstract

Policies have traditionally been a way to specify properties of a system. In this paper, we show how policies can be applied to Organization-based Multiagent Systems Engineering (O-MaSE) [6], specifically, in the OMACS meta-model. In OMACS, policies may constrain assignments of agents to roles, the structure of the goal model for the organization, or how an agent may play a particular role. We also show how traditional policies can be characterized as law policies*.* Law policies *must always be followed by a system. Because of this inflexibility,* law policies *may constrain a multiagent system too much. In order to preserve flexibility of the system, while still being able to guide the system into preferring certain behaviors, we introduce the concept of* guidance policies*. These so-called* guidance policies *need not always be followed. When the system cannot continue with the* guidance policies*, they may be suspended. We show how this can increase performance while not decreasing flexibility of the system to adapt.* Guidance policies *are formally defined and, since multiple* guidance policies *can introduce conflicts, a strategy for resolving conflicts is given.*

1 Introduction

As Autonomic computing has become more prevalent, work has been done to aid in the construction of systems built from autonomous agents. This has led to the use of Multiagent systems, and in turn, Multiagent system engineering. Work on designing formalisms and methodologies has been done to help software engineers design multiagent systems. One aspect of multiagent systems that has been considered is policies. Policies allow one to describe properties of a multiagent system–whether that be behavior or some other design constraints. To have self-managing systems, policies must be able to be specified and tested [10]. Policies have traditionally been properties that must always hold. This does not completely follow the notion of policies in human organizations. Sometimes, policies cannot be followed. When a policy cannot be followed in a multiagent system, the system cannot achieve its goals and thus it cannot continue to perform. In human organizations, a policy may be temporarily suspended in order to allow the system to proceed. This can also be very important in a system built of autonomous agents because of the autonomy itself. We would like to guide the system, but not constrain it too much so that it cannot function or be autonomous.

The contributions in this paper are as follows: 1. We give a formal definition of both *law* and *guidance* policies, showing how traditional policies may be viewed as *law policies*; 2. We present a conflict resolution strategy for guidance policies and the intuition behind it; and 3. We present experimental validation of this approach through simulation. *Guidance policies* can have a great impact by allowing designers to better give the system information about the way they want it to operate. Guidance policies can possibly be used in place of most *law policies*, save policies concerning *safe or secure* operation of a system. These *guidance* policies come with the benefits of the directing force given by *law policies*, but without sacrificing the flexibility of the system to adapt to a changing environment and thus be self-maintaining.

The rest of the paper is organized as follows, we first give some background in policies for multiagent systems. Next, we present a multiagent system example. We define the notion of *system traces* for a multiagent system, which are then used to describe policies. A short description of our language for policies and an example policy is then presented. After which, we define *law policies* as well as *guidance policies*. We give examples and show how *guidance policies* are useful for multiagent systems. A method for ordering guidance policies according to importance and a formal explanation of what is meant by the ordering is given. Experimental results from applying policies to a multiagent system are then presented and analyzed. Conclusions are made and then we present some ideas for future work.

[*]This work was supported by grants from the US National Science Foundation (0347545) and the US Air Force Office of Scientific Research (FA9550-06-1-0058).

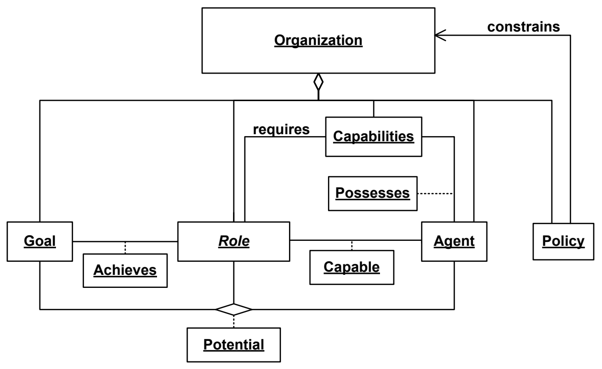


Figure 1. OMACS Organization Model.

2 Background

Policies have been considered for multiagent systems for some time. Efforts have been made to characterize, represent, and reason [1] about policies in the context of multiagent systems. There has also been work on detecting global properties [19] of a distributed system, which could in turn be used to suggest policies for that system. Policies have been proposed as a way to help assure that agents and that the entire multiagent system behave within certain boundaries. Policies have also been proposed as a way to specify security constraints in multiagent systems [9, 14].

It should be noted that policies have been referred to as laws in the past. Yoav Shoham and Moshe Tennenholtz wrote in [17] about *social laws* for multiagent systems. They showed how policies could aid a system in working together, similar to how our rules of driving on a predetermined side of the road help the traffic to move smoothly.

The model we will use for this paper is called the Organization Model for Adaptive Computational Systems (OMACS) [7]. Figure 1 gives a graphical depiction of the OMACS organization model. OMACS defines standard multiagent system components such as goals, roles, capabilities, and agents. Roles can *achieve* goals, agents can *posses* capabilities, and agents can be *capable* of playing roles depending on what capabilities they *posses*. The organization, which represents the entire set of agents, decides which agents to *assign* to what roles to *achieve* particular goals. If an agent is *capable* of playing a role and that role *achieves* a particular goal, then there is a *potential* assignment of that agent to play the role to achieve the goal. When the organization makes an *assignment* of an agent to a particular role, in order to achieve a specific goal, the organization will be constrained by agents capabilities as well as any applicable policies. Goals can be *triggered* (become active) during an agent's activity while playing a role. Only active goals may be assigned along with a role to an agent.

We may observe events in an OMACS system. A *system event* is simply an action taken by the system. In this paper we will be concerned with specific actions that the organization takes. For instance, an assignment of an agent to a role is a system event. The completion of a goal is also a system event. In an OMACS system we can have the following system events of interest:

Event	Definition
$C(g_i)$	goal g_i has been completed(achieved).
$T(g_i)$	goal g_i has been triggered.
$A(a_i, r_j, g_k)$	agent a_i has been assigned role r_j to achieve goal g_k.

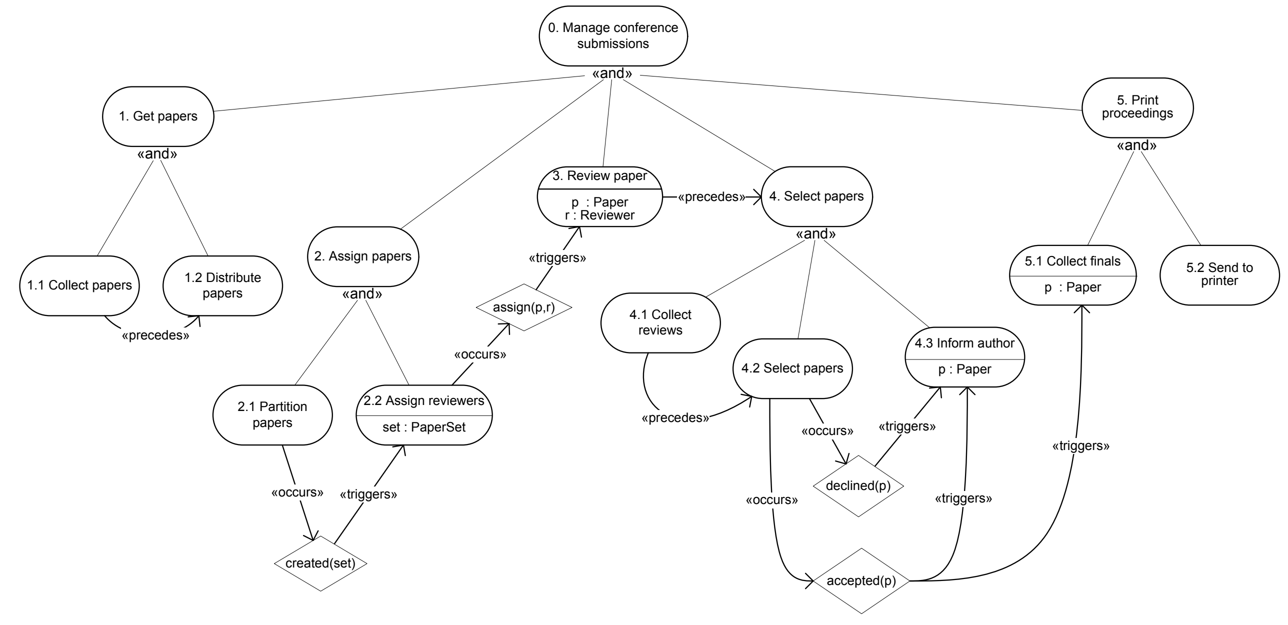


Figure 2. Conference Management Goal Model.

2.1 Conference Management Example

A well known example in multiagent systems is the Conference Management [20, 5] example. The Conference Management example models the workings of a scientific conference, in which, authors submit papers, reviewers review the submitted papers, certain papers are selected for the conference and printed in the proceedings. Figure 2 gives the complete goal model for the conference management example, which we will use to illustrate our policies. In this example, a multiagent system represents the goals and tasks of a generic conference paper management system. Goals of the system are identified and are broken down into subgoals.

The top level goal, *0. Manage conference submissions*, is decomposed into several '*and*' subgoals, which means that in order to achieve the top goal, the system must achieve all of the '*and*' subgoals. These subgoals are then associated through precedence and trigger relations. The *precedes* arrow between goals indicates that the source of the arrow must be *achieved* before the destination can become active. The *triggers* arrow indicates that the domain specific event in the source may trigger the goal in the destination. The *occurs* arrow from a goal to a domain specific event indicates that while playing a role to achieve that goal, said event may occur. A goal that triggers another goal may trigger multiple instances of that goal when the triggering goal is being worked on.

Leaf goals are goals that have no children. The leaf goals in this example consist of *Collect papers*, *Distribute papers*, *Partition papers*, *Assign reviewers*, *Collect reviews*, *Select papers*, *Inform author*, *Collect finals*, and *Send to printer*. For each of these leaf goals to be achieved certain roles are required to be played.

The roles required to achieve the leaf goals are depicted in Figure 3. The role model gives seven roles as well as two outside actors. Each role contains a list of leaf goals that the role can achieve. For example, the *Assigner* role can achieve the *Assign reviewers* leaf goal. In OMACS, roles only achieve leaf goals. The arrows between the roles indicates interaction between particular roles. For example, once the agent playing the *Partitioner* role has some partitions, it will need to hand off these partitions to the agent playing the *Assigner* role. OMACS allows for the same agent to play multiple roles at once, as long as they have the capabilities required by the role, and are allowed by the policies.

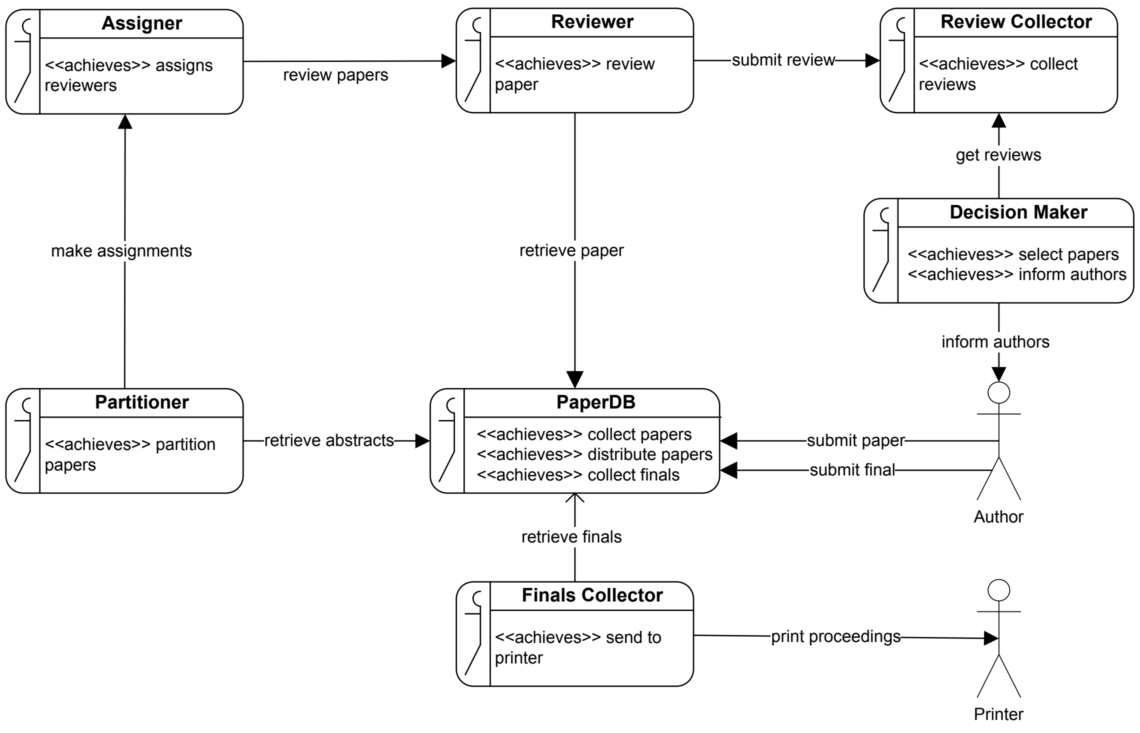

Assigner
<<achieves>> assigns reviewers
Reviewer
<<achieves>> review paper
Review Collector
<<achieves>> collect reviews
review papers
submit review
get reviews
Decision Maker
<<achieves>> select papers
<<achieves>> inform authors
make assignments
retrieve paper
inform authors
Partitioner
<<achieves>> partition papers
retrieve abstracts
PaperDB
<<achieves>> collect papers
<<achieves>> distribute papers
<<achieves>> collect finals
submit paper
submit final
Author
retrieve finals
Finals Collector
<<achieves>> send to printer
print proceedings
Printer

Figure 3. Conference Management Role Model.

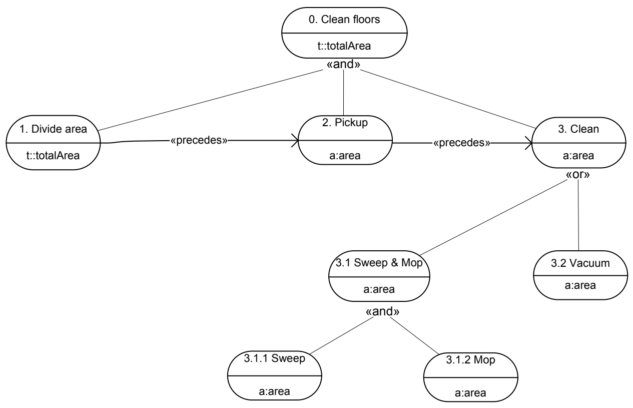


Figure 4. CRFCC Goal Model.

2.2 Robotic Floor Cleaning Example

Another example to illustrate the usefulness of the concept of guidance policies is the Cooperative Robotic Floor Cleaning Company Example (CRFCC). This example was first presented by DeLoach et al. in [16]. In this example, a team of robotic agents clean the floors of a building. The team has a map of the building as well as indications of whether a floor is tile or carpet. Each team member will have a certain set of capabilities (e.g. vacuum, mop, etc). These capabilities may become defective over time. In their analysis, DeLoach et al. showed how breaking up the capabilities affected a team's flexibility to overcome loss of capabilities. We have extended this example, giving the information that the vacuum cleaner's bag needs to be changed after vacuuming three rooms. Thus, we would then like to minimize the number of bag changes. For this we will introduce a guidance policy and show how it affects the performance of our organization.

The goal model for the CRFCC system is fairly simple. As seen in Figure 4, the overall goal of the system (Goal 0) is to clean the floors. This goal is then broken into three conjunctive subgoals: *1. Divide Area*, *2. Pickup*, and *3. Clean*. The *3. Clean* goal is then broken into two disjunctive goals: *3.1 Sweep & Mop* and *3.2 Vacuum*. Depending on the floor type, only one will be needed to accomplish the *3. Clean* goal. If an area needs to be swept and mopped (i.e. it is tile), then goal *3.1 Sweep & Mop* is broken into two conjunctive goals: *3.1.1 Sweep* and *3.1.2 Mop*. After an agent achieves the *1. Divide area* goal, a certain number of *2. Pickup* goals will become active (depending on how many pieces the area is divided into). After the *2. Pickup* goals are completed, a certain number of *3. Clean* goals become active, again depending on how many pieces the area was broken into. This then will activate goals for the tile areas (*3.1.1 Sweep* and *3.1.2 Mop*) as well as goals for the carpeted areas (*3.2 Vacuum*).

Figure 5 gives the role model for the CRFCC. In this role model, each leaf goal of the system is achieved by a specific role. The role model may be broken down many different ways depending on the system's goal model as well as agent and capability models. Thus, depending on the agents and capabilities available the system designer may choose different role models. For this paper, we will look at just one of these role models. In the role model in Figure 5, the only role requiring more than one capability is the *Pickuper* role. This role will require both the *search* and *move* capability. Thus an agent, in order to play this role, must possess both capabilities.

2.3 Properties of Interest

At any stage in a multiagent system, there may be certain properties of interest. Some may be domain specific (only relevant to the current system), and others may be system properties such as the number of roles an agent is

Role Name	Req. Capabilities	Goals Achieved
Organizer	org	1. Divide Area
Pickuper	search, move	2. Pickup
Sweeper	sweep	3.1.1 Sweep
Mopper	mop	3.1.2 Mop
Vacuummer	vacuum	3.2 Vacuum

Figure 5. CRFCC Role Model.

currently playing. State properties that are relevant to our examples are as follows:

Property	Definition
$a.reviews$	the number of reviews agent a has performed.
$a.vacuumedRooms$	the number of rooms agent a has vacuumed.

2.4 System Traces

In order to describe multiagent system execution, we use the notion of a *system trace*. A system trace is a projection of system execution with only desired state and event information preserved (role assignments, goal completions, domain specific state property changes, etc). In this paper, we will only be concerned with the events and properties given above and only traces that result in a successful completion of the system goal. Let E be an event of interest and P be a property of interest. A change of interest in a property is a change for which a system designer has made some policy. For example, if a certain integer should never exceed 5, a change of interest would be when that integer became greater than 5 and when that integer became less than 5. Thus a change of interest in a property is simply a projection of all the changes in the property. ΔP indicates a change of interest in property P. A system trace may contain both events and changes of interest in properties. Changes of interest in properties may be viewed as events, but for simplicity we will include both and use both interchangeably. Thus, a system trace is defined as:

$$[E|\Delta P] \rightarrow [E|\Delta P] \rightarrow \ldots \quad (1)$$

As shown in equation 1, a trace is simply a sequence of events/changes of interest.

Using our conference management example, a subtrace of the system, where g_1 is a goal, a_1 is an agent, and r_1 is a role, might be:

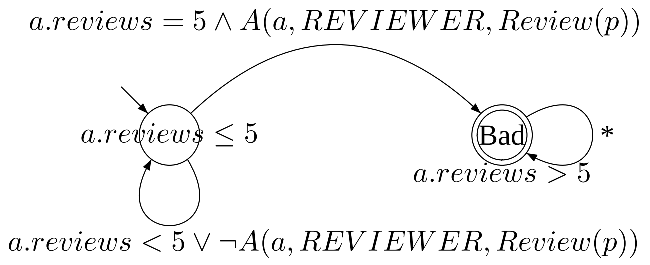
$$\ldots T(g_1) \rightarrow A(a_1, r_1, g_1) \rightarrow C(g_1) \ldots \quad (2)$$

Formula 2 means that goal g_1 is triggered, then agent a_1 is assigned role r_1 to achieve goal g_1, finally, goal g_1 is completed.

We will be using the terms *legal trace* and *illegal trace*. An *illegal trace* is an execution we do not want our system to exhibit, while a *legal trace* is an execution that our system may exhibit. Intuitively, our policies will cause some traces to become *illegal*, while others will remain *legal*.

3 Policies

Policies may restrict or proscribe behavior of a system. An assignment set is a snapshot of the current assignments of agents to roles in a system. Policies concerning agent assignments to roles have the effect of constraining the set of possible assignments. This can greatly reduce the search space when looking for the optimal assignment set [21].


$a.reviews = 5 \wedge A(a, REVIEWER, Review(p))$
$a.reviews \leq 5$
Bad
$*$
$a.reviews > 5$
$a.reviews < 5 \vee \neg A(a, REVIEWER, Review(p))$

$\forall a : Agents, p : Papers$

Figure 6. No agent may review more than five papers.

Other policies could be used for verifying that a goal model meets certain criteria. This allows for an easier statement of properties of the goal model that may be verified against candidate goal models at design time. For example, we might want to ensure that our goal model in Figure 2 will always trigger a *Review Paper* goal for each paper submitted.

Yet, other policies may restrict the way that roles can be played. For example, *when an agent is moving down the sidewalk it always keeps to the right.* These behavior policies also restrict how an agent interacts with its environment, which in turn means that they can restrict protocols and agent interactions. One such policy might be that an agent playing the *Reviewer* role must always give each review a unique number. These sort of policies rely heavily on domain specific information. This is why it is important to have an ontology for relevant state and event information prior to designing policies [8].

3.1 Language for policies

To describe our policies, we use first order logic over our predicates along with temporal operators. This may be converted into Linear Temporal Logic (LTL) [12] or Büchi automata [2] for infinite system traces, or to something like Quantified Regular Expressions [13] for finite system traces. The formulas consist of predicates over goals, roles, events, and assignments (recall that an assignment is the joining of an agent and role for the purpose of achieving a goal). The temporal operators we currently use are given below:

Notation	Definition
$\Box(x)$	x holds always.
$\Diamond(x)$	x holds eventually.
$x\,\mathcal{U}\,y$	x holds until y holds.

We use a mixture of state properties as well as events [3] to obtain compact and readable policies. An example of one such policy formula is:

$$\forall a_1 : Agents,\ \mathcal{L} : \Box(sizeOf(a_1.reviews) \leq 5) \tag{3}$$

Formula 3 states that it should always be the case that each agent never review more than five papers. The $\mathcal{L}$: indicates that this is a *law policy*. The property $.reviews$ can be considered as part of the system's state information. This is domain specific and allows a more compact representation of the property. In Figure 3.1, we show a finite automata representation of this policy.

The use of the $A()$ predicate in Figure 3.1 indicates an assignment of the *Reviewer* role to achieve the *Review paper* goal, which is parametrized on the paper p. This automata depicts the policy in Formula 3, but in a manner for a model checker or some other policy enforcement mechanism to detect when violation occurs. The accepting state indicates that a violation has occurred. Normally, this automata would be run alongside the system, either at design time with a model checker [4], or at run-time with some policy enforcement mechanism [11].

3.2 Law Policies

The traditional notion of a policy is a rule that must always be followed. We will refer to these policies as *law policies*. These policies usually restrict or proscribe behavior of a system. An example of a law policy with respect to our conference management example would be *no agent may review more than five papers.* This means that our system can never assign an agent to the *Reviewer* role more than five times. A law policy can be defined as:

$$\mathcal{L} : Conditions \rightarrow Property \tag{4}$$

$Conditions$ are predicates over state properties and events, which, when held true, imply that the $Property$ holds true. The $Conditions$ portion of the policy may be omitted if the $Property$ portion should hold in all conditions, as in Formula 3.

Intuitively, for the example above, no trace in the system may contain a subtrace in which an agent is assigned to the *Reviewer* role more than five times. This will limit the number of legal traces in the system. In general, *law policies reduce the number of legal traces for a multiagent system*.

The policy to limit the number of reviews an agent can perform, is helpful in that it will ensure that our system does not overburden any agent with too many papers to review. This policy as a pure law policy, however, could lead to trouble in that the system might no longer be able to achieve its goal. Imagine that more papers than expected are submitted. Since there is no bound on the number of papers submitted, this is a plausible scenario. If there are not sufficient agents to spread the load, the system may fail, since it is not able to assign more than five papers to any agent. This is a common problem with using only law policies. They limit the *flexibility* of the system. Flexibility of a multiagent system has been defined [16] as a measure of how well a multiagent system overcomes failures. We will generalize this definition to *how well the system can adapt to changes*.

3.3 Guidance Policies

Let us look at the policy that limits the number of reviews an agent can perform. Recall from the section on law policies how it was a seemingly useful policy, but had the drawback that our system became less flexible. To overcome this problem, we have defined another type of policy called *guidance policies*. Take for example the policy used above, but as a *guidance policy*:

$$\forall a_1 : Agents,\ \mathcal{G} : \Box(sizeOf(a_1.reviews) \leq 5) \tag{5}$$

This is exactly the same as the policy in Formula 3 except for the $\mathcal{G}$:. The $\mathcal{G}$: indicates that this policy is a *guidance policy*. In essence, the formalization for guidance and law policies are the same, the difference is the intention of the system designer. *Law policies* should be used when the designer wants to make sure that the property the policy specifies is always true (e.g. for safety or security), while *guidance policies* should be used when the designer simply wants to guide the system.

This policy will limit our system to only have an agent review up to five papers, *when possible*. Imagine that the system gets more submissions than expected, in this case the system can still be successful since it will be able to assign more than five papers for review to an agent. In the case where there are sufficient agents, however, the system will limit each agent to five or fewer reviews.

Conflict resolution In the definition of *guidance policies* above, we have not clarified how the system should behave when forced to choose which guidance policy to violate. We propose a partial ordering of guidance policies to allow the system designer to set precedence relationships between different guidance policies. We arrange the guidance policies as a lattice, such that a policy that is a parent of another policy in the lattice, is more important than its children. A system trace can define a set of policies that were violated during that trace. This set of

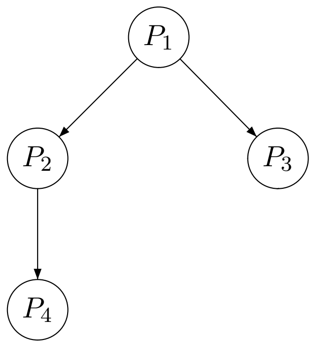


Figure 7. Partial order of Guidance Policies.

violations may be computed by examining the policies and checking for matches against the system trace. When there are two traces that have violations with a common ancestor, and one (and only one) of the traces violate the common ancestor, we mark the trace with the common ancestor as illegal. Intuitively, this trace is illegal because the system had a better option, namely, not violating the more important guidance policy. Thus, if the highest node in each of the two trace is an ancestor of every node in both traces, and that node is not in both traces, then we know the trace with that node is illegal and should not have happened.

Take, for example, the four policies in the table below:

Node	Definition
P_1	PC Chair should not review papers.
P_2	No agent should review more than 5 papers.
P_3	Each paper should receive at least 3 reviews.
P_4	An agent should not review a paper from someone whom they wrote a paper with.

Let these policies be arranged in the lattice shown in Figure 7. The lattice in Figure 7 means that policy P_1 is more important than P_2 and P_3, and P_2 is more important than P_4. Note that we would really like to avoid having the PC Chair review papers, but, if that is the only way to achieve the goals of the system without violating law policies, the PC Chair may review papers. However, if there is any trace that violates any preference policies other than P_1 (and that does not violate a law policy), it should be chosen over one which violates P_1.

When a system cannot achieve its goals without violating policies, it may violate guidance policies. There may be traces that are still illegal, though, depending on the ordering between policies. *For every pair of traces, if the least upper bound of the violations of both traces, let us call this policy violation $\mathcal{P}$, is in one (and only one) of the traces, the trace with $\mathcal{P}$ is illegal.* For example, consider the ordering in Figure 8, let trace t_1 violate P_1 and P_2, while trace t_2 violates P_2 and P_3. Round nodes represent policies violated in t_1, box nodes represent policies violated in t_2, and boxes with rounded corners represent policies violated in both t_1 and t_2. Since P_1 is the least upper bound of P_1, P_2, and P_3 and since P_1 is not in t_2, t_1 is illegal, and thus should not happen.

As shown in Figure 9, the policies may be ordered in such a way that the policy violations of two traces do not have a least upper bound. If there is no least upper bound, $\mathcal{P}$, such that $\mathcal{P}$ is in one of the traces, the two traces cannot be compared and thus both traces are legal. The reason they cannot be compared is that we have no information about which policies are more important, violating the policies in one set may be just as bad as violating the policies in the other set. Thus, either option is legal.

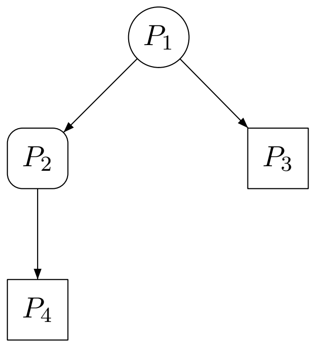


Figure 8. Partial order of Guidance Policies with traces.

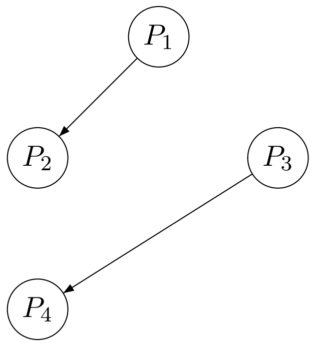


Figure 9. Possible Partial order of Guidance Policies.

One may even have the situation as shown in Figure 8, but where P_1 is not violated by either trace. In this case, the violation sets cannot be compared and thus both traces are legal. In situations such as these, the system designer may want to impose more ordering on the policies.

Intuitively, guidance policies are constraints on a system such that at any given state of the system, if there is a transition enabled that will not violate a guidance policy, this transition is always chosen over a transition that violates a guidance policy. If violation of all guidance policies cannot be avoided, a partial ordering of guidance policies is used to choose which policies may be violated.

4 Experimental Results

Using our CRFCC (Floor cleaning) example and a modified simulation from [16] we have results from running the simulation with the added guidance policy: *no agent should vacuum more than three rooms*. We contrast this with the law policy: *no agent may vacuum more than three rooms*. The guidance policy is presented formally in Equation 6.

$$\forall a_1 : Agents,\ \mathcal{G} : \Box(a_1.vacuumedRooms \leq 3) \tag{6}$$

For this experiment, we used the agent model in Figure 10. These capabilities will restrict what roles our simulator can assign to particular agents. E.g. the Organizer role may only be played by agent a_1 or agent a_5,

Agent	Capabilites
a_1	org, search, move
a_2	search, move, vacuum
a_3	vacuum, sweep
a_4	sweep, mop
a_5	org, mop

Figure 10. CRFCC Agent Model.

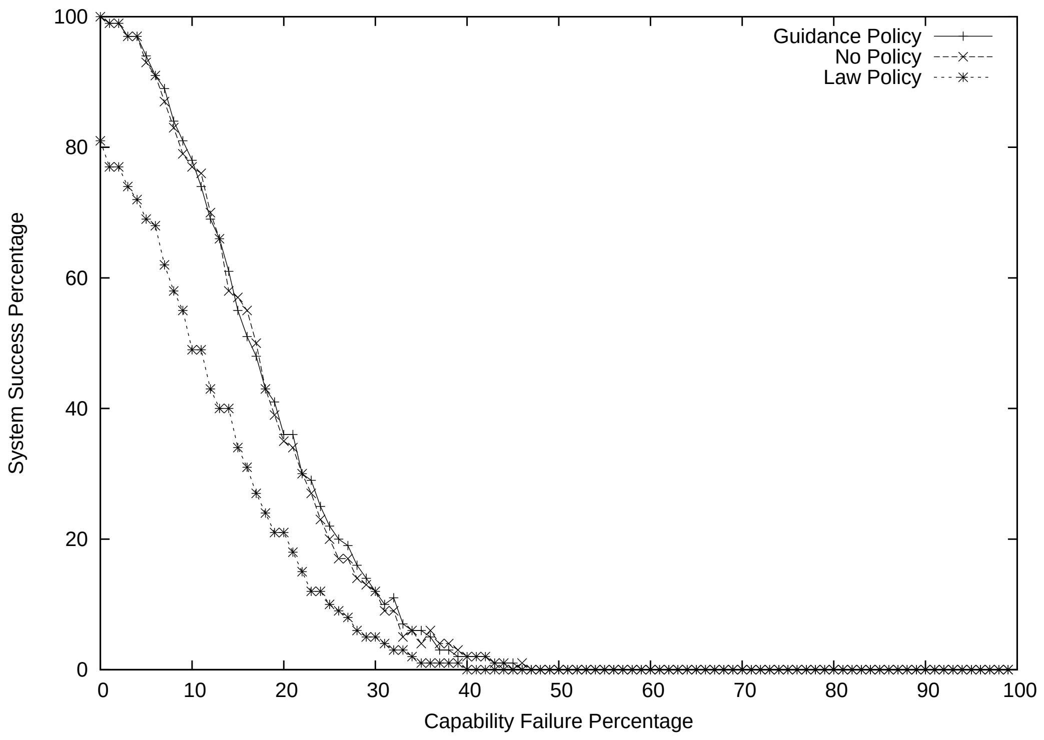


Figure 11. The success rate of the system given capability failure.

since those are the only agents with the *org* capability. In the simulation we will be choosing randomly capabilities to fail. This capability will fail with the probability given by the capability failure rate.

For each experiment, the result of 1000 runs at each capability failure rate was averaged. In the simulation, at each step, a goal that is being played by an agent is randomly achieved. Using the capability failure rate, at each step, a random capability from a random agent may be selected to fail. Once a capability fails in the simulation, it can never return.

In Figure 11, it can be seen that while the system success rate decreases when we enforce the law policy, it does not, however, decrease when we enforce the guidance policy. Guidance policies *do not decrease the flexibility of a system to adapt to a changing environment*, while law policies *do decrease the flexibility of a system to adapt to a changing environment*.

Figure 12 gives the total number of times the system assigned vacuuming to an agent who already vacuumed at least 3 rooms for 1000 runs of the simulation at each failure rate. With no policy, it can be seen that the system will in fact assign an agent to vacuum more than 3 rooms quite often. With the guidance policy, however, the extra vacuum assignments (> 3) stay minimal. The violations of the guidance policy increase as the system must adapt to an increasing failure of capabilities until it reaches a peak. At the peak, increased violations do not aid

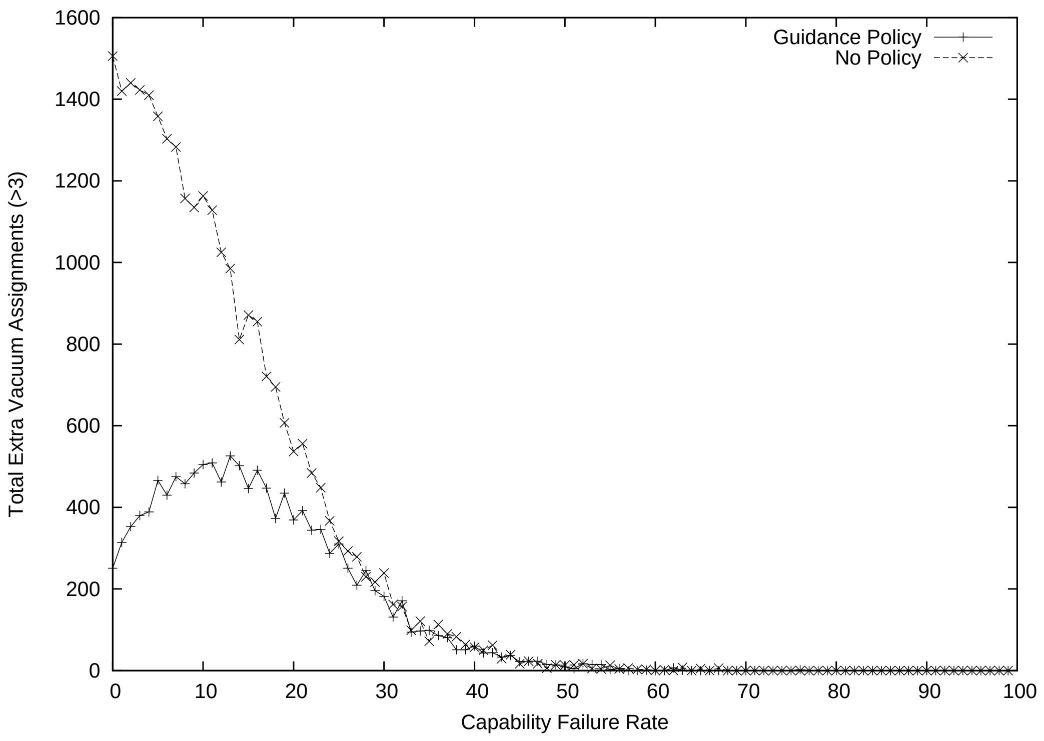


Figure 12. The extra vacuum assignments given capability failure.

in system achievement any longer and eventually the system cannot succeed even without the policy. Thus, the system designer may now wish to purchase equipment with a lower rate of failure, or add more redundancy to the system to compensate. The system designer may also evaluate the peak of the graph and determine whether the cost of the maximum number of violations exceeds the maximum cost he is willing to incur, and if not, make appropriate adjustments.

5 Conclusions and Future Work

Policies are a very useful tool in the development of multiagent systems. As the use of Autonomic computing grows, so grows the need to be able to specify properties of such a system. Guidance policies allow a system designer to guide the system without ‘hard-coding’ a behavior. The system is still given a chance to adapt to new situations. Joaquin Peña et al. described a situation in [15] in which a policy caused a spacecraft to crash into an asteroid. Guidance policies along with precedence relationships between the policies, as presented here, could be one way to resolve such an issue.

Policies may be applied to OMACS by constraining assignments of agents to roles, the structure of the goal model for the organization, or how the agent may play a particular role. Traditional policies may be viewed as *law policies*, since they must never be violated. Law policies are still useful when the system designer never wants a policy to be violated–regardless of system success. Such policies might concern security or human safety.

With the introduction of guidance policies, policies can become an even better mechanism for describing desired properties and behaviors of a system. Guidance policies allow for more flexibility than law policies in that they may be violated under certain circumstances. In this paper, we demonstrated a technique to resolve conflicts when faced with the choice of which guidance policies to violate. It is our belief that guidance policies more truly capture how policies work in human systems. Guidance policies, since they may be violated, can have a partial ordering. That is, one policy may be considered more important than another. In this manner, we may better inform the system allowing it to make better choices when faced with the decision of which policies to violate. We now, through the use of OMACS, along with the metrics described in [16], and with the policy formalisms presented here, are able to provide an environment in which a system designer may evaluate a candidate design, as well as evaluate the impact of changes to that design without deploying or even completely developing the system.

Policies can dramatically improve run-time of reorganization algorithms in OMACS as shown in [21]. Guiding policies can be a way to achieve this run-time improvement without sacrificing system adaptability. System adaptability is an indicator of system flexibility as shown in [16]. The greater the flexibility, the better the chance that the system will be able to achieve its goals.

Policies are an important part of a multiagent system. Future work is planned to ease the expression and analysis of policies. Some work has already been done in this area [18], but it has not been integrated with a multiagent system engineering framework.

Guidance policies add an important tool to multiagent policy specification. However, with this tool comes complexity. Care must be taken to insure that the partial ordering given causes the system to exhibit the behavior intended. Tools, which can visually depict the impact of orderings would be helpful to the engineer considering various orderings. Other work, can be to see if we can infer other policies from a given set of policies. For example, if a system designer wanted to get their system to a state in which they had a defined policy, what sort of guidance policies could be automatically generated?

References

[1] J. Bradshaw, A. Uszok, R. Jeffers, N. Suri, P. Hayes, M. Burstein, A. Acquisti, B. Benyo, M. Breedy, M. Carvalho, D. Diller, M. Johnson, S. Kulkarni, J. Lott, M. Sierhuis, and R. V. Hoof. Representation and reasoning for DAML-based policy and domain services in KAoS and Nomads. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 835–842, New York, NY, USA, 2003. ACM Press.

[2] J. R. Büchi. On a decision method in restricted second-order arithmetics. In *Proceedings of International Congress of Logic Methodology and Philosophy of Science*, pages 1–12, Palo Alto, CA, USA, 1960. Stanford University Press.

[3] S. Chaki, E. M. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. State/event-based software model checking. In E. A. Boiten, J. Derrick, and G. Smith, editors, *Proceedings of the 4th International Conference on Integrated Formal Methods (IFM '04)*, volume 2999 of *Lecture Notes in Computer Science*, pages 128–147. Springer-Verlag, April 2004.

[4] E. M. Clarke Jr., O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[5] S. A. DeLoach. Modeling organizational rules in the multi-agent systems engineering methodology. In *Advances in Artificial Intelligence: 15th Conference of the Canadian Society for Computational Studies of Intelligence (AI 2002)*, volume 2338 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Berlin/Heidelberg, May 2002.

[6] S. A. DeLoach. Engineering organization-based multiagent systems. In *Software Engineering for Multi-Agent Systems IV*, volume 3914 of *Lecture Notes in Computer Science*, pages 109–125. Springer-Berlin/Heidelberg, May 2006.

[7] S. A. DeLoach and W. H. Oyenan. An organizational model and dynamic goal model for autonomous, adaptive systems. Multiagent & Cooperative Robotics Laboratory Technical Report MACR-TR-2006-01, Kansas State University, March 2006.

[8] J. DiLeo, T. Jacobs, and S. DeLoach. Integrating ontologies into multiagent systems engineering. In *Fourth International Conference on Agent-Oriented Information Systems (AIOS-2002)*. CEUR-WS.org, July 2002.

[9] L. Kagal, T. Finin, and A. Joshi. A policy based approach to security for the semantic web. In *The SemanticWeb - ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*, pages 402–418. Springer-Berlin/Heidelberg, 2003.

[10] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[11] J. Ligatti, L. Bauer, and D. Walker. Edit automata: Enforcement mechanisms for run-time security policies. In *International Journal of Information Security*, volume 4, pages 2–16. Springer-Verlag, 2004.

[12] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1991.

[13] K. Olender and L. Osterweil. Cecil: A sequencing constraint language for automatic static analysis generation. *IEEE Transactions on Software Engineering*, 16(3):268–280, 1990.

[14] P. Paruchuri, M. Tambe, F. Ordóñez, and S. Kraus. Security in multiagent systems by policy randomization. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 273–280, New York, NY, USA, 2006. ACM Press.

[15] J. Peña, M. G. Hinchey, and R. Sterritt. Towards modeling, specifying and deploying policies in autonomous and autonomic systems using an AOSE methodology. *EASE*, 0:37–46, 2006.

[16] Robby, S. A. DeLoach, and V. A. Kolesnikov. Using design metrics for predicting system flexibility. In *Fundamental Approaches to Software Engineering (FASE 2006)*, volume 3922 of *Lecture Notes in Computer Science*, pages 184–198. Springer-Berlin/Heidelberg, March 2006.

[17] Y. Shoham and M. Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73(1-2):231–252, 1995.

[18] R. L. Smith, G. S. Avrunin, L. A. Clarke, and L. J. Osterweil. Propel: an approach supporting property elucidation. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 11–21, New York, NY, USA, 2002. ACM Press.

[19] S. D. Stoller, L. Unnikrishnan, and Y. A. Liu. Efficient detection of global properties in distributed systems using partial-order methods. In *Computer Aided Verification*, pages 264–279, 2000.

[20] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Organisational rules as an abstraction for the analysis and design of multi-agent systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):303–328, 2001.

[21] C. Zhong and S. A. DeLoach. An investigation of reorganization algorithms. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI'2006)*, pages 514–517. CSREA Press, June 2006.