A Labeled Graph Approach to Analyze Organizational Performance

Mark Hoogendoorn[1], Jan Treur[1], and Pınar Yolum[2]
[1]*Vrije Universiteit Amsterdam, Department of Artificial Intelligence*
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{mhoogen, treur}@cs.vu.nl
[2]*Bogazici University, Department of Computer Engineering,*
TR-34342 Bebek, Istanbul, Turkey
pinar.yolum@boun.edu.tr

Abstract

Determining the performance of an organization is a must for both human and multi-agent organizations. The performance analysis enables organizations to uncover unexpected properties of organizations and allow them to reconsider their internal workings. To perform such an analysis, this paper represents organizations as labeled graphs that capture, not only the interactions of the entities, but also the characteristics of those interactions, such as their content, frequency, and so on through labels in the graph. Algebraic representation and manipulations of the labels enable analysis of a given organization. Hence, well-known phenomena, such as overloading of participants or asymmetric distribution of workload among participants can easily be detected. Finally, a case study is performed within the domain of incident management.

1. Introduction

Multi-agent organizations consist of agents that interact to carry out their tasks. Current models of multi-agent organizations usually represent organizations as consisting of roles that agents adopt. An organization model then specifies the structure and behavior of the organization in terms of the relations between the roles. An analysis of such an organization model could check if the model satisfies desired properties such as the possibility of completing a desired task given that all agents comply with the requirements of the organization. Whereas such an analysis is useful, it is not sufficient to analyze an executing organization. The main reason is that many design-time choices become concrete during execution. Agents choose who they want to interact with as well as how often they want to do so during run-time. For example, among two agents that enact a merchant role, one might be preferred over the other because the agent has better capabilities, more work capacity, and so on. These subtle interactions of agents at run-time can give rise to interesting situations that can only be detected during execution. That is, as a result of previous decision, one merchant agent will be more loaded than the second merchant will be. Further, the agents that participate in an organization might be designed and developed by independent parties, which requires them to interoperate and execute intelligently at run-time. In other words, such facts about the workings of a multi-agent organization cannot be discovered from a static representation of an organization during design time, but can only be analyzed during the execution time.

Whereas there is a vast literature in the design of multi-agent organizations, there is little work on the analysis of executing multi-agent systems [9, 10]. For this reason, this paper provides a complementary treatment of multi-agent organizations, where in addition to existing design time dynamics of the organizations, a graph representation is used to analyze executing organizations. Executing multi-agent organizations are analyzed by logging the performance of the organization in traces. Graph representations are useful for analyzing organizations; for example for understanding the structure of an organization through theoretical concepts.

This paper presents a formal specification language based on a graph representation. The directed graph captures the relationships between participants in the organization and the labels give semantics to the relationships. Once the labeled graphs are constructed, they can be used to analyze the functioning of the organization at runtime, i.e. analyze traces of the execution of the multi-agent system. Organization designers or analyzers can study the graph to understand the shortcomings of the organizations and to restructure the organization as they see fit. This paper further shows that rules related to the organizations can be developed and automatically checked against the labeled graphs. As a concrete example, detection of overloaded agents is used.

The rest of this paper is organized as follows. Section 2 gives a representation of organizations as labeled graphs. Sections 3 discusses the usage of the graph for external analysis. Section 4 presents a case-study and Section 5 discusses the relevant literature.

2. Organizations as Labeled Graphs

A directed graph $G = (V, E)$ constitutes the basis of the description of an organization in this paper. V denotes the set of nodes, which represent agents that enact a role. E denotes the edges in the graph, which represent the interactions between agents. Graph-based representations are typically used to model processes in areas such as (distributed) workflow management, business process design, organization modeling and organizational performance measurement. Usually the graphs have no labels or simple labels; such as a number that denotes the strength of a link. However, in real organizations edges denote different types of relationships with different properties. To represent such relationships, this paper provides a more complex structure of the labels and formalizes the structure with an algebra.

The example organizations considered here contain agents that fulfill tasks, assign subtasks to other agents, and thus run a business together. There are two primitive concepts we consider: workloads and capacities. An edge e connecting u and v means in this particular application that *u* requires some work to be done by *v*; i.e., edge e denotes a request for *workload*. As in real life, *u* could request different tasks to be performed by *v*. A label on an edge specifies the task type and the strength of the task (i.e., how intensive the work is). The label also includes a list consisting of tasks the current task at hand originates from.

Example 1. Consider the organization in Figure 1. The figure gives a simplified representation of the disaster prevention organization in case of a plane crash in the Netherlands in the form of a labeled graph. Four agents enact the roles as shown in Figure 1: First of all, the airport role is present. This role takes care of the communication with airplanes and is the one that receives the mayday calls. After it has received a mayday call from a plane above the sea, it will contact the coastguard immediately to start a rescue task. The call causes the coastguard a lot of work, as they are in charge of the entire fleet of rescue ships. For possible precautions or backup from the land, the coastguard can contact the alarm center role which will arrange this type of help. The press is also represented as a role as they often request information regarding the number of casualties, information about the cause of the crash, and so on. The coastguard is responsible for fulfilling this task, which is called Inform.

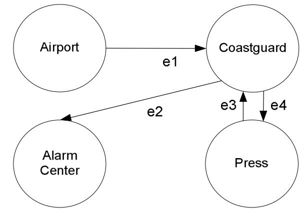

Figure 1. An example organization graph

Each agent in the organization has a certain capacity for each of the tasks that it can perform. Hence, the nodes of the graph are also labeled to denote the capacities of agents. First, a description of a formal language for the labels is given. Next, the capacities of the nodes will be discussed. Finally, the workload is defined.

2.1. Formal Specification Language

The formal language presented in this Section is based on many-sorted algebra. The sorts of the label specification language are shown and explained in Table 1. Based on these sorts, functions are defined to combine these sorts into labels. Statements of this language are equations as the examples accompanying the function definitions show. Throughout the text, when sorts and functions of the algebra are meant, they are denoted in `Courier` font.

First of all, a function is defined to construct a list containing pairs of subtasks. In general, the relation between tasks could be more general than the subtask relationship; for example, by incorporating information on the alternative tasks as well. However, the focus here is on dividing a task into smaller pieces that will be performed by agents. Hence, only concentrating on the subtask relationship.

```
taskSubtaskPair: Task x TaskSubtaskList →
TaskSubtaskList
```

Table 1. Sorts used in the label algebra

Sort	Description
`Value`	Sort for real values.
`Timepoint`	Sort for moments.
`TimeInterval`	Sort for names of intervals that contain two time-points of sort Timepoint.
`Node`	Sort to identify a node.
`Edge`	Sort to identify an edge.
`Task`	Sort to identify tasks.
`Load`	Sort to identify loads.
`LoadValue`	Sort for a Load Value pair.
`LoadValueList`	Sort for a list of LoadValue pairs.
`Label`	Sort to identify a label.
`LabeledLoad`	Sort for a pair containing a LoadValueList and a Label.
`TaskSubtaskList`	Sort for a list of tasks with a subtask relationship between them.
`TaskList`	Sort for a list of tasks.
`Capacity`	Sort to identify a capacity.
`CapacityValue`	Sort for a pair containing the Capacity and a Value.
`OverallCapacity`	Sort to identify the overall capacity.
`OverallCapacity Value`	Sort for a pair containing the OverallCapacity and a Value.
`EdgeActivation`	Sort for specifying the Value of the amount of activations of an Edge during a certain TimeInterval

Considering Example 1 one could express that the `Rescue` task has as a subtask `LandOp` which includes the operations that take place on land. Formally this can be expressed as follows:

```
tS=taskSubtaskPair(Rescue,
      taskSubtaskPair(LandOp, null))
```

Besides that, another function is specified which expresses a regular list of tasks without the subtask relationship between them.

```
taskList: Task x TaskList → TaskList
```

For example, a list containing the tasks that can be performed by the coastguard:

```
tL = taskList(Rescue, taskList(Inform, null))
```

For expressing the load three sorts are used: (i) the list which specifies the task from which this task originates, (ii) the node that carries the load, and (iii) the time interval for which this all holds. Intuitively, a load captures the intensity of the task a node has to do in a given time interval.

```
loadFor: TaskSubtaskList x Node x
TimeInterval → Load
```

In the running example, the load for the coastguard can be expressed for `TimeInterval I` (for example 8 hours) and the `Rescue` task:

```
L = loadFor(tS, Coastguard, I)
```

A load is accompanied by a value expressing the amount of work caused by the load.

```
loadValuePair: Load x Value → LoadValue
```

For the `Load` defined above the value is set to 5:

```
LV = loadValuePair(L, 5)
```

Constructing a list from these `LoadValue` pairs can be done by means of a function. A communication from a role to another role can cause different kinds of load, therefore there is a need to express more than one load for each edge.

```
loadValuePairList: LoadValue x
LoadValueList → LoadValueList
```

In the case of the example, only one `LoadValue` is present:

```
LVL = loadValuePairList(LV, null)
```

Now that the load caused by a connection in a graph can be fully specified it is combined with a label identifier.

```
loadLabel: LoadValueList x Label →
LabeledLoad
```

The label specified above is now called `L1`:

```
LL = loadLabel(LVL, L1)
```

Now a label identifier is associated with an edge.

```
labeledEdge: Edge x Label → LabeledEdge
```

```
LE = labeledEdge(e1, L1)
```

Finally, at runtime an edge will be activated a certain number of times over a certain period, which can also be expressed in the algebra:

```
edgeActivation: Edge x TimeInterval  x
Value → EdgeActivation
```

For example, the edge `E1` was activated 2 times during `TimeInterval I`:

```
EA = edgeActivation(e1, I, 2)
```

Capacities can also be expressed by means of the functions. Capacities belong to nodes, as they are the ones that need to carry the load. The next Section will go into more detail on expressing the capacities. The capacity of a node is the amount of task it can do in a certain time period. The amount of task is denoted by a `TaskList` and the time period is denoted by a `TimeInterval`.

```
capacityOf: TaskList x Node x TimeInterval
→ Capacity
```

A value can be added to the capacity, for example, during the time-interval for which the capacity is specified, one man-hour is available for rescuing.

```
capacityValue: Capacity x Value →
CapacityValue
```

Besides a capacity for specific tasks, a node also has an overall capacity. This overall capacity exists independent of types of tasks it can do.

```
overallCapacity: Node x TimeInterval →
OverallCapacity
```

A value can again be added to this kind of capacity. It can for example say that during the time-interval of a day a maximum of 8 man-hours are available for a specific node.

```
overallCapacityValue: OverallCapacity x
Value → OverallCapacityValue
```

Using the basic ontology of this algebra, its relations can be expressed, and logical relationships can be defined: The primitive terms used in the label algebra are defined by a many-sorted signature. The signature takes into account symbols for sorts, constants, functions and relations, including the equality relation. Among the relations, the equality relation has a special position: the identities (equations) between algebraic term expressions. Further relations can be defined by a relation symbol instantiated with term expressions. Logical relationships involve conditional statements involving relations, both the equality relation and other relations. For simplicity these logical relationships are assumed to be in a clausal format. Examples of constants are names of values, examples of function symbols are +, x, examples of relation symbols are = and <. Examples of logical relationships are

if $t1 < t2$ then $f(t1) < f(t2)$
if $t1 < t2$ then $f(t1 + t2) = f(t2)$

If no other relations than the equality relation occur, the algebra is called functional.

2.2. Capacities

The capacity of a node should be represented flexibly so that realistic situations can be modeled. The following scenarios are seen frequently. For these scenarios, it is assumed that the unit of capacity is man-hours. The maximum man-hours available is fixed: in this case to eight man-hours.

1. **Fixed Capacities:** An agent has a fixed number of hours it can spend on each task as dictated by its role. The sum of these hours should not be more than the maximum amount available.
2. **Constant Task-Specific Capacities**: This time an agent is told how many hours it can spend on each individual task. For example, if the role enacted by this agent has two tasks, coordinating the rescue operations and informing the press, then a possible restriction could state that the agent playing the role can spend at most 5 hours on the rescue operations and 5 hours on informing the press. Of course, working on the rescue operations task for 5 hours still leaves 3 hours for the informing the press task. That is, the maximum number of hours is still constant.
3. **Group-Restricted Capacities:** This time the restriction is not on individual tasks but on sets of tasks. For example, a role can spend a maximum 5 hours on the rescue operations and informing the press and maximum of 4 hours on writing reports. The choice of distributing the 5 hours between the rescue operations task and the informing the press task is up to the agent that plays the role. However, the time spent on the rescue operations and informing the press together cannot exceed 5 hours.
4. **Flexible Capacities**: An agent can decide to work any number of hours on any of its tasks, as long as a certain maximum is not exceeded during the time-interval for which this capacity holds.

It is actually easy to see that both Scenarios 1 and 4 can be modeled in terms of Scenario 2. To model the first scenario, the only thing that needs to be ensured is that the total of the fixed capacities adds up to the maximum. This already defines the scenario in terms of constant task-specific capacities. For the fourth scenario, the individual restriction for each individual work has to be set to the maximum 8 hours. Additionally, Scenario 2 can be modeled a special case of Scenario 3 where each set consists of one task. Hence, accommodating Scenario 3 enables accommodating the remaining scenarios. For the sake of simplicity, disjoint sets of tasks are assumed for a specification of the capacity.

Example 2. To give an example, consider the node `Coastguard`, having capacity for tasks `Rescue` and `Inform`. The capacity of the `Coastguard` concerning the `Rescue` task in the `TimeInterval I` is 8. For the `Inform` task this maximum is set to 2. Combined however, the overall capacity is set to 8, meaning that for the `Inform` and `Rescue` tasks together the time spent can not exceed 8. According to the formal notation as introduced in Section 2.1, the example can be formalized as shown below.

```
c1 = capacityOf(taskList(rescue, null),
                         Coastguard, I)
cval1 = capacityValue(c1, 8)
c2 = capacityOf(taskList(inform, null),
                         Coastguard, I)
cval2 = capacityValue(c2, 2)
co = overallCapacity(Coastguard, I)
coval = overallCapacityValue(cO, 8)
```

2.3. Workloads

A workload of a node is the amount of work it is required to do. Much work has been done to define the concept of workload more precisely, however there is still little consensus on a single definition. In [4] the 'human workload' is described as follows: "The intrinsic difficulty of the activities that an operator must perform establishes the target or nominal level of workload. The difficulty of a particular task may be influenced by any one or several of the following factors: (1) the goals and performance criteria set for a particular task; (2) the structure of the task; (3) the quality, format, and modality in which information is presented; (4) the cognitive processing required; (5) the characteristics of the response devices."

In operations management [8] research has been performed to define the time required to do a job in order to generate a unit of output, which is called work measurement. The initiator of this type of measurement was F.W. Taylor with his scientific management approach. It has however fallen into disfavor because if focuses on routine, repetitive tasks, but recently the labor-intensive service companies have resulted in a new popularity.

The workload of an agent in this paper is determined based on the tasks assigned to it now, how often these assignments take place, and how much of these tasks are delegated to other agents. In general, the agent would perform a percentage of the tasks on its own and assign the remaining tasks to other agents; i.e., create workloads for others. In principle, the newly created workload should be less than that of the initial workload of the agent. The workload of an agent is only determined during execution. Hence, it is not possible to know the workloads exactly during design time and distribute work accordingly.

3. Specification for labels with respect to loads

As has been mentioned before, labeled organization graphs can be used to analyze an organization. It can first be used to model the capacities and the workloads, and thereafter can be applied to analyze a trace representing the state of affairs within a multi-agent system during a certain period.

3.1. Calculations for values of loads

The workload of a node v during an interval I for a task t can be calculated in the following way: Let *workload(e,t)* be the workload for task t caused upon one activation of edge e. This number can be derived from the labeled algebra. First, look up the `taskSubtaskList` associated with this `Task t`: `taskSubtaskList(t, TSL)`. Thereafter get the label for edge e: `labeledEdge(e,L1)`. Now, look up the identifier of the `LoadValueList` via the `Label`: `loadLabel(LVL, L1)` and scan all entries of the `LoadValueList` for a `Load` in which the `TaskSubtaskList` starts with an element in `TSL` or starts with `t`, and holds for `TimeInterval I`. Finally, sum up the `Value` for each of these `Load` elements. Furthermore, for each of these edges, get the amount of activations, during `TimeInterval I`, then the workload can be calculated as shown in Definition 1:

Definition 1. Workload(v, t, I) =
$\sum_{e \in \text{incomingEdges}(v)}$ a1 x workload(e, t)
where edgeActivation(e, I, a1)
- $\sum_{e \in \text{outgoingEdges}(v)}$ a2 x workload(e, t)
where edgeActivation(e, I, a2)

Which entails summing up the workload caused by all incoming nodes, and subtracting from that the workloads distributed through the outgoing edges. The calculation of the overall workload of a node (for all tasks t) is simply summing up all separate workloads, as shown in Definition 2.

Definition 2. workload(v, I) = $\sum_{t \in \text{tasks}}$ workload(v, t, I)

Example 3. Consider the organization as presented in Example 1 and 2. Imagine the following scenario (during an interval I): A Dakota airplane has crashed in the sea, the airport forwards this crash message to the coastguard (causing a load of 5), who in turn delegates the land operations to the alarm center (causing them a load of 1). Besides that, the press starts asking questions about the crash (causing a load of 1 each time), as they have observed the plane crashing in the sea. They request information 40 times, and the Coastguard replies the same number of times (causing the press a load of 0.8 each time). The workload calculation is as follows: workload(coastguard, rescue, I) = (1 * 5) – (1 * 1) = 4 man-hours during interval T for the rescue task workload(coastguard, inform, I) = (40 * 1) – (40 * 0.8)=8 man-hours during interval I for the inform task.

As the calculation for the workload has been explained, the workload of a node can be compared with the capacity of a node, this is referred to as the load of a node. Two different types of loads have been distinguished. First of all, the load for a specific task t can be calculated. To calculate this load, first remember that the capacities are defined for a list of tasks, let l be the list of which t is an element. As it is impossible to calculate loads for individual tasks, loads can only be calculated in terms of these lists of tasks, therefore the calculation of a load for a task t is done by means of the list l the task is part of. Let v be a node, t be a task and I be an interval and let *capacity(v, l, I)* be the capacity of the node v for task list l, during interval I. This can be derived from the labeled algebra as follows: Get the `capacity` for `TaskList L` in which task t is defined for node v during interval I: C =

`capacityOf(L,v,I)`. Thereafter, look up the `Value CV` of this capacity: `capacityValue(C,CV)`. Now the load is defined as shown in Definition 3.

Definition 3. load(v, t, I) =
$(\sum_{task \in l}$ workload(v, task, I))/ capacity(v, l, I) where $t \in l$

This defines that the load for a task is calculated by summing up all workload within the list l (so for every task within l) and dividing it by the capacity defined for that list.

The load can also be calculated for the node as a whole, this is simply done by taking the workload of the node, and dividing it by the overall capacity, *capacity(v, I)*, which can be found using the algebra: `CA = overallCapacity(v,I)` after which the `Value OCV` can be looked up: `overallCapacityValue(CA,OCV)`. The load is now calculated as shown in Definition 4.

Definition 4. load(v, I) = workload(v, I)/ capacity(v, I))

An example of an interesting type of information that can be derived from the load is the load distributions among the nodes in the graph. An organization with evenly balanced nodes is typically preferable over a very uneven distribution of loads.

Example 4. Picture the organization in case of an airplane crash in the North-Sea, the Netherlands again. Following the capacity example as given in Section 2.2 the coastguard has a capacity of 8 man-hours during I for the rescue task, and a capacity of 2 man-hours for the inform task, during that same period. Another capacity that is part of this organization is that of the press. The capacity of the press (which is not shown in a formalization) is defined as being 50 during the time-interval I in which the incident management occurs. The load of the coastguard and the press nodes can be calculated: The general load for the coastguard is: load(coastguard, I) = (12 / 8) = 1.5. More specifically, for the task rescue the load is 0.5 and for the inform task the load is 4.0. For the press, the workload is only caused by the information coming from the coastguard, which can not be distributed elsewhere. Therefore the workload of the press is 40 x 0.8 = 32. As they only have one task, the load of the press, load (press, I), is equal to 0.64. Based on this, it can be seen that the press has a relatively low load compared to the coastguard. By means of this information, a person that is analyzing an organization could suggest that the press should reduce the requests for information to the coastguard and try getting most of their information within the press organization, as they still have sufficient capacity.

3.2. Overloading

As the load for a node has been defined, the definition of a node being overloaded can be given. A certain role is overloaded in case one, or both of the following situations hold: (1) There exists a task t for which the load is greater than 1.0; (2) The load for the entire node, load(v, I) exceeds 1.0. A formal definition is presented below. Please note that due to the choice of representing the capacities by group restricted capacities it can occur that the loads for the individual group are not overloaded whereas the overall load is.

Definition 5: overloaded(v, I) =
$\exists$t:Task (load(v,t,I) > 1.0) $\vee$ (load(v,I) > 1.0)

Example 5. Following from example 4, it can be seen that the role of coastguard is heavily overloaded, for one of the tasks (inform) the load is 4.0, which means 4 times the capacity. The press however is not overloaded as it has a load value of 0.64.

4. Case-Study: Dakota Incident

This Section presents details regarding the implementation of the labeled graph approach into a software tool, and shows an empirical evaluation using a trace obtained from the domain of incident management.

4.1. Implementation

In order to be able to use the algebra and calculations for analyzing multi-agent organizations, a software tool has been created. First, the algebra presented in Section 2 has been implemented in PROLOG [1], including the calculations that are presented in Section 3. For a comparative study of translating an algebraic specification into a PROLOG program, see [2]. A specific interval can be specified over which the calculations of the organizational performance are done. Thereafter, in order to make the calculations of the workloads and loads for the nodes more insightful for e.g. domain experts to evaluate, a visualization tool has been created that graphically shows how much work is being transferred between different nodes within the graph, and represents the load for each of these nodes. Figure 2 shows a screen-shot of the visualization tool. The radius of a node is increased in case the load increases, so the bigger the node the heavier the load on that specific node. Further,

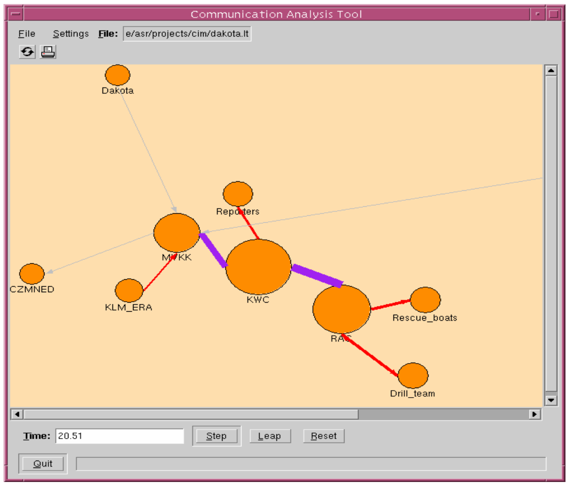

Figure 2. Screenshot of the visualization tool

communication channels that are intensively used (i.e. edges that are activated many times during a particular time interval) are highlighted as well by turning red in case of a lot of activity (or in case of a huge amount of activity purple).

4.2. Empirical Evaluation

In order to evaluate the functioning of the implementation and the approach itself, a case study has been performed in the incident management domain. The case-study itself is based upon reports of a plane crash which occurred in the Netherlands in 1996. A trace of the events that occurred during the rescue of the passengers on board of the plane has been obtained from domain experts and logs that have been made of the communications that took place during the incident management in 1996. The examples used in Sections 2 and 3 include simplifications used for this case study. To enable an analysis, the organization, including the roles and the communications that took place, has been translated to a graph. Thereafter, a domain expert has labeled the graph with the values he thinks are appropriate values for workload caused by activation of a communication line (i.e. an edge). Furthermore, the expert has set capacities for the roles (i.e. nodes) within the incident management organization. According to the experts in the field (written down in incident management reports) the role of the coastguard (abbreviated in the figure to KWC) was heavily overloaded due to too many requests for information of the press, regional alarm center (RAC) and the military airport (MVKK). This indeed showed in the visualization, based on the capacities and workloads set in the graph. The coastguard has a large capacity for handling all the work, but is unable to handle all incoming requests. This shows that the analysis using the labeled graph approach is indeed in line with the manual expert evaluations.

5. Discussion

This paper has presented a formal language for specifying organizations. The specification is based on a graph formalism. The nodes of the graph represent agents and the edges between the nodes are labeled to denote why those edges exist. This allows us to represent the interactions between the agents in an expressive way. It has been shown that using this organization structure properties of executing organizations can be detected, such as the cases where the organization hosts overloaded agents, successfully.

Operations research is a closely related field to the research presented in this paper, see e.g. [5]. Many theories have been developed in that field of research to enable a proper functioning of the organization as a whole, creating a planning for these operations, etc. The research presented in this paper is meant to monitor the performance of these organizations, not to design these operations within the organization.

Another related field is workflow management, in which tools exist that measure and analyze the execution of processes so that continuous improvements can be made. The approach in workflow management can be used as a support tool to analyze the execution, however workflow management systems constitute a huge system which is put into the organization to measure the performance, whereas the approach in this paper simply needs traces of the events and values for the capacities of nodes and workloads regarding tasks. This also enables the presented approach to be used for analyzing occurrences in the past and organizations in which introducing a workflow management system is not feasible.

There is a vast literature on designing multi-agent organizations. Zambonelli *et al.* develop a design methodology, GAIA [11]. GAIA identifies roles, organization rules, environment, and so on as necessary organizational abstractions. Using these constructs, GAIA methodology helps a system designer build its system in a systematic way. Padgham and Winikoff develop Prometheus, an agent-based software development methodology [7]. It consists of a system specification, architectural design, and detailed design phases. While these approaches are useful for designing multi-agent systems, they do not provide any mechanisms for analyzing executing organizations. That is, these methodologies only care for the design phase, but are not targeted for analyzing the multi-agent system during execution, which is the case for the methodology presented in this paper.

Handley and Levis create a model to evaluate the effect of organizational adaptation by means of colored Petri nets [4]. The Petri nets are used to represent external interaction of decision makers as well as internal algorithms the decision maker must perform, and are equipped with labels. In this model the workload of the decision makers is monitored and is used as a performance indicator. The concept of entropy is used to measure the total activity value (which is linked to the workload) of a decision maker. When an overload of a decision maker occurs, the execution time of the internal algorithm has a delay of one additional time point. Decision makers can also base decisions on who to forward an output to on the total activity of the decision maker that can be chosen. Their approach differs from the approach in this paper in the sense that they specify the entire process within the organization, and use the Petri nets to actually simulate an organization. Therefore, their aim is more towards the decision process and the evaluation thereof whereas the approach presented here is more intended as a separate method for evaluating the performance of an organization from an external viewpoint.

Fink *et al.* develop a visualization system to help monitor the performance of businesses [3]. The focus of their work is on presenting a tool that can incorporate different performance metrics from different sources. The aim of the approach presented here is to analyze workings of a business automatically. In this sense, the work of Fink *et al.* is complementary to the work in this paper. Once certain properties are detected by the approach in this paper, they could be feed into a visualization tool to ease the exposure.

The work presented in this paper is open for further improvements. Whereas this paper mainly deals with calculating the effect of an edge on its endpoints, it is also possible to calculate the effects of an edge on nodes that are not immediate endpoints. This can be regarded as calculating the cascading effects of interactions on third parties. Similarly, the representation can be made richer by adding capacities or workflows for groups of agents to model the smaller units in an organization. Ideas developed in this paper can also be used to help agents model others and reason about others' workloads to manage their interactions more efficiently. Such reasoning could possibly even result in change of an organization in case the workload simply cannot be handled, see [6] for more extensive results on this. Furthermore, investigations on how well the approach scales up to large scale multi-agent systems will need to be performed in the future. One important possibility to note here is that of specifying such a system on multiple aggregation levels, whereby the analysis can take place at the highest level (e.g. the workload between departments) while at the lower level focus on parts of the organization (e.g. the workload within a department).

6. References

[1] Colmerauer, A., Kanoui, H., Pasero, R., and Roussel, P., Un Système de Communication Homme-Machine en Français. Groupe de Recherche en Intelligence Artificielle, Université d'Aix-Marseille, Lumini, 1971.

[2] Drosten, K., Translating algebraic specifications to Prolog programs: a comparative study. In: J. Grabowski, P. Lescanne, and W. Wechler, eds., *Algebraic and Logic Programming*, LNCS 343, pp 137–146, Springer, 1988.

[3] Fink G., Krishnamoorthy S., and Kanade A., Naval Crew Workload Monitoring and Visualization. In: First Annual Conf. on Systems Integration, NJ, 2003

[4] Handley, H., and Levis, A., A Model to Evaluate the Effect of Organizational Adaptation. *Computational & Mathematical Organization Theory* 7(1) pp 5–44, 2001.

[5] Hillier, F.S. and Lieberman, G.J., Introduction to Operations Research, McCraw-Hill, SF, 2002.

[6] Hoogendoorn, M., Adaptation of Organizational Models for Multi-Agent Systems based on Max Flow Networks, In: Proceedings of IJCAI 2007, Morgan Kaufmann Publishers, To Appear, 2007.

[7] Padgham, L. and Winikoff, M. Prometheus: A Methodology for Developing Intelligent Agents. In: F. Giunchiglia et al. (eds.) Proc. of the AOSE Workshop, LNCS 2585, pp. 174–185, 2003.

[8] Russell, R.S., and Taylor, B.W., Operations Management, Prentice Hall, New Jersey, 2003.

[9] Shehory O, Sturm A: Evaluation of modeling techniques for agent-based systems. Proceedings of the Intl. Conf. on Autonomous Agents, pa 624–631, ACM Press 2001.

[10] Sudeikat J., Braubach L, Pokahr A., and Lamersdorf W. Evaluation of Agent Oriented Software Methodologies Examination of the Gap between Modeling and Platform. Preproceedings of the AOSE Workshop, 2004.

[11] Zambonelli F., Jennings N.R., Wooldridge M. J., Developing Multiagent Systems: The Gaia Methodology, ACM Transactions on Software Engineering and Methodology, 12(3), September 2003.