Józef Winkowski

FAILURE-RESISTANT RESOURCE MANAGEMENT IN A DISTRIBUTED MULTI-AGENT SYSTEM[1]

Nr 913

Warsaw, September 2000

Abstract

A protocol of resource management in a multi-agent system is presented.

The system consists of a distributed set of agents, each of which runs locally on a computer in the network. The agent running on a computer is responsible for performing jobs initiated on this computer and for managing the local resources. Each job proceeds according to a program and it involves a number of indivisible pieces of work called tasks, each task to be done with the aid of resources of certain types.

The protocol of resource management is presented in the form of rules of a game of tasks for access to resources, in which the role of tasks play the respective agents. It guarantees mutual exclusion of tasks accessing the same resources, starvation freedom of tasks, and as little as possible effect of potential failures.

Keywords: computer, network, agent, multi-agent system, job, task, resource, allocation, competition, game, protocol, mutual exclusion, starvation freedom, failure locality.

Streszczenie

Odporne na zakłócenia zarządzanie zasobami w rozproszonym systemie wieloagentowym

Praca dotyczy protokołu zarządzania zasobami w systemie wieloagentowym.

System składa się z agentów rezydujących w węzłach sieci. Każdy z agentów działa na lokalnym komputerze stanowiącym węzeł sieci. Agent działający w węźle odpowiada za wykonywanie prac inicjowanych w tym węźle i za zarządzanie zasobami lokalnymi. Każda praca przebiega według pewnego programu i wymaga wykonania niepodzielnych akcji zwanych zadaniami, gdzie każde zadanie wymaga zasobów pewnych typów.

Protokół zarządzania zasobami jest podany w postaci reguł gry zadań o dostęp do zasobów, w której rolę zadań pełnią odpowiedni agenci. Protokół ten gwarantuje wzajemne wykluczanie się zadań korzystających z tych samych zasobów, niezagładzanie zadań, oraz możliwie mały wpływ zakłóc eń na działanie systemu.

Słowa kluczowe: komputer, sieć, agent, system wieloagentowy, praca, zadanie, zasób, współzawodnictwo, alokacja, gra, protokół, wzajemne wykluczanie się, niezagładzanie, odporność na zakłócenia.

[1]The work has been supported by CRIT.
The paper is available under the address http://www.ipipan.waw.pl/∼wink/winkowski.htm and as Report 913 of the Institute of Computer Science of the Polish Academy of Sciences.

Pracę zgłosił: Antoni Mazurkiewicz

Adres autora Jozef Winkowski
Instytut Podstaw Informatyki PAN
Ordona 21
01-237 Warszawa
e-mail: wink@ipipan.waw.pl

Symbole klasyfikacji rzeczowej C.2.4, D.4.1
Na prawach rękopisu
Printed as a manuscript

Nakład 140 egzemplarzy. Papier kserograficzny klasy III. Oddano do druku we wrześniu 2000. Wydawnictwo IPI PAN. ISNN 0138-06-48.

1 Introduction

In this paper we present a protocol of resource management for a multi-agent system controlling the execution of jobs in a network of computers.

Suppose that the architecture of the network is as in figure 1, where nodes correspond to computers.

Suppose that the system consists of a distributed set of agents and that it can be specified as follows.

Each agent runs locally on a computer in the network. The agent running on a computer is responsible for executing jobs as in figure 2 that are initiated on this computer, for creating and executing the tasks that must be performed in order to advance the jobs, and for managing the local resources that may be needed in order to execute tasks.

A resource may be either a facility (a processor, a printer, etc.) or a unit of data (a file, a record, etc.). The resources that are present in the network are grouped into pools as in figure 3, where a pool consists of the resources that are present in the same node and can replace each other in the executions of tasks.

The tasks that are created by agents are executed according to individual plans, each plan specifying what resources have to be used, that is indicating the pools to which the needed resources belong. The plan of executing a task may specify resources from different nodes of the network, as it is illustrated in figure 4. The resources that are engaged in execution of a task cannot be used at the same time to execute any other task; in particular, it is impossible to execute simultaneously any set of tasks that are in a conflict in the sense that they require more resources from a pool than it is available in the pool.

The system has no central control. It is controlled only by the agents running in the nodes. The agents are independent and they behave according to a protocol. The protocol, the same for all the agents, specifies what an agent does with the jobs initiated on its local computer, how it creates and executes the respective tasks, how it finds resources for executing these tasks, and how it fulfils its own and of other agents requests concerning the local resources it controls. In particular, it contains a protocol of resource management.

The protocol of resource management that we present in this paper is defined in the form of rules of a game of tasks for access to resources. In this game each agent acts for each task it executes as for a player, and for each pool of its local resources as for a resource manager.

The multi-agent system of our concern is similar in spirit to the system Challenger that has been described in [CMM 97], and to the possible distributed versions of the workflow management systems that have been described in [GH 95] and [Aa 98].

The game of tasks for access to resources can be viewed as a distributed algorithm for dynamic resource allocation corresponding to those described in [CM 84], [AS 90], [WPP 91], [CS 93], and [Rh 98]. It does not require any a priori assumptions about the system. Its main feature as a solution to the resource allocation problem is the best possible *failure locality* in the sense of [CS 93], or *failure extent*, that is the least possible effect of task failures. Namely, only the tasks which are in a direct conflict with a faulty task can be disturbed. This property of our solution is achieved for the price of economy. The latter is usually expressed in terms of *response time*, which measures the time delay between a process wishing to access resources and it actually being able to do so, and *message complexity*, which measures the number of messages sent or received by a process during each access to resources. Usually, these performance measures are defined for the worst cases. As our solution depends on agent reaction times that usually are more or less random, the average response time and message complexity are more adequate. However, any estimation of these performance measures is doubtful because it requires problematic assumptions about appropriate probability distributions. In this situation the only reasonable estimation we are able to give is the response time and message complexity for tasks which are not involved in conflicts with other tasks. We shall see that for such tasks, rather typical in systems not overloaded too much with jobs, both the measures are of order ϱ, where ϱ is the maximal number of resources a task may need.

Under rather weak and realistic assumptions our protocol guarantees that each task that is created in the system is executed sooner or later, and, consequently, no job that is initiated in the network is prevented from progressing.

The idea of the protocol has its origins in the works on distributed algorithms for resource management that have been described in [Wink 81] and [KW 82], and on the application of such algorithms in a distributed operating system that have been described in [FW 86]. The protocol described in the present paper is an improved version of the protocols that have been described in [Wink 98] and [Wink 00]. A version of the protocol has been implemented as described in [Dem 00].

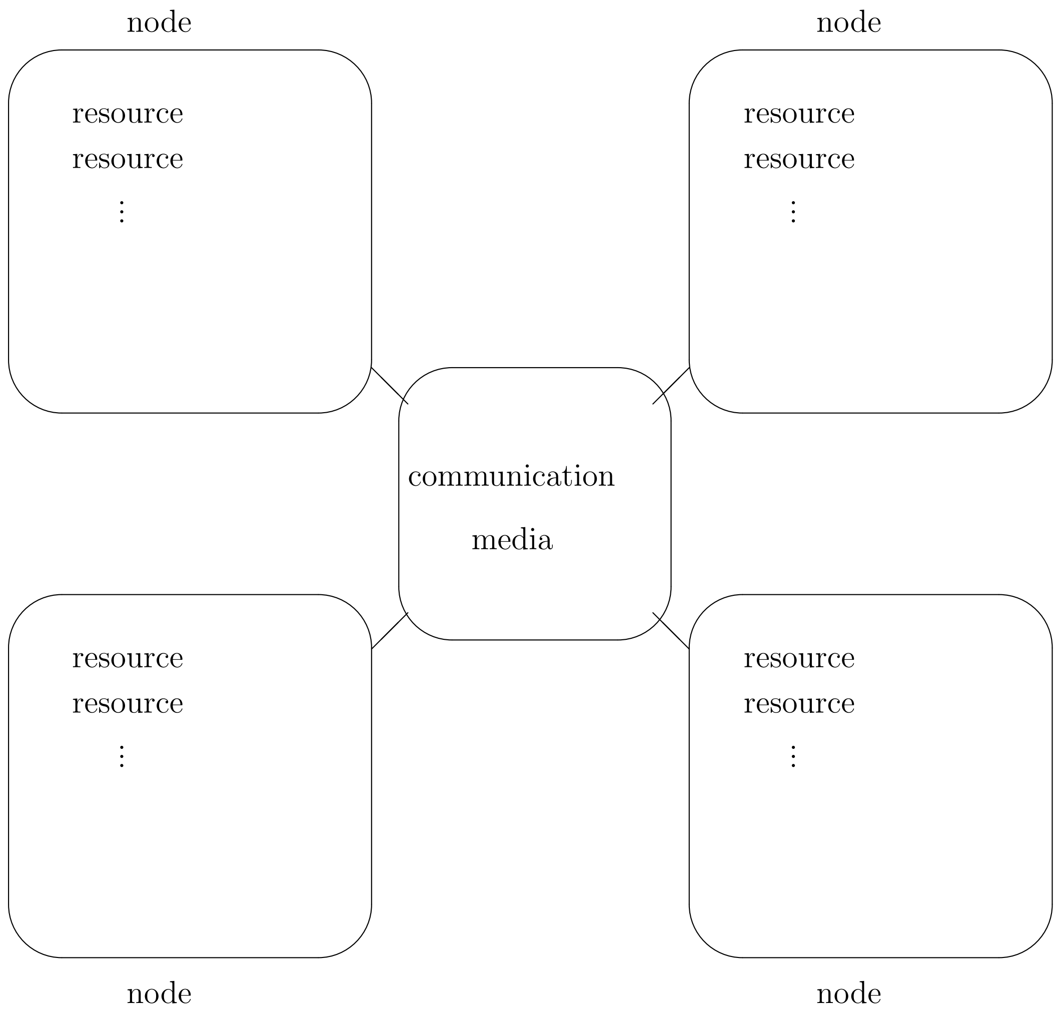

Figure 1: A network of computers

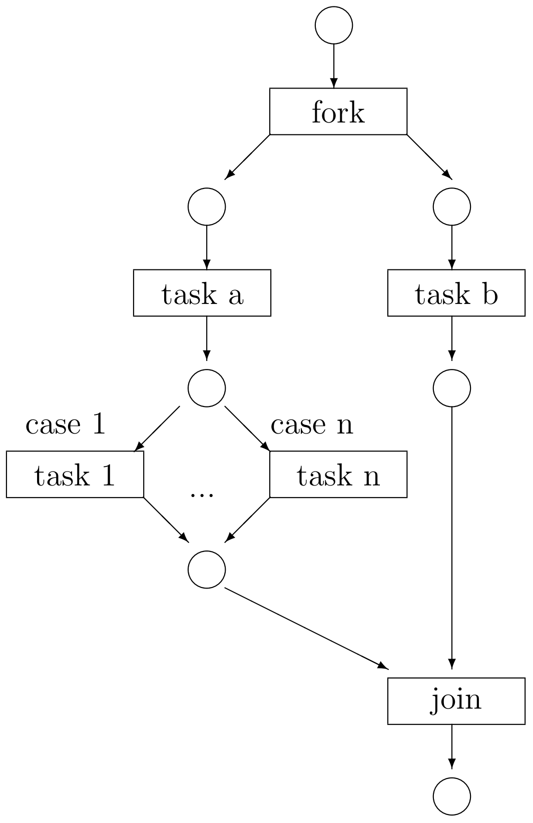

Figure 2: A job

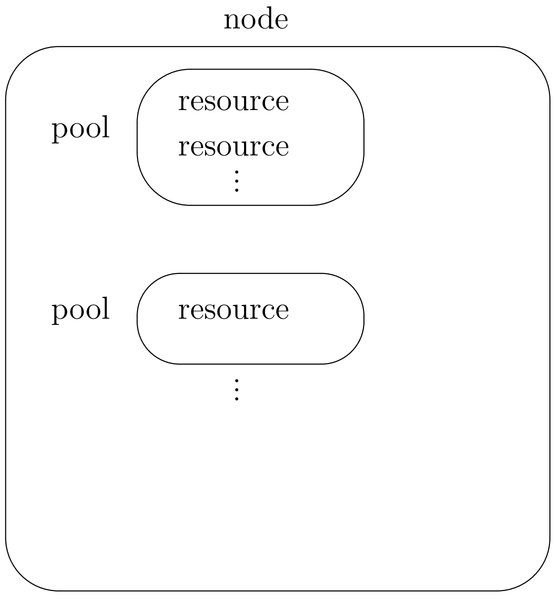

Figure 3: Pools of resources in a node

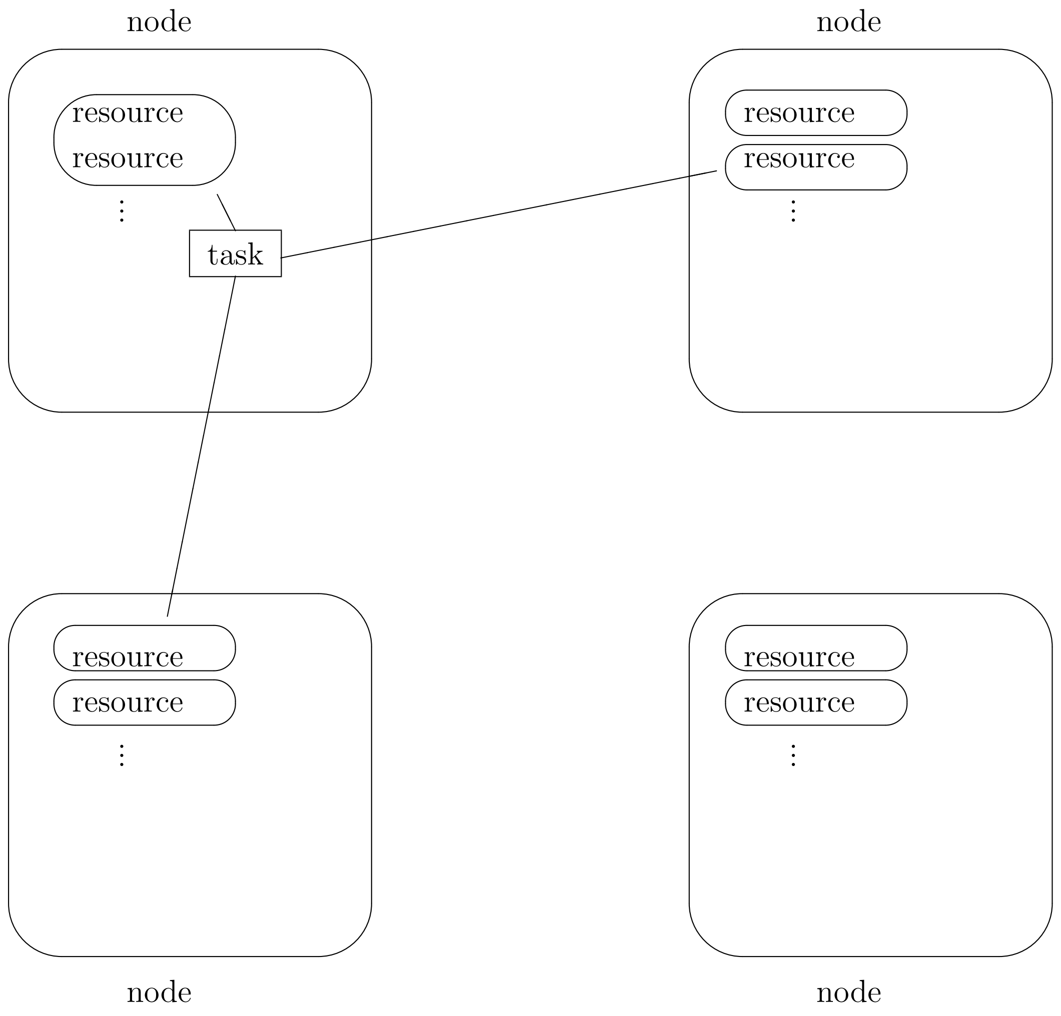

Figure 4: Possible requirements of a task

2 Protocol

As it was said, our protocol of resource management is defined in the form of rules of a game of tasks for access to resources. At each moment the game is played by the tasks that have been created but have not assigned yet resources allowing to realize their plans. Each task plays it with the aid of the respective agent and the resource managers that control the pools of resources of task interest. It does it by sending suitable messages to these resource managers and by tracing the information about the resulting situations at the pools the resource managers broadcast to the interested tasks.

Each task plays the game in two phases: the *entering phase* and the *competition phase*.

The entering phase begins with creation of the respective task and ends with taking from the agent that serves the task a *numbered ticket*.

We assume that each agent has at its disposal a potentially infinite set of numbered tickets such that the sets of tickets of agents are mutually disjoint and no two tickets available in the system are numbered with the same number. We assume also that the agents may give temporarily tickets to tasks and that the tickets define then the priorities of tasks in their competition phases (the lower is the number on the ticket, the higher is the priority).

Note, that these assumptions are satisfied if to each agent is assigned a positive integer that is different from the integers assigned to other agents and less than a certain positive integer K, and if the agent with assigned integer i uses only positive integers of the form $nK + i$ as numbers of its tickets.

A task in the entering phase registers its presence at the pools of its interest by sending messages called *registration forms* to the respective resource managers, and memorizes the tasks that apply for the resources from the pools of its interest and are in the competition phase as its *absolute predecessors*.

The presence of the registration form of a task at a pool obliges the respective resource manager to inform the task about the situation at the pool whenever the situation changes. Consequently, the tasks having their registration forms at a pools of their interest are able to trace with some delays the situations at these pools.

A task that has sent its registration forms to the resource managers controlling all the pools of its interest waits untill all its absolute predecessors access the needed resources and leave the game. When this happens it is allowed to take a numbered ticket from its agent and enter the competition phase.

The competition phase begins with taking a numbered ticket and it ends with accessing and releasing resources allowing to realize the respective plan. At the beginning the task declares it admission to the competition phase by sending to the resource managers controlling the pools of its interest messages called *admission forms*. The next behaviour of the task in the competition phase is cyclic, where in each cycle the task behaves depending on the situation at the pools of resources of its interest: either it is distributing at the pools of resources of its interest information that brings it to winning access to the needed resources (the *progressive behaviour*), or it is withdrawing the already distributed information back and waiting for suitable conditions (the *regressive behaviour*). What information should be distributed depends on the requirements imposed on the protocol.

In a restricted variant of the protocol, that does not take into account possible failures, a task is trying to distribute *requests* of accessing resources. The requests delivered to pools of resources are located by the respective resource managers in queues.

Distribution of requests by a task T is allowed when, according to the knowledge of this task, the following condition is satisfied:

(p1) For each pool P of interest of T, the number of requests that precede at P the actual or potential request of T and come from tasks that have priorities not lower than the priority of T is less than the number of resources currently accessible in P.

The regressive behaviour is obligatory when, according to the knowledge of the task, this condition is not satisfied.

A task wins the required access to resources when it has its requests distributed at all the pools of resources of its interest, and all these requests are preceeded in the respective queues by less requests than the numbers of free resources in the respective pools. Such a task sends a *winning form* to each of the respective resource managers and waits for information from these managers about booking for it concrete resources. The resources booked for the task become accessible to the task and inaccessible to other tasks. When the task learns about such a situation it withdraws its admission forms and requests from the respective resource managers, gives back the numbered ticket it possesses to the respective agent, and is executed. When the execution ends the task sends a *releasing form* to each of the respective resource managers and waits for information from these managers about releasing all the resources booked for the task. When the task learns that the respective resources are released, it leaves the game.

The behaviour of an agent is shown in figure 5. The conditions under which agents may compete with each other if the time from creation to registration does not exceed a certain given value Δ are illustrated in figure 6. A state of the game of tasks for access to resources is illustrated in figure 7.

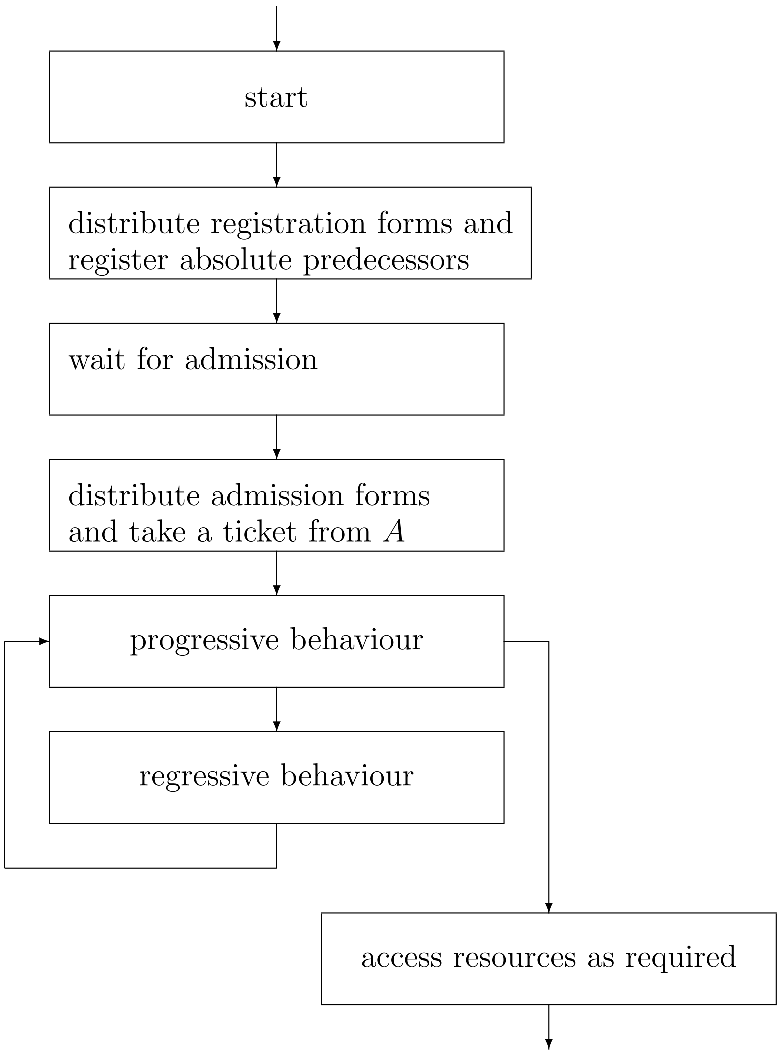
start
distribute registration forms and register absolute predecessors
wait for admission
distribute admission forms and take a ticket from A
progressive behaviour
regressive behaviour
access resources as required

Figure 5: The behaviour of a task T created by an agent A

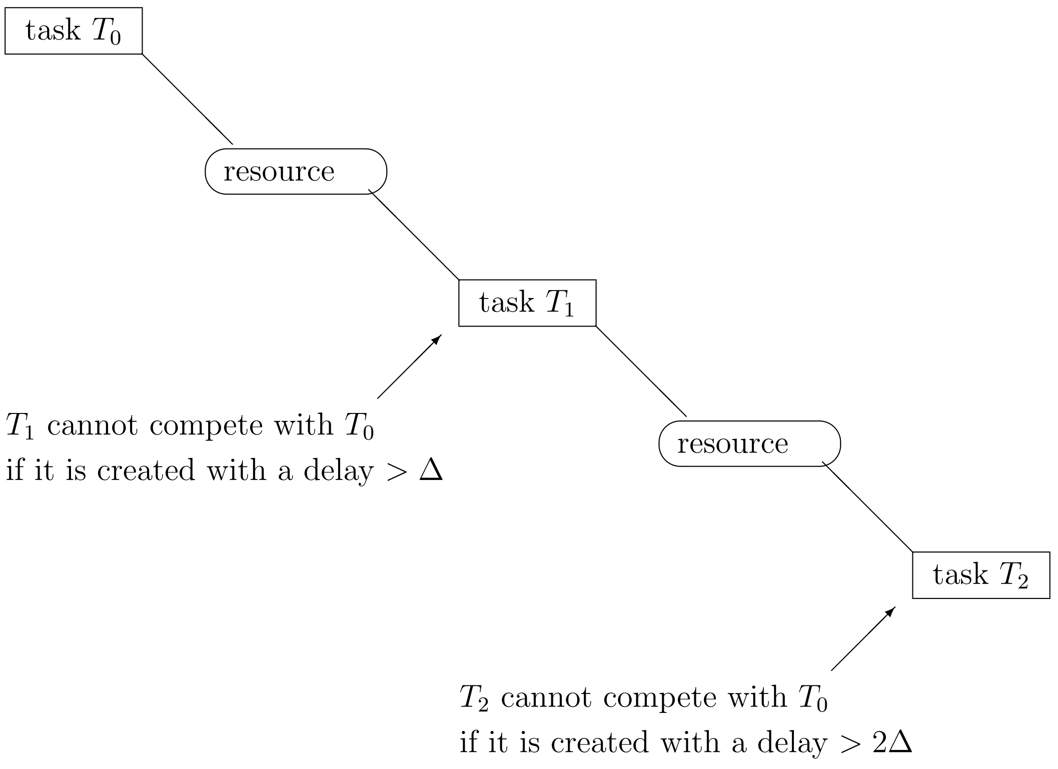
task T_0
resource
task T_1
resource
task T_2
T_1 cannot compete with T_0 if it is created with a delay $> \Delta$
T_2 cannot compete with T_0 if it is created with a delay $> 2\Delta$

Figure 6: T_1 and T_2 cannot compete with T_0 if they are created too late

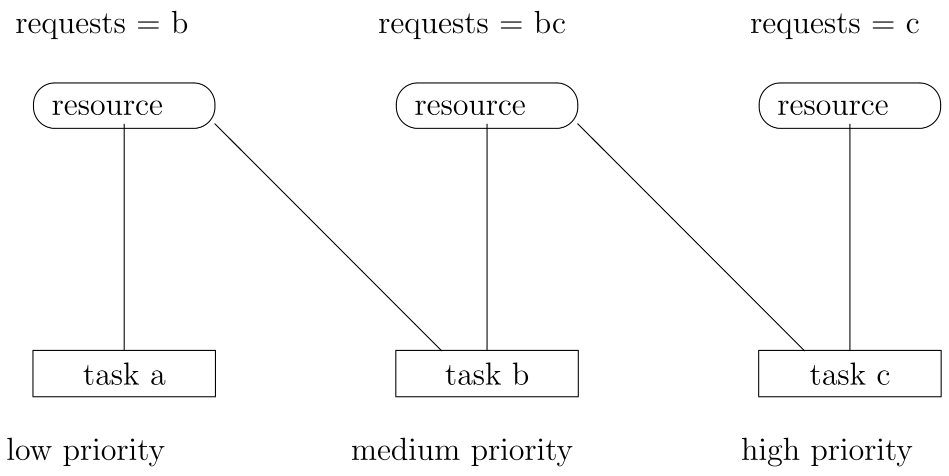

Figure 7: Competing tasks

In a complete variant of the protocol, that takes into account possible failures and is oriented to reducing their effect, the process of distributing requests by a task T may be followed by a process of distributing messages called *order forms*, and the condition (p1) is replaced by the following condition:

(p2) For each pool P of interest of T, the number of requests that precede at P the actual or potential request of T and come from tasks that have priorities not lower than the priority of T or have an order form at P is less than the number of resources currently accessible in P.

The order forms may be distributed if the task has its requests distributed at all the pools of resources of its interest, and all these requests are preceded in the respective queues by less requests than the numbers of accessible resources in the respective pools. The distributed order forms inform the possible competitors of the task that they should consider the task as a potential winner that makes some resources in the respective pools inaccessible. The task becomes an actual winner and proceeds further as in the restricted variant of the protocol if it has its order forms distributed at all the pools of resources of its interest.

If the task has its requests distributed at all the pools of resources of its interest but some of these requests is preceded by at least as many requests as the number of accessible resources in the respective pool then the task sets a special local variable called *alarm-clock* on the moment by Γ units later than the current indication of the local clock of its agent, where Γ is a system parameter, registers the preceding requests of tasks that have no corresponding order forms at the respective pools, called *blocking requests*, and waits for some of these requests to be removed. If none of the blocking requests is removed before the moment thus set, the task is allowed to apply for transposing its requests with the requests of predecessors by sending messages called *advancing forms* to the respective resource managers.

The regressive behaviour is obligatory if (p2) is not satisfied.

The purpose of transposing requests in the queues at pools of resources is to facilitate executing tasks that are blocked by faulty tasks. Due to the requirement of distributing order forms by tasks having chances to win, and due to the resticted time of waiting for tasks whose requests precede the requests of a task, the task can recognize the presence of a failure and overcome its consequences.

3 Implementation

The execution of the protocol is a result of interaction of agents, tasks, and resource managers. We describe the mechanisms of this interaction in terms of local states and actions.

Data

Let *Agents*, *Resources*, and *Pools* denote respectively the set of (names of) agents, the set of (names of) resources, and the set of (names of) pools of resources, all of them finite.

Let *Tickets* denote a set of numbered tickets, exactly one ticket for each natural number.

Let

$manager : Pools \rightarrow Agents$

$pool : Resources \rightarrow Pools$

$owner : Tickets \rightarrow Agents$

denote respectively the function that assigns to each pool of resources the agent that controls this pool, the function that assigns to each resource the pool to which this resource belongs, and the function that assigns to each numbered ticket the agent being the owner of this ticket.

We assume that each agent is equipped with a local clock that is accessible to tasks controlled by the agent. We assume also that the set of tickets owed by each agent is infinite and write $N < N'$ whenever N is a numbered ticket with the number less than the number of a numbered ticket N'.

Let $Tasks$, $RegistrationForms$, $AdmissionForms$, $Requests$, $OrderForms$, $AdvancingForms$, $Withdrawals$, $WinningForms$, and $ReleasingForms$ denote respectively the set of (names of) potential tasks, the set of (names of) potential registration forms of tasks, the set of (names of) potential admission forms of tasks, the set of (names of) potential requests of accessing resources, the set of (names of) potential order forms, the set of (names of) potential advancing forms, the set of (names of) potential withdrawals of requests of accessing resources, the set of (names of) potential winning forms of tasks, and the set of (names of) potential releasing forms, all of them infinite.

Let

$agent : Tasks \to Agents$

$interest : Tasks \to Subsets(Pools)$

$task : RegistrationForms \cup AdmissionForms \cup Requests$
$\quad \cup OrderForms \cup AdvancingForms \cup Withdrawals$
$\quad \cup WinningForms \cup ReleasingForms \to Tasks$

$ticket : AdmissionForms \cup Requests \cup OrderForms$
$\quad \cup AdvancingForms \cup Withdrawals$
$\quad \cup WinningForms \to Tickets$

denote respectively the function that assigns to each task the agent that creates this task, the function that assigns to each task the subset of pools of its interest, the function that assigns to each registration form, admission form, request, order form, advancing form, withdrawal, winning form, and releasing form, the task that creates it, and the function that assigns to each admission form, request, order form, advancing form, withdrawal, and releasing form, the ticket taken by the respective task.

Local states of agents

Each local state of agent A as a controller of tasks and owner of numbered tickets consists of the following data:

- $clock_A$,

  the local clock of A whose values are real numbers,

- $Controlled_A \in FiniteSubsets(Tasks)$,

  the set of tasks that are currently under the control of A,

- $Entering_A \subseteq Controlled_A$,

  the subset of those tasks from $Controlled_A$ that are currently in the entering phase,

- $Competing_A \subseteq Controlled_A$,

  the subset of those tasks from $Controlled_A$ that are currently in the competition phase,

- $DistributedTickets_A \subseteq Tickets$,

  the subset of numbered tickets of A that are currently at disposal of tasks.

We assume that the agent can access and change the local state of each task in $Controlled_A$ and that the sets $Entering_A$ and $Competing_A$ are disjoint.

Local states of resource managers

Each local state of the resource manager that controls a pool P of resources consists of the following data:

- $\mathit{Applications}_P \in \mathit{FiniteSubsets}(\mathit{RegistrationForms})$,

  the set of registration forms that are currently present at P,

- $\mathit{Admissions}_P \in \mathit{FiniteSubsets}(\mathit{AdmissionForms})$,

  the set of admission forms that are currently present at P,

- $\mathit{Requests}_P \in (\mathit{Requests})^*$,

  the current queue of requests of tasks at P,

- $\mathit{Orders}_P \in \mathit{FiniteSubsets}(\mathit{OrderForms})$,

  the current set of order forms of tasks at P,

- $\mathit{InUse}_P \subseteq \mathit{pool}^{-1}(P)$,

  the set of resources from P that are actually in use,

- $\mathit{user}_P : \mathit{InUse}_P \to \mathit{Tasks}$,

  the function that assign to each resource from InUse_P the task that currently uses this resource.

We assume that these data are consistent in the sense that, for each X in $\mathit{Admissions}_P$ there exists Y in $\mathit{Applications}_P$ such that $\mathit{task}(X) = \mathit{task}(Y)$, for each X in $\mathit{Requests}_P$ there exists Y in $\mathit{Admissions}_P$ such that $\mathit{task}(X) = \mathit{task}(Y)$, and for each X in Orders_P there exists Y in $\mathit{Requests}_P$ such that $\mathit{task}(X) = \mathit{task}(Y)$.

Local states of tasks

Each local state of a task T consists of items status_T, $\mathit{TicketsAt}_T$, $\mathit{alarm{-}clock}_T$, $\mathit{Destinations}_T$, and of families
$\mathit{aboutPredecessors}_T = (\mathit{AboutPredecessors}_{T,P} : P \in \mathit{interest}(T))$,
$\mathit{aboutRegistered}_T = (\mathit{AboutRegistered}_{T,P} : P \in \mathit{interest}(T))$,
$\mathit{aboutRequests}_T = (\mathit{AboutRequests}_{T,P} : P \in \mathit{interest}(T))$,
$\mathit{aboutOrders}_T = (\mathit{AboutOrders}_{T,P} : P \in \mathit{interest}(T))$,
$\mathit{aboutUse}_T = (\mathit{AboutUse}_{T,P} : P \in \mathit{interest}(T))$,
$\mathit{blockingRequests}_T = (\mathit{BlockingRequests}_{T,P} : P \in \mathit{interest}(T))$,
where

- status_T

  is an item that may have one of the values $\mathit{REGISTERING}$, $\mathit{ADMITTED}$, TRYING, $\mathit{ADVANCING}$, $\mathit{ORDERING}$, $\mathit{WINNING}$, $\mathit{ACCESSING}$, $\mathit{RELEASING}$,

- $\mathit{TicketsAt}_T \subseteq \mathit{Tickets}$

  is the at most one-element set of numbered tickets currently at disposal of T; we recall that the number of a ticket in this set defines the priority of T,

- $\mathit{alarm} - \mathit{clock}_T$

  is a variable whose value represents a moment set by the task or a special symbol ∞ that represents the fact that no concrete moment has been set,

- $Destinations_T \subseteq interest(T)$

  is the set of pools of resources controlled by the resource managers to which T is supposed to send a message,

- $AboutPredecessors_{T,P} \in FiniteSubsets(AdmissionForms)$

  is the set of admission forms of absolute predecessors that T has found in the game most recently,

- $AboutRegistered_{T,P} \in FiniteSubsets(RegistrationForms)$

  is the set of registration forms at P that T has learned about most recently,

- $AboutRequests_{T,P} \in (Requests)^*$

  is the queue of requests of tasks at P that T has learned about most recently,

- $AboutOrders_{T,P} \in FiniteSubsets(OrderForms)$

  is the set of order forms of tasks at P that T has learned about most recently,

- $AboutUse_{T,P} \in FiniteSubsets(Resources)$

  is the set of resources from P that T considers currently as inaccessible,

- $BlockingRequests_{T,P} \in FiniteSubsets(Requests)$

  is the sequence of blocking requests at P that T has registered most recently.

We assume that these data are consistent in the sense that there is no X in $aboutPredecessors_T$ such that $task(X) = T$, for each X in $aboutRequests_T$ there exists Y in $aboutRegistered_T$ such that $task(X) = task(Y)$, for each X in $aboutOrders_T$ there exists Y in $aboutRequests_T$ such that $task(X) = task(Y)$, each X in $AboutUse_{T,P}$ belongs to $pool^{-1}(P)$, and $BlockingRequests_{T,P}$ is the subsequence of $AboutRequests_{T,P}$ consisting of those predecessors of the request of T that have no order form in $AboutOrders_{T,P}$.

Communication
The interaction of agents, tasks, and resource managers, is based on the asynchronous communication that is presented in figure 8.

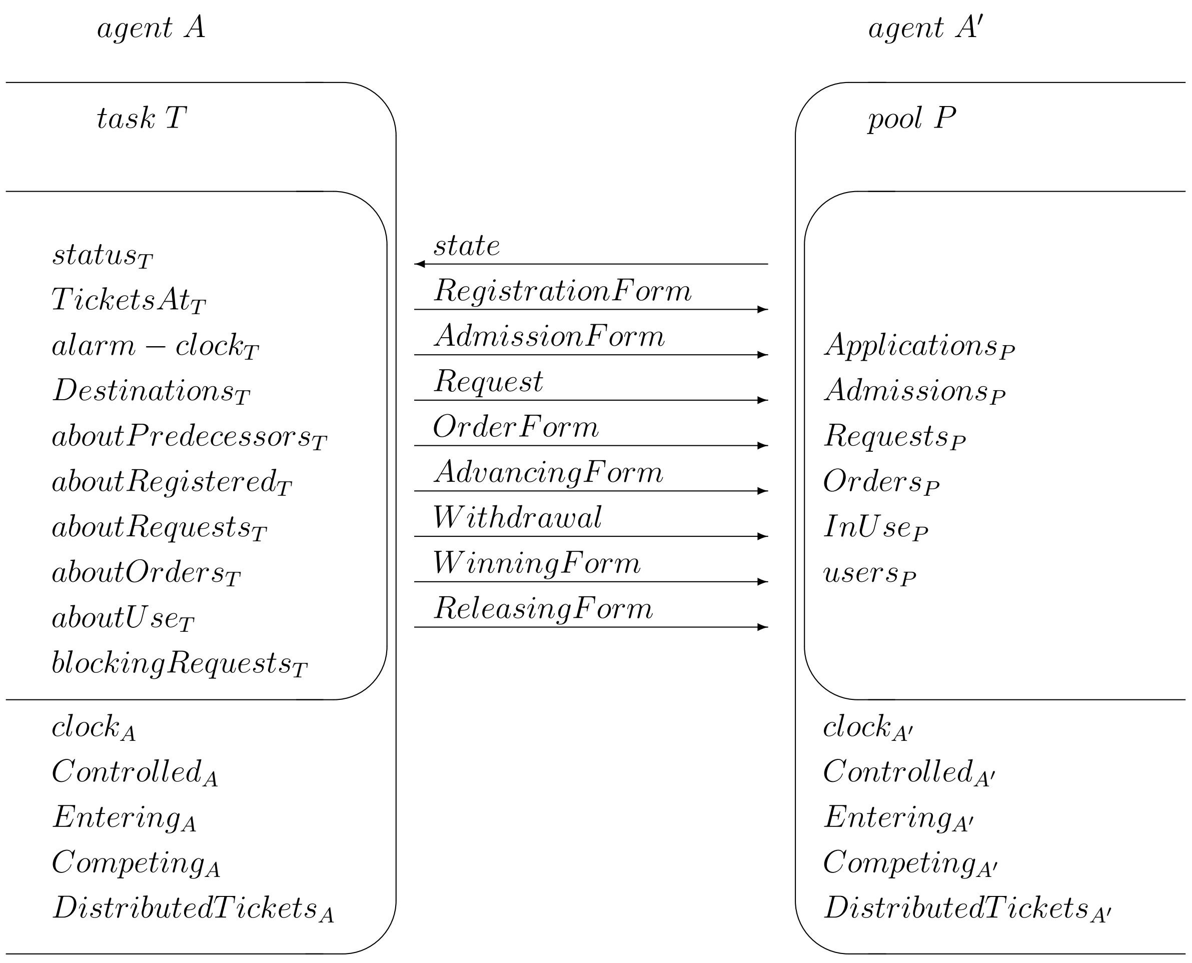

Figure 8: The exchange of data between a task and a resource manager

Actions of agents

Actions of an agent A as a generator of tasks and owner of numbered tickets are as follows.

- Advance of local time.

  This action represents an autonomous increase of the value of the local clock of A.

- Creation of a task T.

  This action refers to such a task T that is generated by A. It may be executed in every local state a of A. It results in the local state a' that differs from a as follows:

  $Controlled_A(a') = Controlled_A(a) \cup \{T\}$

  $Entering_A(a') = Entering_A(a) \cup \{T\}$

  It sets the local state of T to t such that

  $status_T(t) = REGISTERING$

  $TicketsAt_T(t) = \emptyset$

  $alarm - clock_T(t) = \infty$

$AboutPredecessors_{T,P} = \emptyset$ for all $P \in interest(T)$
and $BlockingRequests_{T,P}(t)$ are empty for all $P \in interest(T)$.

- Giving a numbered ticket N to a task T.

  This action refers to such a task T that is generated by A and to such a ticket N that is currently at disposal of A. It is a reaction to reaching by T a state t with $status_T(t) = ADMITTED$.

  It may be executed in every local state a of A. It results in the local state a' that differs from a as follows:

  $Entering_A(a') = Entering_A(a) - \{T\}$

  $Competing_A(a') = Competing_A(a) \cup \{T\}$

  $DistributedTickets_A(a') = DistributedTickets_A(a) \cup \{N\}$

  It changes the local state of T to t' that differs from t as follows:

  $TicketsAt_T(t') = \{N\}$

- Taking a numbered ticket N from a task T.

  This action refers to such a task T that is generated by A and to such a ticket N that is currently at disposal of T. It is a reaction to reaching by T a state t with $status_T(t) = ACCESSING$ and $TicketsAt_T(t) = \{N\}$.

  It may be executed in every local state a of A. It results in the local state a' that differs from a as follows:

  $Controlled_A(a') = Controlled_A(a) - \{T\}$

  $Competing_A(a') = Competing_A(a) - \{T\}$

  $DistributedTickets_A(a') = DistributedTickets_A - \{N\}$

  It changes the local state of T to t' that differs from t as follows:

  $TicketsAt_T(t') = TicketsAt_T(t) - \{N\} = \emptyset$

Notice that actions of A preserve the consistency of the data constituting local states of T.

Actions of resource managers
Actions of the resource manager that controls a pool P of resources are as follows.

- Receiving a registration form F from a task T.

  This action executed in a local state p results in the local state p' that differs from p as follows:

  $Applications_P(p') = Applications_P(p) \cup \{F\}$

  It results also in sending messages containing information on the new local state p' to all the tasks with registration forms in $Applications_P(p')$.

- Receiving an admission form F from a task T.

  This action executed in a local state p results in the local state p' that differs from p as follows:

  $Admissions_P(p') = Admissions_P(p) \cup \{F\}$

  It results also in sending messages containing information on the new local state p' to all the tasks with registration forms in $Applications_P(p')$.

- Receiving a request R from a task T.

  This action executed in a local state p such that R does not occur in $\mathit{Requests}_P(p)$ results in the local state p' that differs from p as follows:

  $\mathit{Requests}_P(p') = \mathit{Requests}_P(p) \frown R$

  where $\mathit{Requests}_P(p) \frown R$ denotes the sequence consisting of $\mathit{Requests}_P(p)$ followed by R. It has no effect on the local state of P if R occurs in $\mathit{Requests}_P(p)$. In each case it results also in sending messages containing information on the new local state p' to all the tasks with registration forms in $\mathit{Applications}_P(p')$.

- Receiving an order form F from a task T.

  This action executed in a local state p results in the local state p' that differs from p as follows:

  $\mathit{Orders}_P(p') = \mathit{Orders}_P(p) \cup \{F\}$

  It results also in sending messages containing information on the new local state p' to all the tasks with registration forms in $\mathit{Applications}_P(p')$.

- Receiving an advancing form F from a task T.

  This action, executed in a local state p such that

  $\mathit{Requests}_P(p) = X \frown R_1 \frown R_2 \frown Y$

  with a request R_2 of T preceded by a request R_1 of a task T' that has no order form in $\mathit{Orders}_P(p)$, results in a local state p' that differs from p as follows:

  $\mathit{Requests}_P(p') = X \frown R_2 \frown R_1 \frown Y$

  It has no effect if in $\mathit{Requests}_P(p)$ there is no request of T, or such a request has no predecessor, or the predecessor is a request of a task that has an order form in $\mathit{Orders}_P(p)$. In each case, it results also in sending messages containing information on the new local state p' to all the tasks with registration forms in $\mathit{Applications}_P(p')$.

- Receiving a withdrawal W from a task T.

  This action executed in a local state p results in the local state p' that differs from p as follows:

  $\mathit{Requests}_P(p') = \mathit{Requests}_P(p) - \{R\}$,

  $\mathit{Orders}_P(p') = \mathit{Orders}_P(p) - \{O\}$,

  where $\mathit{Requests}_P(p) - \{R\}$ denotes the result of removing from $\mathit{Requests}_P(p)$ the request with $\mathit{task}(R) = \mathit{task}(W) = T$, if any, and $\mathit{Orders}_P(p) - \{O\}$ denotes the result of removing from $\mathit{Orders}_P(p)$ the order form with $\mathit{task}(O) = \mathit{task}(W) = T$, if any. It results also in sending messages containing information on the new local state p' to all the tasks with registration forms in $\mathit{Applications}_P(p')$.

- Receiving a winning form F from a task T.

  This action executed in a local state p such that $\mathit{pool}^{-1}(P)$ contains a resource r not in $\mathit{InUse}_P(p)$ results in the local state p' that differs from p as follows:

  $\mathit{InUse}_P(p') = \mathit{InUse}_P(p) \cup \{r\}$

  $\mathit{user}_P(p') = \mathit{user}_P(p) \cup \{(r, T)\}$

It has no effect if a suitable r does not exists. In each case it results also in sending messages containing information on the new local state p' to all the tasks with registration forms in $Applications_P(p')$.

- Receiving a releasing form F from a task T.

  This action executed in a local state p such that $T = (user_P(p)) = r$ for a resource r results in the local state p' that differs from p as follows:

  $Applications_P(p') = Applications_P(p) - \{F_1\}$

  $InUse_P(p') = InUse_P(p) - \{r\}$

  $user_P(p') = user_P(p) - \{(r, T)\}$

  where F_1 is the registration form of T. It has no effect if a suitable r does not exists. In each case it results also in sending messages containing information on the new local state p' to all the tasks with registration forms in $Applications_P(p')$ and to T.

Notice that actions of the resource manager that controls P preserve the consistency of the data constituting local states of P. Notice also that messages containing information about the local state of the resource manager controlling a pool P that is reached due to each action is delivered to all the tasks registered currently at P.

Actions of tasks
Actions of a task T are direct or implicit consequences of receiving messages from the resource managers that control the pools of resources of interest of T and of the flow of the local time of the agent that controls T. They are as follows.

- Receiving a message m from the resource manager that controls a pool P of resources, where m informs that the resource manager has reached a local state p.

  This action represents the act of learning by T that the resource manager that controls P reached some time ago the local state p. When executed in a local state t such that $status_T(t) \neq RELEASING$, or $status_T(t) = RELEASING$ and the registration form of T is present in $AboutRegistered_{T,P}(t)$ and in $Applications_P(p)$, it results in the local state t' that differs from t as follows:

  if $status_T(t) = REGISTERING$ and the registration form of T is absent in

  $AboutRegistered_{T,P}(t)$ and present in $Applications_P(p)$ then

  $AboutPredecessors_{T,P}(t') = Admissions_P(p)$

  $AboutRegistered_{T,P}(t') = Applications_P(p)$

  $AboutRequests_{T,P}(t') = Requests_P(p)$

  $AboutUsers_{T,P}(t') = InUse_P(p)$

  else

  $AboutPredecessors_{T,P}(t') = AboutPredecessors_{T,P}(t) \cap Admissions_P(p)$

  $AboutRegistered_{T,P}(t') = Applications_P(p)$

  $AboutRequests_{T,P}(t') = Requests_P(p)$

  $AboutUsers_{T,P}(t') = InUse_P(p)$

  When executed in a local state t such that $status_T(t) = RELEASING$ and the registration form of T is present in $AboutRegistered_{T,P}(t)$ and absent in $Applications_P(p)$, it results in the change of the local state of the agent that created T from a to a', where a' differs from a as follows:

$Controlled_A(a') = Controlled_A(a) - \{T\}$

$Competing_A(a') = Competing_A(a) - \{T\}$

and the task T leaves the game and the system.

- Sending a registration form F to the resource manager that controls a pool P of resources.

  This action may be executed in every local state t in which

  $status_T(t) = REGISTERING$ and $P \in Destinations_T(t)$.

  It does not change the local state of T. It results with a delay in the action of receiving F by the resource manager that controls P.

- The change of status from $REGISTERING$ to $ADMITTED$.

  This action is executed when T reaches a local state t such that

  $status_T(t) = REGISTERING$, $Destinations_T(t) = \emptyset$, and $AboutPredecessors_{T,P}(t) = \emptyset$ for all $P \in interest(T)$.

  It results in a state t' that differs from t as follows:

  $status_T(t') = ADMITTED$

  $Destinations_T(t') = interest(T)$

- Sending an admission form F to the resource manager that controls a pool P of resources.

  This action may be executed in every local state t in which

  $status_T(t) = ADMITTED$ and $P \in Destinations_T(t)$.

  It results in the local state t' that differs from t as follows:

  $Destinations_T(t') = Destinations_T(t) - \{P\}$

  With a delay it results in the action of receiving F by the resource manager that controls P.

- The change of status from $ADMITTED$ to $TRYING$.

  This action is executed when T reaches a local state t such that $status_T(t) = ADMITTED$, $Destinations_T(t) = \emptyset$, and $TicketsAt_T(t) \neq \emptyset$.

  It results in a state t' that differs from t as follows:

  $status_T(t') = TRYING$

  $Destinations_T(t') = interest(T)$

- Sending a request R to the resource manager that controls a pool P of resources.

  This action is executed in a local state t of in which $status_T(t) = TRYING$, $P \in Destinations_T(t)$, and the following condition is satisfied:

  (p3) For each $P' \in interest(T)$, the number of requests that precede in $AboutRequests_{T,P'}(t)$ all the actual or potential requests of T and come from tasks that have priorities not lower than the priority of T or have order forms in $AboutOrders_{T,P'}(t)$ is less than the the number of resources that are in $pool^{-1}(P')$ but not in $AboutUse_{T,P'}(t)$.

It results in a state t' that differs from t as follows:

$Destinations_T(t') = Destinations_T - \{P\}$

and, with a delay, in the action of receiving R by the resource manager that controls P.

- Setting the alarm-clock to the current value of the local clock of the agent that controls T and registering the sequence of blocking requests.

  This action is executed in a local state t in which $status_T(t) = TRYING$, T has all its requests in $AboutRequests_{T,P}(t)$ with $P \in interest(T)$, and the condition (p3) is satisfied.

  It modifies the local state to t', where $alarm - clock_T(t') = clock_A(a)$ for the agent $A = agent(T)$ and its local state a, and where $BlockingRequests_{T,P}(t')$ consists of those predecessors of the request of T in $AboutRequests_{T,P}(t)$ for $P \in interest(T)$.

- Noticing that the local clock of the agent that controls T has reached the value of the alarm-clock of T.

  This action is executed in a local state t if

  $clock_A(a) \geq alarm - clock_T(t) \neq \infty$

  for the agent $A = agent(T)$ and its local state a,

  $status_T(t) = TRYING$,

  T has all its requests in $AboutRequests_{T,P}(t)$ with $P \in interest(T)$, and the condition (p3) is satisfied.

  If for some $P \in interest(T)$ the proper prefix of the request of T in $AboutRequests_{T,P}(t)$ is different from $BlockingRequests_{T,P}(t)$ then the action modifies the local state to t', where $alarm - clock_T(t') = clock_A(a)$, and where $BlockingRequests_{T,P}(t')$ consists of those predecessors of the request of T in $AboutRequests_{T,P}(t)$ that have no corresponding order forms in $AboutOrders_{T,P}(t)$.

  If for each $P \in interest(T)$ the proper prefix of the request of T in $AboutRequests_{T,P}(t)$ equals to $BlockingRequests_{T,P}(t)$ then the action results in the local state t' that differs from t as follows:

  $alarm - clock_T(t') = \infty$

  $status_T(t') = ADVANCING$

  $Destinations_T = interest(T)$

- Sending an advancing form F to the resource manager that controls a pool P of resources.

  This action is executed in a local state t in which $status_T(t) = ADVANCING$, $P \in Destinations_T(t)$, T has all its requests in $AboutRequests_{T,P}(t)$ with $P \in interest(T)$, and the condition (p3) is satisfied.

  It results in a local state t' that differs from t as follows:

  $Destinations_T(t') = Destinations_T(t) - \{P\}$

  and, with a delay, in the action of receiving F by the resource manager that controls T.

- Sending a withdrawal W to the resource manager that controls a pool P of resources.

  This action is executed in a local state t in which $status_T(t) \in \{TRYING, ADVANCING, ORDERI$
  $P \in Destinations_T(t)$, and the condition (p3) is not satisfied.

  It results in the local state t' of T that differs from t as follows:

$Destinations_T(t') = Destinations_T(t) \cup \{P\}$

and, with a delay, in the action of receiving W by the resource manager that controls P.

- The change of status from $PLAYING$ to $ORDERING$.

  This action is executed in a local state t in which $status_T(t) = TRYING$, T has all its requests in $AboutRequests_{T,P}(t)$ with $P \in interest(T)$, and the condition (p3) is satisfied.

  It results in the local state t' that differs from t as follows:

  $status_T(t') = ORDERING$

  $Destinations_T(t') = interest(T)$

- Sending an order form F to the resource manager that controls a pool P of resources.

  This action is executed in a local state t in which $status_T(t) = ORDERING$, $P \in Destinations_T(t)$, T has all its requests in $AboutRequests_{T,P}(t)$ with $P \in interest(T)$, and the condition (p3) is satisfied.

  It results in a local state t' that differs from t as follows:

  $Destinations_T(t') = Destinations_T(t) - \{P\}$

  and, with a delay, in the action of receiving F by the resource manager that controls T.

- The change of status from $ORDERING$ to $WINNING$.

  This action is executed when T reaches a local state t such that $status_T(t) = ORDERING$, $Destinations_T(t) = \emptyset$, T has all its requests in $AboutRequests_{T,P}(t)$ with $P \in interest(T)$, and the condition (p3) is satisfied.

  It results in a state t' that differs from t as follows:

  $status_T(t') = WINNING$

  $Destinations_T(t') = interest(T)$

- Sending a winning form F to the resource manager that controls a pool P of resources.

  This action is executed in a local state t in which $status_T(t) = WINNING$ and $P \in Destinations_T(t)$.

  It results in a state t' that differs from t as follows:

  $Destinations_T(t') = Destinations_T(t) - \{P\}$

  and, with a delay, in the action of receiving F by the resource manager that controls P.

- The change of status from $WINNING$ to $ACCESSING$.

  This action is executed when T reaches a local state t such that $status_T(t) = WINNING$ and T occurs in the sets $AboutUsers_{T,P'}(t)$ for all $P' \in interest(T)$.

  It results in a state t' that differs from t as follows:

  $status_T(t') = ACCESSING$

  $Destinations_T(t') = interest(T)$

- The change of status from $ACCESSING$ to $RELEASING$.

  This action is executed in a local state t such that $status_T(t) = ACCESSING$.

  It results in a state t' that differs from t as follows:

$status_T(t') = RELEASING$

$Destinations_T(t') = interest(T)$

- Sending a releasing form F to the resource manager that controls a pool P of resources.

  This action is executed in a local state t in which $status_T(t) = RELEASING$ and $P \in Destinations_T(t)$.

  It results in a state t' that differs from t as follows:

  $Destinations_T(t') = Destinations_T(t) - \{P\}$

  and, with a delay, in the action of receiving F by the resource manager that controls P.

Notice that the actions of T preserve the consistency of the data constituting local states of T.

4 Behavioural properties of the protocol

Under reasonable assumptions about the system the protocol guarantees mutual exclusion of tasks accessing the same resources, starvation freedom of tasks, and as little as possible effect of potential failures. To illustrate how this can be shown we shall concentrate on the unfaulty behaviour.

To be realistic, we assume that the considered system *evolves with a finite speed* in the sense that only a finite number of events is possible in each bounded interval of time, and that it has the *finite delay property* in the sense that each event that is enabled sufficiently long is executed.

Under these assumptions we want to show the following two properties:

(1) a task cannot access a resource when the resource is accessed by another task,

(2) if a task is allowed to access one of the resources it needs to realize its plan then it is allowed to access all the resources it needs,

(3) each of the createded tasks is guaranteed to be executed.

The first two of these properties are direct consequences of the rules of the game of tasks for access to resources.

In order to prove the third property we define a task T to be *dependent* on a task T' if either $T = T'$ or there exists a state s of the system and a finite sequence $T_0 = T, T_1, ..., T_n = T'$ of tasks such that T_1,...,T_n are in s in the competition phase and for every two subsequent tasks T_i and T_{i+1} there exists a pool P_i of resources such that both T_i and T_{i+1} apply for a resource from P_i.

Property (3) is a direct consequence of the following four facts.

Fact 1.
If for a given supply of tasks the system reaches a state s such that a certain task T is in the competition phase then further events are possible due to tasks on which T depends. □

Proof outline:
Suppose the contrary.

We have a nonempty finite set $X(T, s)$ of the existing in s tasks on which T depends, and a finite set $Y(T, s)$ of pools that contain resources needed to tasks from $X(T, s)$.

The requests of tasks from $X(T,s)$ cannot be satisfied since otherwise one of these tasks could win the access to resources of its interest and thus a further events due to tasks from $X(T,s)$ would be possible.

Only members of $X(T,s)$ may have their requests in the queues $Requests_P(p)$ for $P \in Y(T,s)$ and the respective local state p. Moreover, either a member of $X(T,s)$ has all its requests distributed or it has none distributed, since otherwise either it could distribute a request or withdraw one already distributed, and thus a further event due to tasks from $X(T,s)$ would be possible.

The rules of the game ensure that in each queue $Requests_P(p)$ with $P \in Y(T,s)$ the number of requests that precede the request of a task $T' \in Y(T,s)$ and are from tasks of higher priority than that of T' does not exceed the number of free resources in the pool P.

As no further event due to tasks from $X(T,s)$ is possible, there must be a sequence T_0, T_1, ... of members of $X(T,s)$ and a sequence P_0, P_1,... of pools of resources such that the following conditions are satisfied for each $i = 0, 1, ...$ and for the respective local states p_0, p_1,...:

(1) T_i has its requests both in $Requests_{P_i}(p_i)$ as well as in $Requests_{P_{i+1}}(p_{i+1})$,

(2) the request of T_i in $Requests_{P_i}(p_i)$ has less predecessors than the number of free resources in the pool P_i,

(3) the request of T_i in $Requests_{P_{i+1}}(p_{i+1})$ has not less predecessors than the number of free resources in the pool P_{i+1},

(4) the priority of T_{i+1} is lower than that of T_i.

As the corresponding sequence of priorities is strictly decreasing, the sequence T_0, T_1,... must be infinite, which is impossible since the number of members of $X(T,s)$ is finite. □

Fact 2.
If for a given supply of tasks the system reaches a state s such that a certain task T is in the competition phase then, after a finite number of events due to tasks on which T depends, a certain task on which T depends is able to win the access to resources of its interest. □

Proof outline:
Suppose the contrary.

From Fact 1 it follows that there exists an infinite sequence of events due to the tasks on which T depends such that no state is reached in which a task on which T depends can win the access to resources of its interest.

As the system is finite and it evolves with a finite speed, there exists a moment y such that no task on which T depends is created at y or later that enters the competition phase in presence of T. Consequently, the set of tasks on which T depends, say $C(T)$, is completed in a state s' reached before y.

As the system has the finite delay property, the task with the highest priority among those belonging to $C(T)$ which exist is able sooner or later to distribute all its requests, and those requests are not preceded in the respective queues by more requests that the numbers of free resources in the respective pools. Consequently, this task is able to win the access to resources of its interest, which contradicts our assumption. □

Fact 3.
If for a given supply of tasks the system reaches a state s such that a certain task T is in the competition phase then, after a finite number of events due to tasks on which T depends, the task T is able to win the access to resources of its interest. □

Proof outline:
Let s' and $C(T)$ be respectively a state and a set of tasks as in the proof of Fact 2. As the system evolves with a finite speed, the set $C(T)$ is finite. According to Fact 2, after a finite number of events due to tasks on which T depends a task from $C(T)$ is able to win the access to resources of its interest and it leaves the game. Hence, after a finite number of events due to tasks on which T depends, also the task T is able to win the access to resources of its interest. $\square$

Fact 4.
If for a given supply of tasks the system reaches a state s such that a certain task T is created then, after a finite number of events due to tasks on which T depends, the task T is able to enter the competition phase. $\square$

Proof outline:
The task T can always register at the pools of resources of its interest. As the system evolves with a finite speed and the procedure of registration takes a finite time, the set of absolute predecessors of T is finite. As the system has the finite delay property, the set of tasks on which the absolute predecessors of T depend reaches its maximum, say $D(T)$, and, according to Fact 3, each of the absolute predecessors of T wins sooner or later the access to resources of its interest and leaves the game. Consequently, sooner or later T is admitted to take a numbered ticket and enter the competition phase. $\square$

References

[Aa 98] van der Aalst W. M. P., *The Application of Petri Nets to Workflow Management*, The Journal of Circuits, Systems and Computers 8 (1) 21-66 (1998)

[AS 90] Awerbuch B., Saks M., *A dining philosophers algorithm with polynomial response time*, Proc. of the 31st IEEE Symposium on Foundations of Computer Science, St. Louis, October 1990, 65-74

[CM 84] Chandy M., Misra J., *The drinking philosophers problem*, ACM Transactions on Programming Languages and Systems 6: 632-646 (1984)

[CS 93] Choy M., Singh A., *Efficient fault-tolerant algorithms for resource allocation in distributed systems*, ACM Transactions on Programming Languages and Systems 17: 535-559 (1995)

[CMM 97] Chavez, A., Moukas, A., Maes, P., *Challenger: A Multi-agent System for Distributed Resource Allocation*, Proceedings of the ACM Conference Agents'97

[Dem 00] Dembiński, P., *A distributed algorithm for dynamic resource allocation*, to appear

[FW 86] Friedlander, C. B., Wedde, F. H., *Distributed Processing under the DRAGON SLAYER Operating System*, Proceedings of the 15th International IEEE Conference on Parallel Processing, Pheasant Run Resort, August 1986

[GH 95] Georgakopoulos, D., Hornick, M., *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure*, Distributed and Parallel Databases 3, 119-153 (1995)

[KW 82] Korczynski, W., Winkowski, J., *A Communication Concept for Distributed Systems*, Inform. Process. Lett. 15(3)(1982)111-114

[Rh 98] Rhee I., *A modular algorithm for resource allocation*, Distributed Computing (1998) 11: 157-168

[WPP 91] Weidman E., Page I., Pervin W., *Explicit dynamic exclusion algorithm*, Proc. of the 3rd IEEE Symposium on Parallel and Distributed Processing, 142-149 (1991)

[Wink 81] Winkowski, J., *Protocols of Accessing Overlapping Sets of Resources*, Inform. Process. Lett. 12(5)(1981)239-243

[Wink 98] Winkowski, J., *A Multi-Agent System for Safe Distributed Task and Resource Management*, Proceedings of the workshop "Multi-Agent Day", Institute of Computer Science of the Polish Academy of Sciences, June 1998, 99-105

[Wink 00] Winkowski, J., *Resource Management in a Distributed Multi-Agent System*, ICS PAS Report Nr 905, February 2000