Modelling a grid

Citation for published version (APA):

Document status and date:
Published: 01/01/2007

Document Version:
Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 05. Jan. 2022
Modelling a Grid*

Carmen Bratosin, Natalia Sidorova and Wil van der Aalst

Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
c.t.bratosin,n.sidorova,w.m.p.v.d.aalst@tue.nl,

Abstract. In this paper we introduce a Petri net based formalism for modeling grid systems. Our grid model includes two perspectives. The first one is the user perspective, related to the definitions of workflows organizing the submission of processes to the grid. The second one is the resource perspective, related to resource constraints in the grid. We allow for a broader class of user-specified workflows than traditional grid workflow models based on directed acyclic graphs. Due to the use of Petri nets, our grid models can be analysed and verified with the use of standard techniques.

Keywords: Petri nets; grid computing; modeling.

1 Introduction

Grid computing [10, 15] emerged in the last decade as a new and challenging research perspective in computer science. The potential of grid is in problem solving by using world wide distributed resources. The applications areas of grid computing include physics, astronomy, biology (see e.g. [1, 3]). Grid computing requires hardware and software infrastructure that provides dependable, consistent, pervasive, inexpensive access to high-end computational capabilities and resource coordination. Several tools have been developed like Globus [2] and Condor [19], that support submitting requests to a grid.

Although grid computing is becoming more and more popular in different areas, very little attention has been paid to the formalisation of the grid concept. Recently, formal descriptions were proposed for user-defined processes. The main idea there is that the user describes his application as a grid workflow, where a grid workflow is a mean for the automation of processes which involves the orchestration of a set of grid services, agents and actors that must be combined together to solve a problem or to define a new service [11, 7, 9, 14, 18]. Most of the formalisms proposed are rather restrictive, e.g. they do not allow for complex behavior such as loops. In fact the most widely used formalism for grid workflows is the directed acyclic graph [11].

Introducing the concept of a grid workflow is a big step forward as it gives users a possibility to define their processes in a structured manner and makes

* This research is supported by the GLANCE NWO project “Workflow Management for Large Parallel and Distributed Applications”.
scheduling more clear and efficient. Still, one needs a complete picture of a grid to formulate correctness properties and develop verification techniques. This complete picture should contain not only the user perspective, but also the resource one (e.g. the degree of resource availability).

In this paper, we propose a formalism for modeling a grid that enhances the user perspective with resource information. Petri nets (PN) is a very popular model for workflows in other application domains (see [5]) because they allow an easy definition of processes, including complex patterns (see [16]). Therefore in our model, the user describes processes in terms of Petri nets. Jobs submitted to the grid are modeled as tokens on distinguished job places of a PN; every token on such a job place contains information about the resource where the corresponding job is allocated. The workflow definition itself is grid-independent and can be (re-)used for different grids, since it only includes requirements on resources for every job type. The grid model includes the information about resource capacities that restrict the behavior of the grid. Based on the job requirements, the grid assigns a resource to execute the job, which is reflected in the semantics we define for the grid. We allow grid evolution by adding/deleting user processes or by the evolution of one of the user processes.

The rest of the paper is organized as follows. In Section 2 we introduce some basic definitions. In Section 3 we present our grid model and in the Section 4 we describe its semantics. Conclusions and directions for future work are given in Section 5.

2 Preliminaries

As usual, \( \mathbb{N} \) denotes the set of natural numbers and \( \mathbb{Z} \) denotes the set of integer numbers.

**Bags** Let \( A \) be a finite set. A bag \( b \) over \( A \) is a function \( b : A \rightarrow \mathbb{N} \).

The set of all bags over the set \( A \) is denoted \( \mathbb{N}^A \).

*Observation:* We write e.g. \( a = 2 \ast [x] + [y] \) for a bag \( a \in \mathbb{N}^A \) with \( a(x) = 2 \), \( a(y) = 1 \), for \( x, y \in A \) and \( a(i) = 0 \) for all \( i \in A \setminus \{x, y\} \).

Let \( a, b, c \in \mathbb{N}^A \). Then:

- \( a \mathcal{R} b \) iff \( \forall i \in A : a(i) \mathcal{R} b(i) \), where \( \mathcal{R} \in \{=, \geq, \leq\} \) and ,
- \( c = a + b \) iff \( \forall i \in A : c(i) = a(i) + b(i) \),
- \( c = a - b \) iff \( \forall i \in A : c(i) = a(i) - b(i) \), and \( a \geq b \), and it is undefined for \( a \not\geq b \)

**Function modification** Let \( f : A \rightarrow B \) and \( a \notin A \). Then the function extension \( f \oplus (a, b) \) is a function \( f \oplus (a, b) : A \cup \{a\} \rightarrow B \cup \{b\} \)
defined as
\[(f \oplus (a, b))(a) = b, \text{ and} \]
\[(f \oplus (a, b))(x) = f(x), \text{ for every } x \in A.\]

Now, let \(f : A \rightarrow B\) and \(a \in A\). Then the function \(\text{reduction } f \ominus (a, f(a))\) is a function:
\[
f \ominus (a, f(a)) : A \setminus \{a\} \rightarrow B
\]
defined as:
\[(f \ominus (a, f(a)))(x) = f(x), \text{ for every } x \in A \setminus \{a\}.\]

**Definition 1 (Petri net).** A Petri net (PN) is a net 
\[N = (P, T, F, l)\]
where \(P\) is a set of places, \(T\) is a set of transitions, \(F \in \mathbb{N}^{(P \times T) \cup (T \times P)}\) is a bag of arcs and \(l : P \rightarrow \Sigma \) (\(\Sigma\) is a set of labels) is a labeling partial function.

We define the following notations:
1. \(\forall x : x \in P \cup T : x = \{y \in P \cup T | F(y, x) \neq 0\}\),
2. \(\forall x : x \in P \cup T : x = \{y \in P \cup T | F(x, y) \neq 0\}\),
3. \(\forall X \subseteq P \cup T : X = \bigcup_{x \in X} (x \bullet)\),
4. \(\forall X \subseteq P \cup T : X \bullet = \bigcup_{x \in X} (\bullet x)\).

## 3 Grid Model

The architecture of a grid is composed of several layers [10]: the application layer, where the grid user describes the processes to be submitted to the grid, the middleware layer, which is in charge of finding a resource for the user requirements and other management issues (e.g. monitoring, fault-recovery), and, the infrastructure layer composed from resources.

Based on this architecture, we identify the three elements that a grid model must contain: (1) a description of user processes with associated requirements, (2) a list of the infrastructure components (i.e. resources) with their properties (e.g. RAM, HDD, softwares) and availability information, and (3) a mechanism that links the resource view with the user view.

We build our grid model, under the hypothesis that the infrastructure is fixed and will not change (e.g. resources cannot be added or removed to/from the grid) and all resources belong to the same class (e.g. computing elements).

We write \(\square\) for a dummy resource, which is used when actually no resource is required.

**Definition 2 (Grid).**
A grid is a tuple \(G = (R, \Pi, J, Q, cap, req)\), where:

- \(R\) is a finite set of resources;
- \(\Pi\) is a finite set of properties;
- \(J\) is a set of jobs;
\( Q \) is the set of all marked Petri Nets of the type \(((P^j \cup P^n, T, F, l), M) \in Q\), where:

- \((P^j \cup P^n, T, F, l)\) is a Petri net, where \(P = P^j \cup P^n\) with \(P^j\) being a non-empty set of job-holder places and \(P^n\) a non-empty set of untyped places, and \(l : P^j \rightarrow J\) being a labeling function that maps every job-holder place to a job type.
- \(M \in N^{(P^j \times (R^j \cup \{\square\})) \cup (P^n \times \{\square\})}\) is a marking;
- \(cap : R \rightarrow N^\Pi\) is a function mapping a resource to the bag of properties it has;
- \(req : J \rightarrow N^\Pi\) is a function mapping every job to the bag of properties it requires from a resource.

For all \(p \in P^j\), where \(P^j = \bigcup_{((P^j \cup P^n, T, F, l), M) \in Q} P^j\), the following conditions hold:

1. \(|(\bullet p) \cap P^j| = |\bullet (p\bullet) \cap P^j| = 1\). The input transition of a job place has only one output job place. Also, the output transition of a job place has only one input job place.
2. \(F(p, t) = 1\), for any \(t \in p\bullet\), i.e. every transition terminates at most one job.
3. \(F(t, p) = 1\), for any \(t \in \bullet p\), i.e. every transition can allocates at most one job.

According to this model, a grid user will describe his processes in terms of PNs. The set of places of a PN is split into the set of job-holder places and the set of untyped places. The tokens in the PN are colored by a real resource or by the dummy resource. A token on a job-holder place represents a job allocated on some resource. All untyped places receive the dummy resource \(\square\) as color.

The conditions express the fact that a transition may not allocate multiple jobs to resources at the same time. Also, by symmetry, a transition may not release multiple resources at the same time. The motivation of this choice is given by the need to have a clear view when and how a job is submitted to a certain resource.

The user processes are independent of the infrastructure of the grid. The PNs describing user processes do not contain any information related to concrete resources. The only information available is the requirements that a resource should fulfill in order to host a certain job. A user process defined in such a way can be reused and submitted to a different grid without any modification.

**Example 3.** A company wants to analyze the logs related to an administrative process. The analysis require a lot of computer power due to the size of logs. Therefore, the company wants to use a grid for the analysis. The logs are split in \(n\) batches. The model of the process is given in Figure 1.

The set \(P^j\) in Figure 1 contains three job places: \(p^j_1\), whose tokens represent the execution of algorithm 1 for one batch on some resources, \(p^j_2\), where in case the conformance test fails a second algorithm is performed for the corresponding batch, and, finally, \(p^j_3\) where all the results are combined. All the input transitions of these places have also the function of “claiming” a resource for a job execution, and the output transitions the one of “releasing” a resource.
For simplicity we assume that the grid has just two resources, $r_1, r_2$, and each one has capacity 2, while every job requires capacity 1.

For this example $G = (R, \Pi, J, Q, \text{cap}, \text{req})$, where:

- $R = \{r_1, r_2\}$;
- $\Pi = \{\text{execution}\}$;
- $J = \{\text{algorithm1, algorithm2, combining results}\}$;
- $((P^j \cup P^n, T, F, l), M) \in Q$, where:
  - $P^j = \{p_{j1}, p_{j2}, p_{j3}\}$;
  - $P^n = \{p_1, p_2, p_3, p_4, p_5, p_6\}$;
  - $l : P^j \rightarrow J$ such that: $l(p_{j1}) = \text{algorithm1}, l(p_{j2}) = \text{algorithm2}, l(p_{j3}) = \text{combining results}$;
  - $M \in \mathbb{N}^{(P^j \cup P^n) \times (R \cup \{\Pi\})}$;
- cap : $R \rightarrow \mathbb{N}^\Pi$, such that cap$(r_1, \text{exec}) = \text{cap}(r_2, \text{exec}) = 2$;
- req : $J \rightarrow \mathbb{N}^\Pi$, such that req$(\text{algorithm1, exec}) = \text{req}(\text{algorithm2, exec}) = \text{req}(\text{combining results, exec}) = 1$.

When a user submits a marked PN associated with his process, the grid creates a process instance and maps it with the user marked PN.

The state of the grid is given by the combination of the markings of all active process instances.

**Definition 4 (Grid state).** A state $S$ of a grid $G = (R, \Pi, J, Q, \text{cap}, \text{req})$ is defined as a pair $(PI, \beta)$ where:

- $PI \subseteq U_{PI}$ is a set of process instances and $U_{PI}$ is the universe of all process instances,
- $\beta : PI \rightarrow Q$, is a function defining the state of every process instance, given by a marked Petri net.

Given a grid state, the availability of the resources can be computed as follows:
Definition 5 (Resource availability).
The occupation of the resources by process instances in a grid
\( G = (R, \Pi, J, Q, \text{cap, req}) \) with current state \( S = (\Pi, \beta) \), is given by the function
\[
\text{occup}_{\Pi} : (\Pi \times R) \rightarrow \mathbb{N}^\Pi
\]
such that
\[
\text{occup}_{\Pi}(i, r) = \sum_{p \in \Pi} M(p, r) \ast \text{req}(l(p))
\]
for every \( i \in \Pi \) with \( \beta(i) = ((P^j \cup P^n, T, F, l), M), r \in R \).
The resource availability is given by the function
\[
\text{free} : R \cup \{\square\} \rightarrow \mathbb{Z}^\Pi
\]
such that:
\[
\text{free}(r) = \begin{cases} 
\text{cap}(r) - \sum_{i \in \Pi} \text{occup}_{\Pi}(i, r) & \text{for every } r \in R, \\
0, & r = \square.
\end{cases}
\]

4 Grid semantics

The semantics of a grid is given by a transition system, with three types of transitions: (1) creation of a new process instance to be executed on the grid, (2) a deletion of a process instance (user cancellation or the process instance has finished its run), and (3) progress of a process instances due to its execution. We use the interleaving semantics, i.e. only one event can occur at a time.

Adding a new process instance \( \text{add}(i, q) \) When a user creates a new instance, the initial marking of the associated marked PN may not contain marked job places, since jobs must be allocated first.

For a grid \( G = (R, \Pi, J, Q, \text{cap, req}) \) with a current state \( S = (\Pi, \beta) \), a new state \( S' = (\Pi \cup \{i\}, \beta \oplus (i, q)) \) can be reached from \( S \) by adding a new instance \( i \in \Pi \) \( \setminus \Pi \) with an initial state \( q \in Q \) where \( q = ((P^j \cup P^n, T, F, l), M) \) with \( M(p, r) = 0 \) for every \( p \in P^j, r \in R \cup \{\square\} \), denoted as:
\[
S \xrightarrow{\text{add}(i, q)} S'. \quad (1)
\]

Deleting a process instance \( \text{del}(i) \) For a grid \( G = (R, \Pi, J, Q, \text{cap, req}) \) with a current state \( S = (\Pi, \beta) \), a new state \( S' = (\Pi \setminus \{i\}, \beta \ominus (i, \beta(i))) \) can be reached from \( S \) by deleting instance \( i \in \Pi \), denoted as:
\[
S \xrightarrow{\text{del}(i)} S'. \quad (2)
\]
Fig. 2. Firing rule: no resource-bound firing

Process instance evolution $ev(i, (t, r_1, r_2))$. Let $G = (R, \Pi, J, Q, cap, req)$ be a grid with current state $S = (PI, \beta)$, $i \in PI$ and $\beta(i) = q_1 = ((P_i \cup P^m, T, F, l), M_1)$. A new state $S' = (PI, (\beta \ominus (i, q_1)) \oplus (i, q_2))$ can be reached by the evolution $ev(i, (t, r_1, r_2))$ of instance $i$:

$$S \xrightarrow{ev(i, (t, r_1, r_2))} S'$$

where $t \in T$, $r_1, r_2 \in R \cup \{\square\}$, if $q_2 \in Q$ is reachable from $q_1$ (denoted $q_1 \xrightarrow{(t, r_1, r_2)} q_2$) by the firing of $t$ that releases resource $r_1$ and claims resource $r_2$, according to the defined below firing rule.

**Firing rule.** A process instance can change its state (marking) when a transition fires. In terms of classical PNs, a transition may only fire if the input places have enough tokens (the transition is enabled). If none of the input/output places are job-holder places, the same firing rule can be applied to a process instance in a grid (see Figure 2).

When the job places are involved in the firing, the enabling condition must be enhanced with the resource perspective, since the job place tokens are resource colored. When a transition fires, only one resource-colored token will be produced and/or only one resource-colored token will be consumed (cf. Definition 2).

If a transition “allocates” a job to a resource $r_2$, i.e. it is an input transition for a job place (see Figure 3), the effect of the firing is a new job running on resource $r_2$. Therefore, the transition may be enabled only if there is an available resource that satisfies the job requirements.

If a transition “deallocates” a job, i.e. it is an output transition for a job place (see Figure 4), then the firing corresponds to the release of the resource hosting the job. Therefore it is important to know which of the resources gets released from this job.

Finally a transition can be input and output for (a) job place(s) at the same time (Figure 5).

Note that the firing rule allows immediate allocation of jobs to resources that are just released by the same firing.
Fig. 3. Firing rule: resource-bound case 1

Fig. 4. Firing rule: resource-bound case 2

Fig. 5. Firing rule: resource-bound case 3
We will consider two “markings”

\[
\text{cons}(t, r_1, r_2), \quad \text{prod}(t, r_1, r_2) \in \mathbb{N}^{(P^3 \times (R \cup \{\square\})) \cup (P^n \times \{\square\})}
\]

where \text{cons} contains the tokens to be consumed during the firing, \text{prod} contains the tokens produced by the firing, \( t \) is the firing transition, and \( r_1 \) and \( r_2 \) are the claimed resource and the released resource by the firing (Observation By resource we understand also the dummy resource \( \square \)).

Let \( t \) be the transition in question. We pick two places, \( p_{j_1} \) and \( p_{j_2} \) such that \( \{p_{j_1}\} \supseteq \bullet \cap P^j \) and \( \{p_{j_2}\} \supseteq \bullet \cap P^j \) (where possibly \( p_{j_1} = p_{j_2} \)). Note that for transitions without input/output job places, \( p_{j_1}/p_{j_2} \) may be an arbitrary job places.

Then we choose \( \text{cons}(t, r_1, r_2) \) and \( \text{prod}(t, r_1, r_2) \) in such a way that the following conditions are satisfied:

\[
\text{cons}(t, r_1, r_2) = \begin{cases} 
F(p, t), & \text{for every } (p, r) \in P^n \times \{\square\}, \\
F(p_{j_1}, t), & \text{for } (p, r) = (p_{j_1}, r_1), \text{ for } r_1 \in R, \\
0, & \text{for } (p, r) \in (P^j \times \{\square\}) \cup ((P^j \times R) \setminus \{p_{j_1}, r_1\}).
\end{cases}
\]

and,

\[
\text{prod}(t, r_1, r_2) = \begin{cases} 
F(t, p), & \text{for every } (p, r) \in P^n \times \{\square\}, \\
F(t, p_{j_2}), & \text{for } (p, r) = (p_{j_2}, r_2), \text{ for } r_2 \in R, \\
0, & \text{for } (p, r) \in (P^j \times \{\square\}) \cup ((P^j \times R) \setminus \{p_{j_2}, r_2\}).
\end{cases}
\]

We say that a marking \( M_1 \) enables transition \( t \) with respect to \( r_1, r_2 \) iff

1. \( M_1 \geq \text{cons} \), and
2. \( \text{free}(r_2) + \text{cons}(p_{j_1}, r_2) \ast \text{req}(l(p_{j_1})) \geq \text{prod}(p_{j_2}, r_2) \ast \text{req}(l(p_{j_2})). \)

Note: If \( r_2 = \square \) (i.e. no resource is necessary to be allocated), the second condition always holds.

The marked Petri Net \( q_2 = ((P^j \cup P^n, T, F, l), M_2) \) is reachable from \( q_1 \) by the firing of transition \( t \), denoted \( q_1 \xrightarrow{(t,r_1,r_2)} q_2 \), iff \( t \) is enabled and: \( M_2 = M_1 - \text{cons} + \text{prod} \).

Example 6. For the model from Example 3, Figure 6 presents a possible evolution of the grid.

The following two invariant properties, that one naturally expects to have in a grid, hold during the grid evolution according to the semantics defined above:

**Lemma 7 (Markings invariant).**

Let \( G = (R, H, J, Q, \text{cap}, \text{req}) \) be a grid. Then, for all \( i \in P^1 \) and for all its reachable markings \( \beta(i) = ((P^j \cup P^n, T, F, l), M) \),

\[
M(p, \square) = 0, \text{ for every } p \in P^j.
\]
Lemma 8 (State invariant).
Let grid $G = (R, \Pi, J, Q, \text{cap}, \text{req})$. Then, during the evolution, for every state $S = (P, I, \beta)$ reachable from the initial state $S_0 = \{\emptyset, \emptyset \rightarrow Q\}$:

$$\text{free}(r) \geq 0, \text{ for every } r \in R.$$ 

Marking invariant ensures that there is no job that runs nowhere, and state invariant that resource availability has to be always greater or equal to 0. If this invariants are not fulfilled by grid evolution, then the grid does not act as it should. In our case, the grid semantics guarantees that the two invariants hold.

5 Conclusion

In this paper we proposed a formalism for modeling grid systems. Our formalism allows the user to define his process in terms of Petri Nets where components to be submitted to the grid are modeled as distinguished places, called job-holder places. The tokens on these places are colored with resources on which the corresponding jobs are running. We defined an interleaving semantics for evolution of the grid.

Our model can also be seen as a two-layered nested net [12, 20, 6], where the upper layer reflects the dynamics on the grid infrastructure and the token nets correspond to workflow instances.

Related work Our grid model is conceptually close to resource-constraint workflows [13, 21] and flexible manufacturing systems [8, 17]. As opposed to our approach, resources are modeled there as resource places and their (available) capacities are defined by the number of tokens on those places. We want however
Fig. 7. A grid model to a resource-constrained workflows model

to model grid workflows in such a way that the workflow would be reusable on
different grids, and therefore we do want to incorporate information related to
the resource demands of jobs into our model, but we do not want to include in-
formation about the exact infrastructure of the grid into it. Our grid model can
still be translated into a resource-constrained workflow net allowing for multiple
tokens on the initial place. Note that the translation would be less readable than
the original model (see an example in Figure 7).

Future work Since we are interested in the verification of correctness properties
on our grid model, our next step will be providing a translation for our grid
models to classical (uncolored) PNs. Another direction for future work is analysis
of the effect of different scheduling approaches on the overall grid behavior.
We plan to implement our framework using the YAWL workflow management
system [4] and the Globus toolkit [2].

References