Abstract State Machine pour la Sémantique Opérationnelle d'un Grid

Frédéric Teller

Faculté Universitaire Notre-Dame de la paix à Namur
21 Rue Grandgagnage 5000 Namur Belgique
fteller@info.fundp.ac.be

Résumé Actuellement, le terme *Grid* est fortement utilisé dans le domaine de l'informatique distribuée. Néanmoins, une définition précise n'est pas encore communément acceptée. C'est pour cela que cet article traite d'une proposition de deux auteurs [1] qui modélisent avec le langage formel de l'*abstract state machine* (ASM), la sémantique opérationnelle d'un *Grid*. Cette modélisation conceptuelle de la sémantique d'un *Grid* identifie ses caractérisations et fonctionnalités en terme d'abstraction de ressources et d'utilisateurs. En acceptant communément des hypothèses initiales sur la notion de ressource dans le cadre de l'informatique distribuée et dans le cas du *Grid*, l'apport de cette formulation abstraite est de pouvoir définir facilement et précisément la frontière entre un système distribué ordinaire et un *Grid*.

Mots clés : Abstract State Machine, Grid, Système Distribué.

1 Introduction

En informatique actuelle, le concept de *Grid* rencontre beaucoup d'intérêts. Des projets de recherche sont entretenus dans des laboratoires à travers le monde et des logiciels publics et commerciaux applicables sur *Grid* se multiplient pour différents domaines d'application. Néanmoins, une caractérisation précise du concept de *Grid* n'existent pas. De multiples articles [2] de différents experts en systèmes distribués et de nombreuses communautés [3] autour du *Grid* n'ont pas pu dégager une définition commune de la notion du *Grid*. Ainsi, les fonctionnalités et caractéristiques du *Grid* restent floues dans la vague des idées qui entourent les termes comme *Computional Grids*, *Data Grids*, *High-Throughput Grids*, *Bio Grids* ou encore *Cluster Grids*.

Un essai de clarification de ce concept est tenté dans un article de Zsolt Németh et Vaidy Sunderam [1]. Au lieu de se baser sur la statique des composants qui constituent un *Grid*, ces deux auteurs décrivent les caractéristiques du *Grid* selon sa sémantique opérationnelle. Pour identifier les fonctionnalités déterminantes d'un *Grid*, ils utilisent une approche formelle basée sur la méthode de l'*Abstract State Machine* (ASM). Plus particulièrement, leur modélisation de la sémantique du *Grid* dans ce langage permet d'identifier les attributs qui différencient clairement les *Grids* de tous les autres systèmes distribués.

Cet article commence par des idées issues de la littérature sur des notions comme le partage des ressources et l'hétérogénéité des composants qui entourent

le concept de *Grid*. Aussi, nous allons parcourir dans une deuxième section les concepts de base de l'ASM et de son fondement mathématique des *evolving algebras*. Ensuite, en se basant sur ces deux sections introductives, nous allons décrire les différences entre les modélisations en ASM proposer par Zsolt Németh et Vaidy Sunderam du *Grid* d'une part et d'un système distribué d'autre part. Finalement, nous discutons dans une réflexion critique les conclusions que les deux auteurs dérivent de leurs modélisations.

2 Élaboration des caractéristiques d'un *Grid*

Pour comprendre les défis d'une modélisation formelle d'un *Grid*, cette première section introductive donne un petit état d'art des caractéristiques qui spécifient un système *Grid*. Partons pour cet état d'art d'une image intuitive et originaire du *Grid* en le comparant à un réseau d'électricité. Comme ce dernier, qui lui fournit de l'électricité à un ensemble important d'utilisateurs, un système *Grid* est un réseau qui fournit des services informatiques (puissance de calcul, mémoire, ...) en regroupant des ressources (CPU, place mémoire, entrées-sorties, ...) disponibles dans le réseau.

Dans la plupart des publications qui tentent de définir la notion d'un *Grid*, nous trouvons une liste de caractéristiques informelles qui décrivent un réseau de ressources mis à disposition d'un ensemble d'utilisateurs. Souvent, ces listes donnent une vue statique des composants, des couches, des protocoles ou des interfaces qu'un système *Grid* doit disposer. Par exemple, un ensemble de caractéristiques informelles inspirées des articles [4,5] est : (1) Taille importante : un *Grid* doit être capable de gérer des milliers de ressources. (2) Distribution géographique : les ressources d'un *Grid* peuvent se trouver à des locations différentes et distantes. (3) Hétérogénéité : un système *Grid* comprend en général des logiciels et des architectures physiques différents. (4) Partage des ressources : les ressources d'un *Grid* sont possédées par différentes organisations, ainsi l'accès aux ressources doit être possible par d'autres organisations et utilisateurs. (5) Administration multiple : chaque organisation peut établir des politiques de sécurité et d'accès différentes pour les ressources qu'elle possède. (6) Coordination des ressources : afin de livrer des capacités de calcul agrégé, les différentes ressources doivent être coordonnées pour travailler coopérativement. (7) Accès transparent : l'accès aux ressources distantes doit être transparent pour l'utilisateur. Pour cacher l'hétérogénéité des ressources et pour garantir une extensibilité, on exige souvent d'un système *Grid* d'être basé sur des protocoles et interfaces standard. (8) Qualité : les services délivrés par un *Grid* doivent respecter un niveau de besoin en qualité (*QoS*). Par exemple, on exige souvent d'un grid de fournir des capacités de calcul de hautes performances. (9) Gestion dynamique : un *Grid* doit garantir l'accès aux ressources disponibles en s'adaptant à l'environnement dynamique où des pannes sont fréquentes.

Souvent les *Grids* et leur partage et utilisations de ressources sont vus comme les successeurs du système distribué [6]. Cependant, dans la liste donnée au paragraphe précédent, nous remarquons que les caractéristiques sont présentes dans

l'un ou l'autre système distribué ou coopératif. Ainsi, en décrivant d'une manière informelle et statique les composants d'un *Grid*, il est difficile de déterminer une frontière précise entre un système distribué ordinaire[1] et un *Grid*. Par exemple, la question à partir de quelle distribution géographique un système distribué devient un *Grid* est difficile à déterminer.

Une approche plutôt horizontale de ce qui a été proposé jusqu'à présent est suivie par l'*Open Grid Services Architecture* (OGSA) [7]. Comme nous allons le voir aussi pour le modèle formelle du *Grid* dans la section 4, l'OGSA essaye de définir les fonctionnalités caractérisantes d'un *Grid*. Pour l'OGSA, un *Grid* peut se spécifier par un ensemble de services à fournir. (Un service est une entité qui peut fournir une capacité à travers un réseau.) L'OGSA spécifie en plus des interfaces et protocoles standards, la sémantique des services. Par exemple, le cycle de vie (création, gestion de l'exécution et destruction) d'un service est défini par l'OGSA. Naturellement, les services devront être supportés par les ressources disponibles dans un *Grid*. La notion d'*organisation virtuelle* de l'OGSA reprend ainsi différentes ressources distribuées, hétérogènes et dynamiques. Dès lors, le partage de ces ressources entre différentes organisations (virtuelles) est réalisé par des mécanismes de *Web Services*. (Un service qui est accessible par un standard internet comme XML.) Remarquons déjà que la virtualisation de milliers de ressources distribuées en services constitue la différence entre un *Grid* et un système distribué. En effet, cette abstraction obligée des ressources en services utilisables via internet rend d'une part possible l'implémentation de la liste informelle de caractéristiques d'un *Grid* au-dessus et n'est, d'autre part, pas présente dans des systèmes distribués ordinaires. Cette fonctionnalité qui consiste à faire abstraction des ressources d'un *Grid* pour réaliser le partage des ressources et à cacher l'hétérogénéité des composants du système est une des idées principales dans le modèle formel en-dessous.

3 L'*abstract state machine* (ASM) et *evolving algebras*

Pour le modèle formel de la sémantique d'un *Grid*, nous devons avoir un langage qui permet d'exprimer formellement cette sémantique. Les auteurs du modèle ont choisi l'*abstract state Machine* (ASM) [8] qui a sa base mathématique dans les *evolving algebras* [9]. Nous allons introduire dans cette section ces formalismes.

Dans la logique du premier ordre, une structure est un ensemble non-vide avec des fonctions et relations. En mathématique usuelle, une structure est appelée *algèbre*. Définissons les structures en termes d'ASM. Les ensembles non-vides sont appelés des univers. En ASM, les éléments d'un univers représentent des objets de la réalité. Par exemple, l'univers COULEUR contient les éléments *rouge* et *bleu*. Les noms et arités des fonctions et relations de la structure sont regroupés dans un ensemble fini appelé signature. La signature représente les relations entre objets de la réalité. Par exemple, une signature ϕ contient les

[1] Par exemple des middelware CORBA ou JavaRMI mais aussi des systèmes de calcul parallèle comme MPI ou PVM.

noms des fonctions *col* : TOKEN → COULEUR et *val* : TOKEN → VALEUR. De plus, le formalisme ASM ajoute des symboles *true*, *false* et *undef* (fonctions partielles), l'opération d'égalité = et les opérations standards sur les booléens.

Un état (*state* ou *static algebra*) A de signature ϕ est un univers X combiné avec une interprétation des noms de fonctions de ϕ sur X. Par exemple, si X est un super-univers composé des univers TOKEN, COULEUR et VALEUR. Prenons la signature ϕ déjà définie. Supposons que TOKEN contient un élément t qui est rouge et a la valeur 2. L'état A peut être représenté par X avec les interprétations : $col(t) = rouge, val(t) = 2$.

Pour simuler un comportement dynamique ASM introduit un mécanisme qui permet de mettre à jour les valeurs des fonctions pour certains arguments. Ce mécanisme d'*update* (:=) réalise les transitions entre états. Dans ce principe des *evolving algebras* une transition est alors une règle dont l'exécution pour transformer un état dépend d'une condition. Par exemple la règle

```
if col(t) = rouge then
   val(t) := val(t) + 1
   col(t) := bleu
```

décrit l'activité du système modélisé. Pour notre état A, la condition est satisfaite et le corps de la règle est donc exécuté. Après cette exécution, la signature ϕ peut être interprétée sur X comme $col(t) = bleu$ et $val(t) = 2$ ce qui représente un nouvel état B.

Un modèle ASM s'exécute en séquentiel vu que le modèle traite seulement une règle à la fois. Remarquons qu'il est possible que des conditions de plusieurs règles soient vérifiées à un moment. Dans ce cas, l'ordre dans lequel ces règles s'exécutent n'est pas déterministe. Pour simuler explicitement un comportement parallèle, le formalisme ASM possède une notion d'agent ou de module. Chaque agent ou module représente un modèle ASM (séquentiel) qui s'exécute en parallèle avec les autres agents. Ainsi, nous avons un modèle ASM *multi-agent* pour simuler les modèles parallèles.

Avec la définition des univers et d'une signature, nous pouvons donc modéliser un système réel dans le langage formel ASM. Ainsi, ce formalisme permet de modéliser une nature hautement abstraite d'un système. Aussi il permet d'exprimer la sémantique opérationnelle du système par des règles qui s'exécutent dès que leur condition est vérifiée. Semblable à une *machine de Turing universelle*, ASM est un formalisme général qui peut modéliser chaque procédure effective. Ainsi, il existent différents logiciels qui permettent d'exécuter un modèle ASM [10]. Finalement, ASM a la capacité de modéliser un système à différents niveaux d'abstraction. Par exemple, une fonction hautement abstraite dans un modèle peut être spécifiée plus en détail dans un modèle ASM de plus bas niveau. Ainsi, il est possible de transformer un modèle abstrait (exprimant une sémantique) par plusieurs étapes de spécialisation en un modèle concret (implémentant un protocole réel), tout en prouvant l'équivalence sémantique entre les différents modèles ASM intermédiaires.

4 Deux modèles ASM : *Grid* vs système distribué

L'enjeu fondamental de la modélisation d'un *Grid* est de montrer la différence avec un système distribué traditionnel. C'est pour cela que nous allons nous baser sur un article de Zsolt Németh et Vaidy Sunderam [1] qui décrit d'abord un modèle formel en ASM d'un système distribué classique et qui montre ensuite les modifications de ce modèle pour le *Grid*. La modélisation complète des deux systèmes dépasse le cadre de cet article. Nous allons nous concentrer sur les différences fondamentales entre les deux modélisations. Le lecteur intéressé à en apprendre davantage doit alors se référencer à l'article de base pour les modélisations complètes.

Afin d'augmenter ces performances, une application distribuée exploite des ressources qui ne sont pas nécessairement locales. Souvent, on regroupe dans une idée d'abstraction les ressources d'un environnement distribué dans une *machine virtuelle* pour que les utilisateurs et leurs applications puissent y accéder virtuellement en local. Les deux auteurs de l'article de base ont identifié des différences dans la manière dont une telle machine virtuelle est établie à partir des ressources disponibles entre un système distribué et un *Grid*. Dans les deux prochains paragraphes, nous décrivons les hypothèses initiales que les auteurs prennent pour le regroupement et l'accès aux ressources dans les deux systèmes.

Un système distribué est souvent constitué d'un pool de noeuds de calcul. (Un noeud de calcul est un ensemble d'ordinateurs mono- ou multi-processeurs qui fournit des services de puissance de calcul.) Un sous-ensemble de ces noeuds de calcul forme alors une machine virtuelle pour un utilisateur et ces applications. Nous pouvons supposer que cet utilisateur a un *login* valide sur chaque noeud de la machine virtuelle. Ainsi, les applications de l'utilisateur ont accès aux ressources locales des noeuds du pool virtuel. De plus, comme l'utilisateur a un compte sur les noeuds de la machine virtuelle, il peut avoir connaissance des caractéristiques physiques d'un noeud. En général, un tel pool virtuel est assez statique, vu que les changements dans les compositions des noeuds et leurs ressources sont rares.

A côté du nombre de ressources qui se situe pour un *Grid* entre 1000 et 10000 par rapport au système distribué qui utilise entre 10 et 100 ressources, la machine virtuelle d'un *Grid* est composée d'un pool virtuel de ressources multi-types[2] au lieu de se baser sur un ensemble de noeuds axés sur des ressources de calcul. Les ressources d'un pool virtuel peuvent être distribuées géographiquement et à travers différents domaines de sécurité. Vu que les ressources ne font plus nécessairement parties d'un noeud où l'utilisateur est connu, l'accès à une ressource se fait par un mécanisme d'authentification (*credential*) chez le propriétaire de la ressource. C'est aussi uniquement le propriétaire d'une ressource qui a la connaissance physique de celle-ci. Aussi, à cause du nombre de ressources disponibles dans le système, l'utilisateur ignore cette information et se base sur une gestion dynamique des ressources disponibles. En effet, dans un

[2] Autre que la puissance de calcul on trouve des ressources de mémoires, d'entrées/sorties, ...

Système distribué traditionnel	Grid
Un pool virtuel de noeuds de calcul	Un pool virtuel de ressources
L'utilisateur a accès à tous les noeuds du pool virtuel	L'utilisateur a accès au pool pas aux noeuds individuels
L'accès au noeud implique l'accès aux ressources du noeud	L'utilisateur doit avoir une permission (*credential*) pour utiliser une ressource
L'utilisateur a connaissance des caractéristiques du noeud	L'utilisateur ignore l'états des ressources
Les éléments du pool sont plus au moins statiques	Les éléments du pool sont dynamiques

Tab. 1. *Comparaison entre les caractéristiques des machines virtuelles d'un Grid et d'un système distribué.*

Grid la disponibilité des ressources varie en fonction de la charge du système ou de l'ajout, ainsi que la suppression d'une ressource par son propriétaire. En se basant sur le tableau 1 des différentes hypothèses initiales sur le partage des ressources, nous allons décrire les différences entre les modèles ASM des deux systèmes.

D'abord, spécifions les univers des processus PROCESSUS, des tâches TACHE, des utilisateurs $u \in$ UTILISATEUR, des ressources $r \in$ RESSOURCE et des noeuds NOEUDS. Pour les fonctions entre ces univers, définissons les relations suivantes : Un processus a un utilisateur *utilisateur* : PROCESSUS $\rightarrow$ UTILISATEUR. Un processus exécute une tâche *tache* : PROCESSUS $\rightarrow$ TACHE et une tâche peut seulement être exécutée si elle est installée sur un noeud *installé* : TACHE $\rightarrow$ NOEUDS. Un processus requière certaines ressources *demande* : PROCESSUS $\times$ RESSOURCE $\rightarrow$ {*true*,*false*} ou utilise déjà des ressources *utilise* : PROCESSUS $\times$ RESSOURCE $\rightarrow$ {*true*,*false*}. Pour exécuter un processus, il doit être affecté à un noeud *affecté* : PROCESSUS $\rightarrow$ NOEUD. Une ressource fait aussi partie ou non d'un noeud *faitPartie* : RESSOURCE $\times$ NOEUD $\rightarrow$ {*true*,*false*}. Un utilisateur peut se loguer sur certains noeuds *peutLoguer* : UTILISATEUR $\times$ NOEUD $\rightarrow$ {*true*,*false*}.

4.1 Système distribué

Pour un système distribué, décrivons l'affectation des processus aux noeuds du pool virtuel et la réservation des ressources par les processus. D'abord, fixons l'état initial de notre système : Soit $p_1, \ldots, p_k \in$ PROCESSUS l'ensemble des processus initiaux de notre système. Un processus appartient à un utilisateur $\forall p_i \ 1 \leq i \leq k$: $utilisateur(p_i) = u \in$ UTILISATEUR et est assigné à une tâche $\forall p_i \ 1 \leq i \leq k$: $tache(p_i) \neq undef$. Les utilisateurs peuvent se loguer aux noeuds du pool virtuel $\forall u \in$ UTILISATEUR : $\exists n_1, \ldots, n_l \in$ NOEUDS : $peutLoguer(u, n_i) = true$ avec $1 \leq i \leq l$. Initialement, les processus ne sont pas encore affectés à un noeud : $\forall p_i \ 1 \leq i \leq k$: $affecté(p_i) = undef$. De plus, chaque processus a besoin d'au moins une ressource $\forall p_i \ 1 \leq i \leq k$: $\exists r \in$ RESSOURCE :

$demande(p_i, r) = true$ et aucune ressource n'est encore accordée aux processus $\forall p_i\ 1 \leq i \leq k : \forall r \in \text{RESSOURCE} : utilise(p_i, r) = false$.

Comme notre système distribué se base sur un pool de noeuds (tableau 1), notre première règle effectue l'affectation des processus aux noeuds. Les conditions minimales pour affecter un processus sont que l'utilisateur du processus puissent se loguer sur le noeud et que la tâche assignée au processus soit installée sur le noeuds.

```
if affecté(p) = undef then
   choose n in NOEUD satisfying peutLoguer(utilisateur(p), n)
                                & installé(tache(p), n)
         affecté(p) := n                                    (règle 1)
   endchoose
```

L'instruction `choose` est une abstraction d'un mécanisme de sélection d'un noeud. A ce niveau, le modèle ne détermine pas la manière de sélection d'un noeud, tous les noeuds qui vérifient la condition peuvent être choisis. Remarquons que nous pourrions par exemple introduire à ce niveau des mécanismes d'équilibrage de charge qui choisiraient alors un noeud approprié.

Avant de pouvoir exécuter un processus, on doit encore lui accorder les ressources nécessaires. Selon nos hypothèses de départ, un utilisateur qui peut se loguer sur un noeud peut aussi utiliser les ressources du noeud. Il suffit donc de vérifier si une ressource est présente sur un noeud pour l'accorder à un processus. A ce niveau d'abstraction et avec l'hypothèse que chaque ressource devienne disponible dans un temps fini, le modèle ignore l'utilisation des ressources en concurrence ou en exclusivité.

```
if (∃r ∈ RESSOURCE) : demande(p, r) =true & faitPartie(r,affecté(p))
   then utilise(p, r) :=true                                (règle 2)
        demande(p, r) :=false
```

Remarquons que l'instruction `choose` de la règle 1 devait choisir un noeud spécifique dans le cas où le processus aurait besoin d'une ressource particulière non-disponible sur chaque noeud du pool virtuel. Après que tous les ressources soient accordées à un processus, il peut commencer son exécution. Dans l'exécution, un processus peut demander d'autres ressources, lancer une copie de lui-même dans un processus fils, communiquer avec d'autres processus et finalement se terminer. Vu que les règles ASM relative au changement d'état du système distribué à la suite des exécutions des processus n'influencent pas la différence sémantique entre un système distribué et un *Grid*, elles sont négligées dans le cadre de cet article[3].

4.2 Grid

Pour le modèle de *Grid* nous remplaçons le pool de noeuds de calcul par un pool de ressources. Comme un utilisateur ne peut plus se loguer sur un noeud spécifique et utiliser les ressources de ce noeud nous devons ajouter une fonction à la signature de notre modèle qui exprime le fait qu'un utilisateur puisse utiliser une ressource distante *peutUtiliser* : UTILISATEUR × RESSOURCE →

[3] Ces règles se trouvent explicitement dans l'article de base [1].

{*true*,*false*}. Ainsi, on complète notre état initial du modèle qui décrit maintenant un pool de ressources et des utilisateurs qui ont une autorisation d'utilisation chez les propriétaires des ressources : $\forall u \in$ UTILISATEUR , $\exists r_1, \ldots, r_m \in$ RESSOURCES : $peutUtiliser(u, r_i) = true$ avec $1 \leq i \leq m$.

Pour nos deux règles du système distribué, on constate que pour un *Grid*, on recherche d'abord une ressource dans le pool des ressources et puis, on affecte le processus au noeud spécifique si la ressource demandée est la puissance de calcul ($ressource_0$). A première vue, nous pouvons donc regrouper dans le cas du *Grid* les règles 1 et 2 :

if ($\exists r \in$ RESSOURCE) : $demande(p, r) = true$ & $peutUtiliser(utilisateur(p), r)$
  then if $type(r) = ressource_0$ **then** $affecté(p) := location(r)$
    $installé(task(p),\ location(r)) := true$
    endif (règle 3)
    $demande(p, r) := false$
    $utilise(p, r) := true$

Or, cette règle est incomplète selon les conditions du *Grid*. Ainsi, le r de la fonction $demande(p, r)$ est une ressource abstraite qui décrit les caractéristiques de la ressources demandée par le processus p. Dans le système distribué, l'affectation de la ressource abstraite à la ressource physique est implicite car le processus est déjà affecté à un noeud (règle 1) et ce noeud dispose les ressources nécessaires (validées par *faitPartie*). Comme pour le *Grid*, la fonction $utilise(p, r)$ ne peut pas être interprétée sur une ressource abstraite, on introduit un agent $\pi\_affecterRessource$ qui affecte la ressource abstraite à une ressource physique parmi les centaines ressources physiques qui sont compatibles avec les caractéristiques de la ressource abstraite. Pour cela, divisons l'univers RESSOURCE entre ressources abstraites ARESSOURCE et ressources physiques PRESSOURCE. Comme l'utilisateur n'a pas ou peu de connaissance sur l'état des ressources (tableau 1) introduisons la fonction *affecterRessource* : PROCESSUS$\times$ ARESSOURCE $\rightarrow$ PRESSOURCE qui réalise l'affectation des ressources.

if ($\exists ar \in$ ARESSOURCE, $p \in$ PROCESSUS) : $affecterRessource(p, ar) = undef$
    & $demander(p, ar) = true$
  then choose $r \in$ PRESSOURCE **satisfying** $compatible(attr(ar),\ attr(r))$
    affecterRessource$(p, ar) := r$
    endchose ($\pi\_affecterRessource$)

Comme pour la sélection du noeud dans le cas du système distribué, nous pouvons inclure dans l'agent $\pi\_affecterRessource$ pour l'affectation de la ressource abstraite à une ressource physique des conditions supplémentaires qui font partie d'un système d'équilibrage de charge.

Avec l'agent $\pi\_affecterRessource$ notre règle 3 devient :

let $r = affecterRessource(p, ar)$
if ($\exists ar \in$ ARESSOURCE) : $demande(p, ar) = true$ & $peutUtiliser(utilisateur(p), r)$
    & $r \neq undef$
  then if $type(r) = ressource_0$ **then** $affecté(p) := location(r)$
    $installé(task(p),\ location(r)) := true$
    endif
    $demande(p, ar) := false$ (règle 3 bis)
    $utilise(p, r) := true$

Dans cette dernière règle, la fonction $utilise(p, r)$ pose encore problème. En effet, dans le cas du *Grid* la permission d'utiliser une ressource du pool virtuel n'implique pas que l'utilisateur puisse aussi se loguer sur le noeud auquel la ressource appartient. Ainsi, un modèle de *Grid* devrait aussi posséder une affectation de l'utilisateur qui a une permission d'utiliser une ressource (validée par *peutUtiliser*) à un utilisateur local du noeud qui puisse alors exploiter cette ressource pratiquement. A un niveau hautement abstrait, la fonction $utilise(p, r)$ implique que chaque processus peut utiliser chaque ressource distante. A un niveau plus bas où le système d'exploitation accorde la gestion d'une ressource à un processus local, l'abstraction d'utilisateur est alors nécessaire afin d'utiliser à travers un concept d'utilisateur local, le processus local responsable de la ressource. Dans le cadre restrictif de cet article, nous allons nous limiter a cette idée intuitive[4] de l'abstraction des utilisateurs pour le *Grid*.

5 Réflexions sur le modèle ASM du *Grid*

Avec leur modélisation, les auteurs de l'article de base tracent une frontière entre les systèmes distribués et les *Grids*. Ainsi, ils montrent que les *Grids* ne sont pas simplement un successeur des systèmes distribués traditionnels, mais qu'il existe au niveau de la sémantique opérationnelle des différences fondamentales entre les deux systèmes analysés. À partir de quelques hypothèses initiales sur le partage des ressources dans un système distribué et dans un *Grid*, ils identifient l'abstraction des ressources et des utilisateurs comme deux fonctionnalités minimales qui déterminent le mécanisme d'exécution d'un système *Grid*. Comme déjà énoncé au-dessus, l'OGSA dispose avec ces services d'une sorte d'abstraction de ressources et d'utilisateur et satisfait donc au modèle ASM présenté ici. Vu le haut niveau d'abstraction du modèle du *Grid*, on peut ainsi comparer avec ce modèle de base différents développements et applications qui affirment être applicable sur *Grid*. Ainsi, l'objectif primaire qui consiste à pouvoir déterminer et définir un système *Grid* est atteint par la modélisation.

Par la suite, les auteurs du modèle prétendent, en suivant les avantages du formalisme ASM, pouvoir modéliser les détails du système en modélisant des modèles plus spécifiques. Par exemple, la fonction *peutUtilser* apparaît comme une fonction *oracle* dans le modèle et peut être spécifiée a un niveau plus bas pour modéliser d'autres aspects d'un *Grid*. Ainsi, la modélisation en ASM de certaines parties et aspects est intéressante pour la compréhension du système. Par une démarche de plusieurs étapes de spécialisation, on pourrait par la suite spécifier le système jusqu'au niveau concret. D'un point vue analytique, cette modélisation bas niveau du système serait intéressante pour raisonner sur le fonctionnement du système concret. Néanmoins, l'expressivité du modèle résultant pourrait s'avérer d'être un frein à la facilité du raisonnement pour les analystes non-familiarisés avec les expressions du langage mathématique. Par exemple, pour modéliser le parallélisme, ASM propose la notion d'agent mais il manque

[4] De nouveau, le modèle formelle en règles ASM pour l'abstraction des utilisateurs se trouve dans l'article de base [1].

des concepts pour modéliser l'accès mutuellement exclusif à une ressource critique. Aussi, la traduction du modèle pour l'implémentation en paradigmes de programmation non-fonctionnels réserve entres autres des problèmes d'équivalence sémantique.

Comme déjà affirmé, notre modèle haut niveau du *Grid* spécifie bien les caractéristiques fonctionnelles d'un tel système. Néanmoins, nous sommes curieux de voir si un tel modèle mathématique trouvera de l'application dans l'industrie. Semblable au rôle d'un protocole IP/TCP dans les définitions de l'internet, les spécifications des protocoles et interfaces de l'OGSA semblent devenir plutôt le standard dans le monde du *Grid*.

6 Conclusion

En se basant sur quelques hypothèses pour le partage des ressources, la modélisation discutée et montrée dans le formalisme ASM clarifie bien le mode de fonctionnement d'un *Grid*. Notamment, un système *Grid* doit réaliser l'abstraction des ressources et des utilisateurs. Au contraire, des projets comme OGSA où par exemple, l'abstraction des ressources est *implicite* dans une approche orienté-service, l'avantage de notre méthode formelle est de définir précisément et de mettre en avance tous ces aspects qui caractérisent un système comme *Grid*. Ainsi, notre modèle abstrait et minimal du *Grid* clarifie la compréhension du *Grid* et permet ainsi une réflexion approfondie et guidée sur les problèmes dans l'informatique distribuée actuelle.

Bibliographie

[1] Zsolt Németh et Vaidy Sunderam, *Characterizing Grids : Attributes, Defintions and Formalisms*, Journal of Grid Computing, 2003.

[2] Grid Today, `http://www.gridtoday.com`.

[3] Grid Forum, `http://www.gridforum.org`.

[4] Miguel L. Bote-Lorenzo, Yannis A. Dimitriadis et Eduardo Gómez-Sánchez, *Grid Characteristics and Uses : a Grid Definition*, University of Valladolid, Espagne, 2003.

[5] Ian Foster, *What Is a Grid ? a Three Point Checklist*, Grid Today, Num. 6, 2002.

[6] Andrew Grimshaw, *What Is a Grid ?*, Grid Today, Num. 26, 2002.

[7] Ian Foster, Carl Kesselman, Jeffrey M. Nick et Steven Tuecke, *The Physiology of the Grid : An Open Grid Servies Architecture for Distributed Systems Integration*, 2002.

[8] ASM, `http://www.cecs.umich.edu/gasm/`.

[9] Yuri Gurevich, *Evovlving Algebras 1993 : Lipari Guide*, Specification and Validation Methods, Oxford University Press, p. 9-37, 1994.

[10] xASM : un logiciel open source pour le langage ASM, `http://www.xasm.org`.