Towards Formal Interaction-Based Models of Grid Computing Infrastructures

Carlos Alberto Ramírez Restrepo
EISC – Universidad del Valle, Cali, Colombia

Jorge A. Pérez
CITI/DI – FCT - Universidade Nova de Lisboa, Portugal

Jesús Aranda
EISC – Universidad del Valle, Cali, Colombia

Juan Francisco Díaz-Frias
EISC – Universidad del Valle, Cali, Colombia

Grid computing (GC) systems are large-scale virtual machines, built upon a massive pool of resources (processing time, storage, software) that often span multiple distributed *domains*. Concurrent users interact with the grid by adding new tasks; the grid is expected to assign resources to tasks in a fair, trustworthy way. These distinctive features of GC systems make their specification and verification a challenging issue. Although prior works have proposed formal approaches to the specification of GC systems, a precise account of the *interaction model* which underlies resource sharing has not been yet proposed. In this paper, we describe ongoing work aimed at filling in this gap. Our approach relies on *(higher-order) process calculi*: these core languages for concurrency offer a compositional framework in which GC systems can be precisely described and potentially reasoned about.

1 Introduction

Context. Grid computing (GC in the following) systems comprise a large pool of computational resources, which are made available by multiple institutions (*administrative domains*) to users wishing to execute tasks that would be hard (or even impossible) to perform in a single administrative domain. This is in sharp contrast with usual distributed systems, in which each resource is owned and controlled by a single institution. That is, while in distributed systems there is a clear correspondence between system users and valid resource users, in GC systems an analogous correspondence is less explicit, as resources may belong to multiple administrative domains. Moreover, a grid user may not correspond to an actual user in the administrative domains. Yet another point of contrast concerns transparency and security requirements: while in conventional distributed systems users typically know a priori the resources that they need for executing their tasks, grid users may execute tasks without being aware of the internal structure of the system. GC systems differ also from emerging cloud computing platforms, which offer economies of scale for exploiting virtually unlimited resources, based on the Software as a Service (SaaS) paradigm. In fact, differently from clouds, GC systems aim at executing computationally intensive tasks, subject to constraints on resource availability/access. Other notable differences between clouds and grids concern failure management, resource ownership, and infrastructure transparency [5].

This Work. Here we are concerned with principled approaches to the correct design and construction of GC systems. As discussed above, a critical aspect is that of appropriately assigning resources to a potentially huge number of user tasks running concurrently. Given the scale, complexity, and peculiarities of GC systems, this is a challenging issue from several perspectives. In this paper, we describe ongoing work aimed at tackling this issue from the perspective of formal models of computation based on *communication*. More precisely, we explore a *process calculi* approach: based on a small set of operators —typically, atomic interaction, sequencing, parallel composition, and scoping— process calculi such as CCS [11] and the π-calculus [12] have been developed within the concurrency theory community as basic models for communicating systems. As process calculi are *compositional*, they have proved useful for developing reasoning techniques over specifications (e.g. behavioral equivalences and type systems) and for investigating new programming abstractions based on communication. These features make process calculi an attractive basis for the formal specification and verification of GC systems.

In particular, in this paper we rely on *higher-order process calculi*, i.e., calculi in which processes (more generally, values containing processes) can be communicated. This is in contrast to *first-order* calculi such as the π-calculus, in which only basic values and/or communication channels can be exchanged. Higher-order process calculi can be seen as concurrent variants of the λ-calculus. In fact, the reduction rule for communication in these formalisms is reminiscent of well-known β-reduction in functional calculi. In the grid setting, higher-order (or process passing) concurrency naturally models the fact that user tasks —typically, arbitrarily complex descriptions of computational behaviors— need to be exchanged among different grid components in order to achieve their execution. In particular, we rely on the higher-order π-calculus (HOπ) [15], a core language which enhances the name passing abilities of the π-calculus with process passing. HOπ specifications can represent forms of code mobility, therefore allowing for flexibility in descriptions of concurrent communications. Moreover, useful proof techniques based on behavioral equivalences are well-understood for (variants of) HOπ (see, for instance, [16]).

The main contribution of this work is a formal model of GC infrastructures, with a focus on the resource assignment facility that is central to them. Our model distills the main features of GC systems, as informally discussed in the literature (see, e.g., [6]) and as identified in our own exchanges with GC experts. The model is divided into *static* and *dynamic* components. The static component, defined in first-order logic, formalizes the essential pre-conditions and invariants that should hold for the different grid subsystems. Using HOπ processes, the dynamic component captures the (concurrent) execution sequences associated to potentially many users interacting simultaneously with the grid. These components are intended to be complementary: building upon the relations defined by the static component, the dynamic component accounts for the main agents present in real GC infrastructures, such as users, tasks, administative domains, virtual organizations, and resources. Our model also considers user and resource proxies, which facilitate user interaction with the GC system and resource management (see Sect. 2).

While simple, our formal model already provides a good basis for obtaining more detailed descriptions of GC systems and for reasoning about their correctness properties. Examples of such properties are authentication and authorization guarantees: they are intended to ensure that users only access and use the administrative domains and resources for which they can prove their identity/permissions. An extension of our current model with suitable cryptographic elements (using, e.g., the higher-order language in [9]) would be a step in this direction. Another relevant property concerns the balanced assignment of administrative domains and their use of resources. To this end, process specifications of different scheduling and assignment policies may be necessary —this issue is largely orthogonal to the model given here. Other properties of interest for GC modelers involve task termination and resource delivery aspects. By adapting known results on termination and reachability properties for calculi such as CCS [3] and for variants of HOπ [10, 8], our model could offer an alternative for investigating such properties.

Related Work. We believe that our work improves on previous attempts for grid formalization. The π-calculus has been used in [19, 21, 20] for analyzing the specific aspects of grid services composition and workflow. These approaches only model the GC components related to grid services such as resources and tasks; other aspects of the GC dynamics are not considered. In contrast, our model adopts a more comprehensive view of GC systems, including, e.g., key interaction patterns related to user intervention, and the rôle of user and resource proxies in resource assignment and task execution. In [13], Abstract State Machines (ASM) are used to give a declarative characterization of GC systems; this characterization formally describes some of the main attributes that a GC system should support. The GC elements are modeled as universes (sets); their behavior is represented using rules over universes. The only grid agents considered in [13] are tasks (there called *processes*); there are also user and resource mapping agents. Each agent executes the rules over the defined universes. In contrast to our work, in the model of [13] concurrent interactions among GC components are not explicitly represented; also, such a model does not consider the key concept of virtual organizations and the rôle of user and resource proxies. Finally, in [1, 4], high-level and colored Petri nets were used to analyze grid architectures and grid workflows. A 3-layer grid architecture and the interaction between GC components in these layers is modeled. However, these approaches do not consider virtual organizations, administrative domains, and security requirements —all of these being central elements in resource assignment.

Organization. The rest of this paper is organized as follows. Sect. 2 briefly recalls the main features of GC systems. In Sect. 3 we present the syntax and semantics of the higher-order π-calculus. Sect. 4 gives a brief description of our GC formalization, and Sect. 5 illustrates it via a small example. Finally, future work is discussed in Sect. 6. A full description of our formal process model is available in [14].

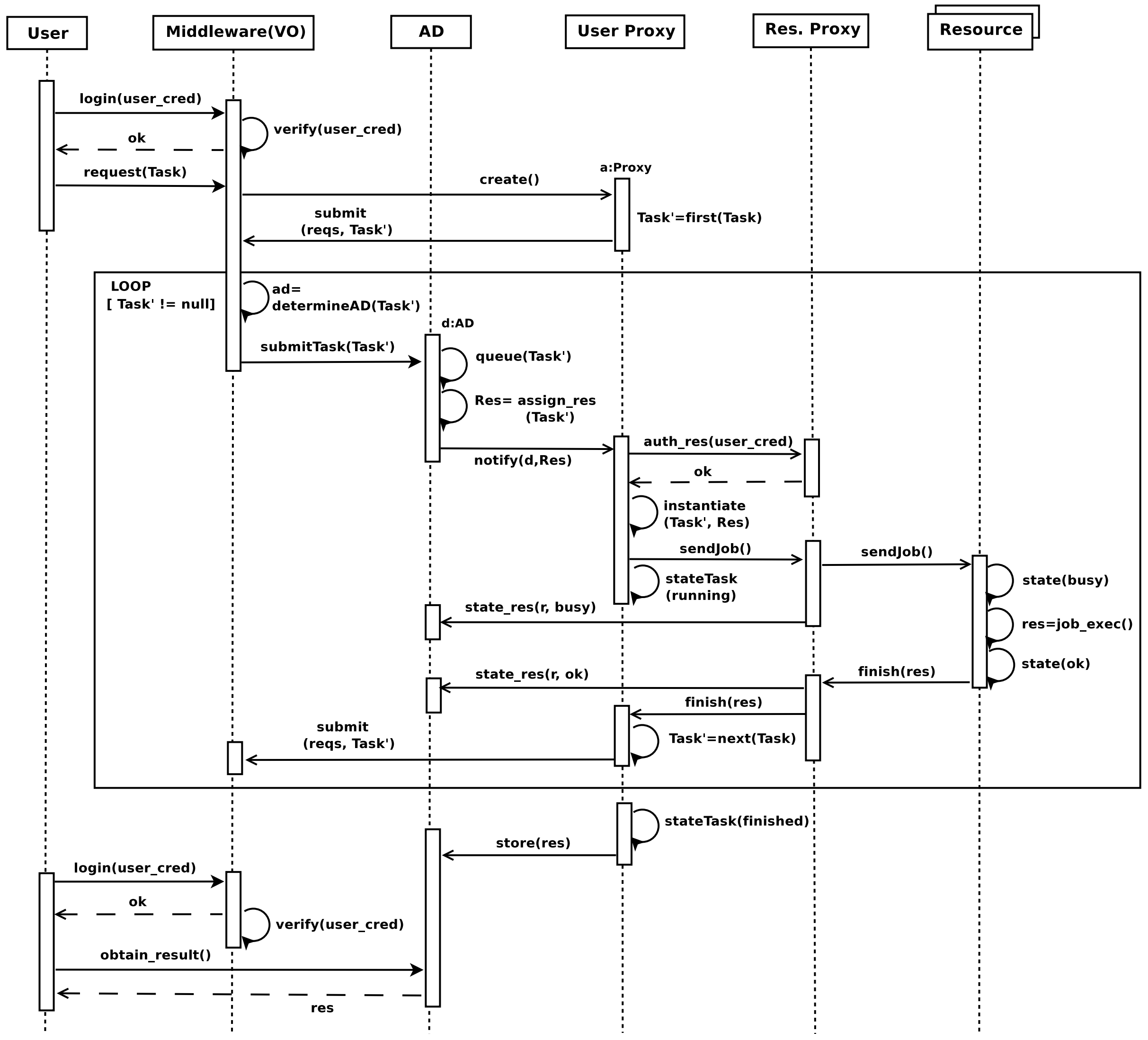
2 Grid Computing: A Brief Overview

Grid computing broadly refers to the coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. GC systems often require interoperability features and support for heterogeneous environments. Other typical requirements are decentralized control, security, access transparency, scalability, availability, and reconfigurability [6]. Sharing in GC systems not only refers to data and information but also to direct access to all kinds of resources which may be required for executing complex tasks (computing power, storage, software applications, data). Each *administrative domain* (AD in the following) establishes what resources are shared and their access and usage policies. A *virtual organization* (VO in the following) is a set of ADs defined by such policies. The participants in a VO share resources in a controlled way in order to cooperate in executing a specific task. VOs vary in their purpose, scope, size, duration, structure, community, and sociology [6].

In GC systems, users can transparently share or access resources—they do not need to know (or be aware of) what resources they are using, where such resources are physically located or that they may have previously failed and recovered. This transparency is achieved by the so-called *grid middleware*. This is a software layer that (i) implements the *protocols* and *services* that enable the seamless sharing of heterogeneous resources, and (ii) provides key functionalities for enabling task execution and VOs establishment [18]. In this way, the middleware allows users to access resources while satisfying security policies such as authentication, authorization, delegation, and single sign-on. To this end, the grid middleware includes user and resource *proxies* [7]. While a user proxy is an entity that is given permission to act on behalf of a user for a fixed period of time, a resource proxy serves as interface between the middleware and a resource, thus simplifying (i) the authentication between user proxy and the resource and (ii) the mapping between grid users and the local users which are valid in the resource.

Grid Resource Assignment Protocol. As our interest is in an interaction-based approach to GC systems, below we present a protocol which describes the interaction sequence among the main grid components (users, ADs, VOs, resources, proxies). The protocol is described as a sequence diagram in Fig. 1;

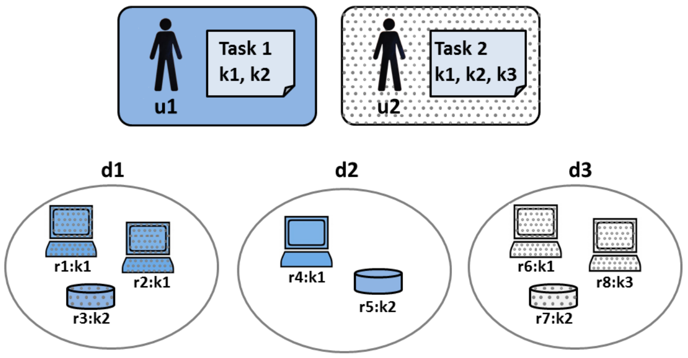
Figure 1 The Grid Interaction Protocol as a Sequence Diagram



it formalizes requirements and mechanisms which have been described in the literature only informally. The formal model given in Sect. 4 is then intended to give a precise account of this protocol.

1. A user sends its credentials to a grid node in order to authenticate. In the figure, this step is represented by the message $login(user\_cred)$ from User to VO.
2. If the authentication is successful then the user is granted to access the grid. Otherwise, the user must revise its credentials. For simplicity, the figure shows only the case in which authentication is successful; this step is represented by message *ok* from VO to User.
3. The authenticated user sends a proxy creation request, and a task with its requirements to the grid node. The task may be a complex object; in particular, it may be structured in terms of subtasks which

Figure 2 A GC Scenario: Two users (u_1, u_2), two VOs (v_1 – blue, v_2 – dotted), three ADs (d_1, d_2, d_3)



follow some process logic. In the figure, these steps are represented by messages $userproxy\_creation()$ (from User to VO), $create()$ (from VO to User Proxy), and $request(Task)$ (from User to VO).

4. The user proxy sends to the grid node the requirements of each subtask. In the figure, this step is represented by the message $submit(reqs, Task')$ from User Proxy to VO.

5. The node selects an AD in the VO with available resources to satisfy the subtask requirements. This subtask is assigned and sent to the selected AD. In Fig. 1, this is represented by messages $determineAd(Task')$ (inside VO), $submitTask(Task')$ (from VO to AD), and $queue(Task')$ (inside AD).

6. The AD assigns appropriate resources for this subtask according to some scheduling strategy. In the figure, this step is represented by the message $assignRes(Task')$ inside AD.

7. The user proxy authenticates into the resource proxies of assigned resources. If authentication is successful then the subtask is executed. Otherwise, the subtask is sent back to the grid node. In the figure, these steps are represented by messages $auth\_res(user\_cred)$ (from User Proxy to Resource Proxy), ok (from Resource Proxy to User Proxy), $sendJob()$ (from Resource Proxy to Resource), and $job\_exec()$ (inside Resource).

8. When the subtask has finished (message $finish(res)$, from Resource to its Resource Proxy), it is detected if there are more subtasks (condition $Task' \neq null$ in the loop). If yes then the result of the previous subtask is transmitted to the next subtask and the previous subprotocol is executed again (message $submit(reqs, Task')$). Otherwise, if the just executed subtask is the last one, then the result is stored and the protocol finishes (message $store(res)$ from User Proxy to AD).

A Representative GC Scenario. We now describe a small, representative example of a GC system. Depicted in Fig. 2, our scenario draws inspiration from the one given in [6]. It contains three ADs (denoted d_1, d_2, and d_3) and two VOs (denoted v_1 and v_2); in the figure, they are depicted as ovals and rectangles, respectively. VO v_1 (blue background) groups participants in an aerospace design consortium and v_2 (dotted background) links participants for sharing spare computing cycles. AD d_1 is member of both v_1 and v_2. Also, ADs d_1 and d_2 participate in v_1 and AD d_3 participates in v_2. We also consider users u_1 and u_2: while u_1 belongs to v_1, user u_2 belongs to u_2. Both u_1 and u_2 have a task to execute in the grid, denoted Task1 and Task2 in the figure, respectively. To perform, Task1 requires one resource of type (or descriptor) k_1 and one resource of type k_2. Similarly, Task2 requires three resources, distinguished by types k_1, k_2, and k_3. Resources are located in appropriate ADs: AD d_1 owns three resources: r_1 (type

k_1), r_2 (type k_1), and r_3 (type k_2); AD d_2 owns two resources: r_4 (type k_1) and r_5 (type k_2); and AD d_3 owns three resources: r_6 (type k_1), r_7 (type k_2), and r_8 (type k_8). While resources of d_1 are shared by v_1 and v_2, resources of d_2 are available only to v_1, and resources of d_3 are available only to v_2.

3 The Process Model: Syntax and Semantics

This section briefly presents the syntax and semantics of the higher-order π calculus, HOπ. Our presentation closely follows [15]. In HOπ, both names (communication channels) and processes may be passed around by synchronization on names; communication can be thus loosely assimilated to β-reduction in the λ-calculus. We assume a set of names/channels ranged over $x, y, z, \ldots$ and a set of process variables ranged over $X, Y, Z, \ldots$. We write $\widetilde{o}$ to denote a finite tuple of elements $o_1, \ldots, o_k$.

Definition 3.1. *The language of HOπ* processes *is given by the following syntax:*

$$\begin{array}{lcl} \alpha & ::= & x(\widetilde{U}) \;\big|\; \overline{x}\langle\widetilde{K}\rangle \\ P & ::= & \sum_{i\in I}\alpha_i.P_i \;\big|\; P_1 \mid P_2 \;\big|\; (\nu\, x)P \;\big|\; \mathtt{if}\ [x=y]\ \mathtt{then}\ P_1\ \mathtt{else}\ P_2 \;\big|\; \mathsf{D}\langle\widetilde{K}\rangle \;\big|\; X\langle\widetilde{K}\rangle \end{array}$$

We have two *prefixes*, ranged over $\alpha, \alpha', \ldots$. An input prefix $x(\widetilde{U})$ (resp. output prefix $\overline{x}\langle\widetilde{K}\rangle$) denotes an atomic input action (resp. output action) on a name x. Above, $\widetilde{K}$ and $\widetilde{U}$ denote tuples of names and processes, and of names and variables, respectively. Process $\sum_{i\in I}\alpha_i.P_i$ represents the *non-deterministic choice* among prefixed processes $\alpha_i.P_i$. The operational semantics ensures that only one of them will be executed, discarding the rest. When $I = \emptyset$ we write $\mathbf{0}$; when $I = |2|$ we write $\alpha_1.P_1 + \alpha_2.P_2$. Also, we simply write α to refer to process $\alpha.\mathbf{0}$. Process $P_1 \mid P_2$ stands for the *parallel composition* of processes P_1 and P_2. We write $\prod_{j\in J} P_j$ as a shorthand notation for process $P_1 \mid \ldots \mid P_{|J|}$. Process $(\nu x)P$ declares the name x private to process P. That is, the scope of x is P; this scope may be enlarged by communication to other processes (*scope extrusion*). The conditional $\mathtt{if}\ [x=y]\ \mathtt{then}\ P_1\ \mathtt{else}\ P_2$ is based on equality of names x and y: if $x=y$ then the process continues as P_1; otherwise it continues as P_2. By taking inputs and restriction as binders, notions of free and bound names/variables arise as expected. We identify processes up to consistent renaming of bound names/variables, writing $\equiv_\alpha$ for this congruence.

One way of specifying infinite process behavior is via *parametric definitions*. Notation $\mathsf{D}\langle\tilde{K}\rangle$ denotes the application of a constant identifier D with parameters $\widetilde{K}$. We assume each D has a unique definition $\mathsf{D}(\tilde{U}) \overset{def}{=} P$, where $\widetilde{U}$ is composed of all free names or variables in P, i.e. names or variable which occur out the scope of any binding. Then, $X\langle\tilde{K}\rangle$ denotes the application of parameters $\widetilde{K}$ to process variable X.

We endow our process language with a *reduction semantics*. Intuitively, a reduction $P \longrightarrow Q$ denotes a single evolution step from process P to Q, without interaction from its surrounding environment.

Definition 3.2. Reduction, *$P \longrightarrow Q$, is the binary relation on processes defined by the rules in Fig. 3.*

As usual, we write $\Longrightarrow$ to denote the reflexive, transitive closure of $\longrightarrow$. The rules in Fig. 3 formalize process communication and reduction under parallel composition and restriction. In rule (COM), notation $P\{\widetilde{K}/\widetilde{U}\}$ stands for process P in which all free occurrences of names/variables in $\widetilde{U}$ have been substituted by names/processes in $\widetilde{K}$. We assume arity in communications is consistent, i.e., the length of $\widetilde{U}$ must be equal to the length of $\widetilde{K}$, with one-to-one correspondence among elements of both tuples. By means of rule (STR), reduction is closed under a *structural congruence* relation, written $\equiv$, which is used to promote process interactions. It is defined as follows:

Figure 3 Reduction semantics for HOπ

$$\textsf{(COM)}\quad (\ldots + x(\widetilde{U}).P) \mid (\ldots + x\langle\widetilde{K}\rangle.Q) \longrightarrow P\{\widetilde{K}/\widetilde{U}\} \mid Q$$

$$\textsf{(PAR)}\ \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \textsf{(RES)}\ \frac{P \longrightarrow Q}{(\nu x)P \longrightarrow (\nu x)Q} \qquad \textsf{(STR)}\ \frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q}$$

Definition 3.3. *Structural congruence, written $P \equiv Q$, is the smallest process congruence such that*

$$P \mid \mathbf{0} \equiv P \qquad P \equiv_\alpha Q \Rightarrow P \equiv Q \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad (\nu x)\mathbf{0} \equiv \mathbf{0}$$
$$x \notin fn(P) \Rightarrow P \mid (\nu x)Q \equiv (\nu x)(P \mid Q) \qquad (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$$
$$\texttt{if } [x=y] \texttt{ then } P_1 \texttt{ else } P_2 \ \equiv\ P_1\ (\textit{if } x=y) \qquad \texttt{if } [x=y] \texttt{ then } P_1 \texttt{ else } P_2 \ \equiv\ P_2\ (\textit{if } x \neq y)$$
$$D(\widetilde{U}) \overset{def}{=} P \ \Rightarrow\ D\langle\widetilde{K}\rangle \ \equiv\ P\{\widetilde{K}/\widetilde{U}\} \qquad \sum_{i\in I} \alpha_i.P_i \ \equiv\ \sum_{j\in J} \alpha_j.P_j\ (\textit{if } J \textit{ is a permutation of } I)$$

4 A Formal Model of Grid Interaction

We now give an overview of our formal model of GC systems; a full description can be found in [14]. The model is intended as a formal counterpart of the informal interaction protocol given in Sect. 2. As already discussed, the model is divided into static and dynamic components. While the former is given in terms of invariants (first-order logic formulas), the latter is specified using HOπ processes. The two components play complementary rôles in our model. On the one hand, the invariants and conditions in the static part are used to:

– Define the actors in the system (e.g. users, administrative domains, resources, tasks) and useful relationships between them;

– Describe the initial configuration of the system;

– Define well-formedness conditions for the processes of the dynamic part.

On the other hand, the dynamic part focuses on representing:

– How the grid assigns resources to each task;

– The start of task execution;

– The state of tasks and their assigned resources.

It is worth highlighting that processes of the dynamic component cannot add new tasks, resources or users. Although these capabilities are present in some real grid systems, in the current development we focus on systems in which those elements cannot be added at runtime.

Static Component: Base Sets and Invariants

In order to formalize the key components of GC systems, we first relate such components to base reference sets. Then, we state associated invariant properties by defining static predicates over elements of such sets. Table 4 summarizes our notation for these base sets. The intuitive meaning of most of them

Figure 4 Static model: Base sets for GC components

GC Component	**Base set**	**GC Component**	**Base set**
Users	$u \in U$	VOs	$v \in V$
ADs	$d \in D$	Tasks	$\mathtt{T} \in T$
Resources	$r \in R$	User Tasks	$\mathtt{S} \in S$
Nodes	$n \in N$	User Proxies	$a \in A$
Resource Proxies	$x \in X$	Resource Descriptors	$k \in K$
Logs	$l \in L$		

should be clear from the description given in Sect. 2. We consider that each VO is associated to a group of access points (nodes) which are contained in the base set N. Observe that we distinguish between *task definitions* (which belong to base set T) and *task instances*, which are submitted by users (and belong to base set S). We assume that task definitions are built using the next grammar:

$$\mathtt{T} ::= \mathtt{J}\langle k_1, \ldots, k_m\rangle \;\big|\; \mathtt{T}.\mathtt{T} \;\big|\; \mathtt{T} \parallel \mathtt{T} \;\big|\; \mathtt{T} \oplus \mathtt{T} \;\big|\; \mathtt{end}$$

Above, $\mathtt{J}\langle k_1, \ldots, k_m\rangle$ denotes a *basic task* $\mathtt{J}$ with resources of type $k_1, \ldots, k_m$, respectively ($m \geq 1$). Building upon basic tasks, more complex ones can be defined, using sequential and parallel composition (denoted $\mathtt{T}.\mathtt{T}$ and $\mathtt{T} \parallel \mathtt{T}$, respectively) and non-deterministic choice ($\mathtt{T} \oplus \mathtt{T}$). We also assume a termination task, denoted `end`. As discussed above, we assume that each user is associated to a single task. This is not a limitation, for tasks may involve several subtasks in parallel and sequential composition.

As for the invariants, based on informal descriptions in the literature [6], we have identified the elements that we consider essential to GC systems. Using first-order logic, we formalize such elements in terms of predicates over the elements of the reference sets. Some of such invariants are the following:

– *Each user is member of exactly one VO.* Using predicate $member(u,v)$, which holds if user $u \in U$ is member of VO $v \in V$, we may state this invariant as:

$$\forall_{u \in U}, \exists_{v \in V}.\ member(u,v) \ \wedge\ \forall_{u \in U,\ v,v' \in V}.\ (member(u,v) \ \wedge\ member(u,v') \ \rightarrow\ v = v')$$

– *Each user is associated to exactly one task to be executed in the GC system.* Using predicate $task(u,\mathtt{S})$, which holds if user $u \in U$ is the owner of task $\mathtt{S} \in S$, we may state this invariant as:

$$\forall_{u \in U}, \exists_{\mathtt{S} \in S}.\ task(u,\mathtt{S}) \ \wedge\ \forall_{u \in U,\ \mathtt{S},\mathtt{S}' \in S}.\ (task(u,\mathtt{S}) \ \wedge\ task(u,\mathtt{S}') \ \rightarrow\ \mathtt{S} = \mathtt{S}') \ \wedge$$
$$\forall_{u,u' \in U,\ \mathtt{S} \in S}.\ (task(u,\mathtt{S}) \ \wedge\ task(u',\mathtt{S}) \ \rightarrow\ u = u').$$

– *Each resource belongs to exactly one AD.* Using predicate $belongsTo(r,d)$, which holds if resource $r \in R$ belongs to AD $d \in D$, we may express this invariant as:

$$\forall_{r \in R}, \exists_{d \in D}.\ belongsTo(r,d) \ \wedge\ \forall_{r \in R,\ d,d' \in D}.\ (belongsTo(r,d) \ \wedge\ belongsTo(r,d') \ \rightarrow\ d = d')$$

– *Every AD can participate in one or more VOs.* Using predicate $participate(d,v)$, which holds if AD $d \in D$ participates into VO $v \in V$, we may state this invariant as: $\forall_{d \in D}, \exists_{v \in V}.\ participate(d,v)$.

Additional invariants concern access points (nodes), resource descriptors, task states, resource states, and task logs; they are given in terms of appropriate base sets, and are omitted here for the sake of space.

Figure 5 Dynamic model: Correspondence among GC components and processes (full details in [14])

GC Component	HOπ **Process**	**Intuitive Description**
Grid system	$\mathsf{Grid}^{\omega,\mu}_{\delta,\eta}$	Represents the whole GC system
User ($u \in U$)	$[\![u,\mathtt{S}]\!]^{c,y} = \mathsf{User}(\widetilde{c}, [\![\mathtt{S}]\!]^{t,e}, y)$	Models the behavior of u to authenticate and submit its task $\mathtt{S}$
	$\hookrightarrow \mathsf{UsrMonitor}(\widetilde{c}, g, a, y, \mathsf{P})$	Monitors tasks submitted by u
Node ($n \in N$)	$[\![n]\!]^{v,y,\widetilde{d}} = \mathsf{AP}(y, \widetilde{d})$	Models the interaction of n with users to authenticate
	$\hookrightarrow \mathsf{AP\text{-}UsrHandler}(ch_1, ch_2, ce)$	Models the user proxy creation and task submission
	$\hookrightarrow \mathsf{AP\text{-}ProxyHandler}(ce, \widetilde{d})$	Represents the interaction with the user task
VO ($v \in V$)	Composition of instances of $\mathsf{AP}(y, \widetilde{d})$	A collection of nodes
AD ($d \in D$)	$[\![d]\!] = \mathsf{AD}(d)$	Models the AD with its resources, proxy resources and management elements
	$\hookrightarrow \mathsf{AD\text{-}RecReq}(b, d)$	Receives the tasks assigned to the AD and puts them in the queue
	$\hookrightarrow \mathsf{AD\text{-}AsgRes}(b, d, ch)$	Dequeues tasks and assigns appropriate resources to them
	$\hookrightarrow \mathsf{AD\text{-}LRM}(\widetilde{s}, \widetilde{x}, \widetilde{w}, ch, d)$	Supervises the state of resources, and determines the available resources for a task
Resource ($r \in R$)	$[\![r]\!]^{r,q} = \mathsf{AD\text{-}Resource}(r, q)$	Models a resource's behavior when is used by a task
User Proxy ($a \in A$)	$[\![a]\!]^{ce,p,t,g} = \mathsf{AP\text{-}UserProxy}(ce, p, t, g)$	Models the task management, the request of execution of subtasks and the authentication with resource proxies
Res. Proxy ($x \in X$)	$[\![x]\!]^{x,q,r,w} = \mathsf{AD\text{-}ResourceProxy}(x, q, r, w)$	Acts as a mediator between GC components and a resource
Log ($l \in L$)	$\mathsf{AP\text{-}Log}(g_r, g_w, st, \widetilde{z})$	Interacts with GC components to register the changes in the task state and result
Task ($\mathtt{T} \in T$)	$\lceil \mathtt{T} \rceil^{t,e}$ definition	Represents the behavior of a task
User Task ($\mathtt{S} \in S$)	$[\![\mathtt{S}]\!] = \lceil \mathtt{T} \rceil^{t,e}$	Models a task instance corresponding to a user task
Descriptors ($k \in K$)	Names $\mathtt{k}_1, \ldots, \mathtt{k}_\kappa$	Models the different types of resources

Dynamic Component: Model in the HOπ calculus

In addition to specifying the main system components and the valid relations among them, our model should unambiguously describe how the system may evolve as a result of the interaction of its components. In the light of the protocol given in Sect. 2, such interactions may follow intricate patterns and must adhere to basic correctness and trustworthiness criteria. We would like formal mechanisms to ensure that models indeed satisfy such criteria. As we wish to describe GC systems compositionally, precisely specifying the interacting mechanisms and their relationships, first-order logic is not the most appropriate formalism for this task. We then appeal to specifications expressed as HOπ processes: they offer a basis on which interaction features in GC systems can be succinctly represented, and potentially verified using reasoning techniques over interacting processes. We thus extend the static description overviewed above so as to define in HOπ the behavior of GC components and their interactions according to the invariants and predicates of the static representation. Fig. 5 summarizes the correspondence between the elements in the static description and their respective process representation in the dynamic component of the model. In the figure, we use the symbol $\hookrightarrow$ to represent sub-processes which are triggered as part of the execution of a main process. Complete descriptions of the processes mentioned the figure can be found in [14].

Next we briefly describe process representations for some grid components (users, middleware, ADs) mentioned in Fig. 5. We use ω, μ, δ, and η to denote, respectively, the number of VOs, users, ADs, and nodes (access points) in the system. Also, we rely on standard process representations of queues (and associated operations) which can be easily encoded in HOπ via name passing (see, e.g., [17]). It is worth highlighting that the HOπ representations of the GC components are related to the invariants and other elements of the static component of the model. This means that process interactions do not concern arbitrary elements of the base sets; rather, they involve elements which may be subject to invariants. For example, our process representation for users only can interact with the process representation of a node that corresponds to a VO where such a user is member. Interestingly, key elements of the process language (notably, the exchange of fresh channels and scope extrusion) turn out to be useful to enforce the static invariants in the dynamic specification, and to rule out undesirable interferences among components (as in, e.g., two users which concurrently access a VO). This way, sensible correctness/consistency properties are ensured by construction. Establishing a formal correspondence between the static and dynamic components is part of ongoing work (see Sect. 6).

Grid system. A grid system is modeled as the composition of processes representation of users, ADs, and access points. These are denoted $[\![u,\mathtt{S}]\!]^{\widetilde{c},y}$, $[\![d]\!]$, and $[\![n]\!]^{v,y,\widetilde{d}}$, respectively, which are used as intermediate notations for processes $\mathsf{User}(\widetilde{c},[\![\mathtt{S}]\!]^{t,e},y)$, $\mathsf{AD}(d)$, and $\mathsf{AP}(y,\widetilde{d})$, respectively. This structure promotes interaction: while user processes interact with access point processes through private channels $y_1,\ldots,y_\eta$, AD processes communicate with access point processes in private channels $d_1,\ldots,d_\delta$. This way, our process model of a GC system, parametric on ω, μ, δ, and η, is the following:

$$\mathsf{Grid}\,^{\omega,\mu}_{\delta,\eta} \;\stackrel{\text{def}}{=}\; (\nu\, y_{n_1},\ldots,y_{n_\eta})\big(\prod_{i\in I}[\![u_i,\mathtt{S_i}]\!]^{\widetilde{c_i},y_{node(u_i)}} \;\mid\; (\nu\, d_1,\ldots,d_\delta)(\prod_{h\in H}[\![n_h]\!]^{vo(n_h),y_{n_h},\widetilde{d_h}} \;\mid\; \prod_{l\in L}[\![d_l]\!])\big)$$

where $I=\{1,\ldots,\mu\}, H=\{1,\ldots,\eta\}$, and $L=\{1,\ldots,\delta\}$ are index sets over users, access points, and ADs, resp. In process $\mathsf{User}(\widetilde{c},[\![\mathtt{S}]\!]^{t,e},y)$ (defined below), $[\![\mathtt{S}]\!]^{t,e}$ is a process representation of task $\mathtt{T}$ (where $\mathtt{S}$ is a instance of $\mathtt{T}$) that depends on names t and e: while t is used to send subtasks requirements to the appropriate user proxy, e is used to signal task completion. Given $i\in I$, we write $node(u_i)$ to denote the index of the access point for user u_i. Name d in $\mathsf{AD}(d)$ is used for interaction between the AD and access point processes. In $\mathsf{AP}(y,\widetilde{d})$, name y is used to interact with user processes, while $\widetilde{d}$ stands for a tuple with the access channel of the ADs in the VO associated to the access point. We write $vo(n_h)$ to denote the VO associated to node n_h.

Users. The process model for users, denoted $\mathsf{User}(\widetilde{c},[\![\mathtt{S}]\!]^{t,e},y)$, is parametric on a tuple of user credentials $\widetilde{c}$, a task process $[\![\mathtt{S}]\!]^{t,e}$ (explained above), and a name y, which is used to access a grid node (an instance of process $\mathsf{AP}(y,\widetilde{d})$). Process $\mathsf{User}(\widetilde{c},[\![\mathtt{S}]\!]^{t,e},y)$ interacts with node process $\mathsf{AP}\langle y,\widetilde{d}\rangle$ in order to authenticate to the grid, create a user proxy, and submit/monitor her task. More precisely, we have:

$$\begin{aligned}\mathsf{User}(\widetilde{c},[\![\mathtt{S}]\!]^{t,e},y) \;\stackrel{\text{def}}{=}\;& (\nu\, u)(\overline{y}\,\langle\widetilde{c},u\rangle\,.u(ch_1,-,m).\\ &\quad \mathtt{if}\ [m=\mathtt{ok}]\ \mathtt{then}\ \overline{ch_1}.ch_1(a).\overline{ch_1}\,\big\langle[\![\mathtt{S}]\!]^{t,e}\big\rangle\,.ch_1(g).\mathsf{UsrMonitor}\langle\widetilde{c},g,a,y,\mathsf{P}_{\mathtt{S}}\rangle\\ &\quad \mathtt{else}\ \mathbf{0})\end{aligned}$$

Figure 6 Process $\mathsf{AP\text{-}ProxyHandler}(ce,\widetilde{d})$, part of the middleware, interacts with the user proxy process.

$$
\begin{aligned}
\mathsf{AP\text{-}ProxyHandler}(ce,\widetilde{d}) \overset{\text{def}}{=}\ & ce(\widetilde{k},m,a,g).(\mathsf{AP\text{-}ProxyHandler}\langle ce,\widetilde{d}\rangle \ \mid \\
& \qquad (\nu c,b,f)(\prod_{d_i \in \widetilde{d}} \mathsf{AP\text{-}Search}^{\widetilde{k}}\langle c,f,d_i\rangle \ \mid \ \mathsf{AP\text{-}Acc}\langle c,f,b\rangle \ \mid \\
& \qquad\qquad b(d_1,\ldots,d_\sigma).\sum_{j\in 1\ldots\sigma} \overline{d_j}\left\langle \widetilde{k},m,a,g\right\rangle))
\end{aligned}
$$

Above, the first output on y represents an authentication request against a service deployed at $\mathsf{AP}(y,\widetilde{d})$. This service returns name `ok` (resp. `denied`) if the authentication is successful (resp. failed). We write $u(ch_1,-,m)$ to denote a reception of three arguments along u, in which the second one is not relevant. Name ch_1 is a private name communicated by $\mathsf{AP}(y,\widetilde{d})$: this enables the interference-free communication between user process and a subprocess of the grid node process. Also, ch_1 is used for user proxy creation and task submission: proxy creation is requested by an output signal on ch_1; then, a name a (to be used to access the user proxy) is received on ch_1; subsequently, the task can be sent: this is represented by the (higher-order) output prefix $\overline{ch_1}\langle[\![\mathsf{S}]\!]^{t,e}\rangle$. Once the task has been sent, a channel associated to the log of the submitted task is received on ch_1, and process $\mathsf{UsrMonitor}(\widetilde{c},g,a,y,\mathsf{P})$ is launched: it abstracts the user interaction with her access point for monitoring the task just submitted. The last parameter for $\mathsf{UsrMonitor}$, process $\mathsf{P_S}$, specifies the user behavior that is executed upon reception of the final result of her task. Such a process may correspond to, e.g., a query that stores such a result into a remote database.

Middleware. The middleware is represented as the composition of access point processes $\mathsf{AP}(y,\widetilde{d})$. For each VO in the grid, there are some instances of access point processes associated to it. An instance of $\mathsf{AP}(y,\widetilde{d})$ interacts with an instance of $\mathsf{User}(\widetilde{c},[\![\mathsf{S}]\!]^{t,e},y)$ for authentication purposes, user proxy creation, and task submission/monitoring, as just explained. Then, process $\mathsf{AP}(y,\widetilde{d})$ launches a process $\mathsf{AP\text{-}ProxyHandler}(ce,\widetilde{d})$, given in Fig 6, which interacts with the user proxy process.

Process $\mathsf{AP\text{-}ProxyHandler}(ce,\widetilde{d})$ is parametric on (i) name ce, which is used to receive the task requirements from the user proxy process; and (ii) tuple $\widetilde{d}$, which contains the names associated to the ADs of the VO of the access point. Once $\mathsf{AP\text{-}ProxyHandler}(ce,\widetilde{d})$ has received on ce the tuple $\widetilde{k}$ which represents the descriptors of the required grid resources, it selects the appropriate ADs for the requested resources. We abstract this selection by processes $\mathsf{AP\text{-}Search}^K$ and $\mathsf{AP\text{-}Acc}$. Given a tuple/set of resources descriptors K, each instance of process $\mathsf{AP\text{-}Search}^K$ searches among the resources shared by an AD with resources satisfying the requirements in K. Once a suitable AD has been found, $\mathsf{AP\text{-}Search}^K$ sends the access channel of that AD to $\mathsf{AP\text{-}Acc}$, which records all such access channels. Once all instances of $\mathsf{AP\text{-}Search}^K$ have completed the search, $\mathsf{AP\text{-}Acc}$ sends such ADs to process $\mathsf{AP\text{-}ProxyHandler}(ce,\widetilde{d})$ along name b. Then, $\mathsf{AP\text{-}ProxyHandler}(ce,\widetilde{d})$ non-deterministically selects an AD.

Administrative domains. As mentioned above, an AD is represented as process $\mathsf{AD}(d)$, which consists of the parallel composition of processes in charge of receiving, queuing, and attending task execution requests. Also, $\mathsf{AD}(d)$ comprises process models of resources and resource proxies (see below). For the sake of space, we only present the process $\mathsf{AD\text{-}AsgRes}(b,d,ch)$, which is in charge of assigning the appropriate resources for the subtasks assigned to the AD. This process, given in Fig. 7, is parametric

Figure 7 Process $\mathsf{AD\text{-}AsgRes}(b,d,ch)$ assigns appropriate resources for the subtasks assigned to the AD.

$$
\begin{array}{lcl}
\mathsf{AD\text{-}AsgRes}(b,d,ch) & \overset{\text{def}}{=} & (\nu\, n,c)(\overline{b}\,\langle n,c\rangle\,.(c(k_1,\ldots,k_\zeta,m,p,g,b').\\
& & \quad (\nu\, o,ans_1,ans_2)\\
& & \quad (\overline{ch}\,\langle k_1,\ldots,k_\zeta,ans_1,ans_2\rangle\,.\\
& & \qquad (ans_1(cr_1,\ldots,cr_\zeta).\\
& & \qquad\quad \overline{p}\,\langle k_1,cr_1,...,k_\zeta,cr_\zeta,m,o\rangle\,.\mathsf{AD\text{-}AsgRes}\langle b',d,ch\rangle\\
& & \qquad +\\
& & \qquad ans_2.\overline{d}\,\langle k_1,\ldots,k_\zeta,m,p,g\rangle\,.\mathsf{AD\text{-}AsgRes}\langle b',d,ch\rangle)\\
& & \quad \mid o(X).X)\\
& & \quad +\, n.\mathsf{AD\text{-}AsgRes}\langle b,d,ch\rangle))
\end{array}
$$

on channels $b,d,$ and ch: it extracts a request of the queue through channels b and c, and proceeds to attend it. Then, $\mathsf{AD\text{-}AsgRes}(b,d,ch)$ interacts with the local resource manager process through channels ch, ans_1, and ans_2 in order to determine the resources for the request. If appropriate resources for the request are available then $\mathsf{AD\text{-}AsgRes}(b,d,ch)$ receives in ans_1 the access channels of the resource proxies and forwards them to the user proxy through name p. Otherwise, if there are no resources then $\mathsf{AD\text{-}AsgRes}(b,d,ch)$ receives an input in ans_2 and sends the request back to the queue.

Observe how also $\mathsf{AD\text{-}AsgRes}(b,d,ch)$ features higher-order communication in its interaction with task process $[\![\mathtt{S}]\!]^{t,e}$. In fact, using a higher-order process communication on name o (not shown), the task process $[\![\mathtt{S}]\!]^{t,e}$ is expected to send a job to $\mathsf{AD\text{-}AsgRes}(b,d,ch)$—which is denoted by process variable X. As soon as the reception on o takes place, process $\mathsf{AD\text{-}AsgRes}(b,d,ch)$ will execute the involved job.

User and Resource Proxies. We represent user proxies as instances of a process which receives the requirements of the subtasks of the user task process $[\![\mathtt{S}]\!]^{t,e}$ and submits such requirements to an access point process. Moreover, a user proxy process interacts with process $\mathsf{AD\text{-}AsgRes}(b,d,ch)$ which sends it the channels of the resource proxies of assigned resources. Finally, the user proxy process communicates with resources proxies process in order to authenticate and obtain the direct access to resources. Resource proxies are abstracted as a process which interacts with its associated resource process and instances of user proxy process. The interaction with its associated resource process allows the resource proxy to keep track of the state of the resource, as a resource notifies its proxy when a task has been completed.

Other components. In addition to the components described above, our process models also includes representations for other components in the GC system, namely logs processes, resource processes, and queue processes. There is a log process for each user task: it is in charge of registering the current state and the result of a task. Middleware processes (access points) interact to read the log when the user process requests it. In fact, processes $\mathsf{AP\text{-}ProxyHandler}(ce,\widetilde{d})$ and $\mathsf{AP\text{-}UserProxy}(ce,p,t,g)$ interact with the log process to register a new state and/or the final result. Resource processes abstract the behavior of actual grid resources. They interact with resource proxy process and task process $[\![\mathtt{S}]\!]^{t,e}$. Finally, the queue process is a process representation of a queue structure. There is a queue process for each AD, which is used to store the subtasks requests of resources assigned to the AD.

5 Formalizing a Representative Grid Scenario

We now illustrate our formal model by instantiating it with the scenario presented in Sect. 2 (see also Fig. 2). The following table summarizes some of the corresponding base sets:

Base Set		Description
D	$= \{d_1, d_2, d_3\}$	Administrative domains
U	$= \{u_1, u_2\}$	Users
V	$= \{v_1, v_2\}$	Virtual organizations
N	$= \{n_1, n_2\}$	Grid nodes
R	$= \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8\}$	Resources
K	$= \{k_1, k_2, k_3\}$	Resource descriptors
T	$= \{\mathtt{T}_1, \mathtt{T}_2\}$	Task definitions
S	$= \{\mathtt{S}_1, \mathtt{S}_2\}$	User tasks

For the sake of space, we do not present the static component of the model. Still, the description of the scenario given in Sect. 2 should provide an intuitive idea of the key valid relationships between the main grid components. We only highlight the fact that user tasks $\mathtt{S}_1$ and $\mathtt{S}_2$ are instances of task definitions $\mathtt{T}_1$ and $\mathtt{T}_2$, respectively. As for the dynamic component of the model, following the notation given in Fig. 5, our scenario is represented by the following HOπ process:

$$\begin{aligned}\mathsf{Grid}\,{}^{\omega,\mu}_{\delta,\eta} \;=\;& (\nu\, y_1,y_2)\big(\llbracket u_1,\mathtt{S}_1\rrbracket^{\widetilde{c_1},y_1} \mid \llbracket u_2,\mathtt{S}_2\rrbracket^{\widetilde{c_2},y_2} \mid \\ & (\nu\, d_1,d_2,d_3)(\llbracket n_1\rrbracket^{v_1,y_1,d_1,d_2} \mid \llbracket n_2\rrbracket^{v_2,y_2,d_2,d_3} \mid \llbracket d_1\rrbracket \mid \llbracket d_2\rrbracket \mid \llbracket d_3\rrbracket)\big)\end{aligned}$$

where $\omega = 2$, $\mu = 2$, $\delta = 2$, and $\eta = 2$. By expanding the definitions of $\llbracket u,\mathtt{S}\rrbracket^{\widetilde{c},y}$, $\llbracket n\rrbracket^{v,y,\widetilde{d}}$, and $\llbracket d_1\rrbracket$, the above process can be equivalently stated as follows:

$$\begin{aligned}\mathsf{Grid}\,{}^{\omega,\mu}_{\delta,\eta} \;=\;& (\nu\, y_1,y_2)\big((\nu\, t_1,e_1)\,\mathsf{User}\langle\widetilde{c_1},\llbracket \mathtt{S}_1\rrbracket^{t_1,e_1},y_1\rangle \mid (\nu\, t_2,e_2)\,\mathsf{User}\langle\widetilde{c_2},\llbracket \mathtt{S}_2\rrbracket^{t_2,e_2},y_2\rangle \mid \\ & (\nu\, d_1,d_2,d_3)(\mathsf{AP}^{v_1}\langle y_1,d_1,d_2\rangle \mid \mathsf{AP}^{v_2}\langle y_2,d_2,d_3\rangle \mid \mathsf{AD}\langle d_1\rangle \mid \mathsf{AD}\langle d_2\rangle \mid \mathsf{AD}\langle d_3\rangle)\big)\end{aligned}$$

To illustrate process evolution, we now describe a particular reduction sequence that originates from $\mathsf{Grid}\,{}^{\omega,\mu}_{\delta,\eta}$. Precisely, we show the interactions that occur when the user u_1 accesses the grid for executing task $\mathtt{S}_1$. Clearly, (concurrent) interactions related to user u_2 are also possible, but below we restrict to comment on the reductions related to the process representation of u_1.

First, we have a sequence of reductions $\mathsf{Grid}\,{}^{\omega,\mu}_{\delta,\eta} \Longrightarrow \mathsf{GRID}^1$, that represents the steps in which $\mathsf{User}\langle\widetilde{c_1},\llbracket \mathtt{S}_1\rrbracket^{t_1,e_1},y_1\rangle$ interacts with process $\mathsf{AP}^{v_1}\langle y_1,d_1,d_2\rangle$ to perform steps of user authentication, proxy creation, and submission of task $\mathtt{S}_1$, as stipulated in the protocol. Process GRID^1 is as follows:

$$\begin{aligned}\mathsf{GRID}^1 \;\equiv\;& (\nu\, y_1,y_2)\big(\mathsf{UsrMonitor}\langle\widetilde{c_1},g_{r1},a_1,y_1,\mathsf{P}\rangle \mid (\nu\, t_2,e_2)\mathsf{User}\langle\widetilde{c_2},\llbracket \mathtt{S}_2\rrbracket^{t_2,e_2},y_2\rangle \mid \\ & (\nu\, d_1,d_2,d_3)(\mathsf{AP}^1(\mathtt{S}_1) \mid \mathsf{RestSystem}^1)\big)\end{aligned}$$

where residual processes $\mathsf{AP}^1(\mathtt{S}_1)$ and $\mathsf{RestSystem}^1$ are as follows:

$$\begin{aligned}\mathsf{AP}^1(\mathtt{S}_1) \;\equiv\;& \llbracket \mathtt{S}_1\rrbracket^{t_1,e_1} \mid (\nu\, g_{w1},ce_1)(\mathsf{AP\text{-}Log}\langle g_{w1},g_{r1},\mathtt{submitted},\mathtt{null}\rangle \mid \\ & e_1(\widetilde{r}).\overline{g_{w1}}\langle \mathtt{state},\mathtt{finished}\rangle.\overline{g_{w1}}\langle \mathtt{result},\widetilde{r}\rangle \mid \mathsf{AP\text{-}UserProxy}\langle ce_1,a_1,t_1,g_{w1}\rangle \mid \\ & \mathsf{AP\text{-}ProxyHandler}\langle ce_1,d_1,d_2\rangle) \\ \mathsf{RestSystem}^1 \;\equiv\;& \mathsf{AP}^{v_1}\langle y_1,d_1,d_2\rangle \mid \mathsf{AP}^{v_2}\langle y_2,d_2,d_3\rangle \mid \mathsf{AD}\langle d_1\rangle \mid \mathsf{AD}\langle d_2\rangle \mid \mathsf{AD}\langle d_3\rangle\end{aligned}$$

In process $\mathsf{AP}^1(\mathtt{S}_1)$ above, private name g_{w1} is used by processes $\mathsf{AP\text{-}UserProxy}\langle ce_1, a_1, t_1, g_{w1}\rangle$ and $\mathsf{AP\text{-}ProxyHandler}\langle ce_1, d_1, d_2\rangle$ to register the changes in the state of task $\mathtt{S}_1$. Name ce_1 stands for the private channel on which these two processes may interact. At this point, we have the reduction sequence $\mathsf{GRID}^1 \Longrightarrow \mathsf{GRID}^2$, which represents reductions corresponding to the AD selection in the VO and the task submission to such an AD. In this case, we assume the AD d_1 is selected for the execution of the task. Process GRID^2 is as follows:

$$\begin{aligned}\mathsf{GRID}^2 \;\equiv\;& (\nu\, y_1, y_2)\big(\mathsf{UsrMonitor}\langle\widetilde{c_1}, g_{r1}, a_1, y_1, \mathsf{P}\rangle \mid (\nu\, t_2, e_2)\mathsf{User}\langle\widetilde{c_2}, [\![\mathtt{S}_2]\!]^{t_2,e_2}, y_2\rangle \mid \\ & (\nu\, d_1, d_2, d_3)(\mathsf{AP}^2(\mathtt{S}_1) \mid \mathsf{AD}^1\langle d_1\rangle \mid \mathsf{RestSystem}^2)\big)\end{aligned}$$

where $\mathsf{AP}^2(\mathtt{S}_1)$ and $\mathsf{AD}^1\langle d_1\rangle$ stand for residual processes for the access point and for the representation of AD d_1, respectively. As above, $\mathsf{RestSystem}^2$ stands for the composition of processes for the remaining components. In process $\mathsf{AD}^1\langle d_1\rangle$, the interaction between the task process and the user proxy process has evolved to $[\![\mathtt{S}_1^1]\!]$ and $\mathsf{AP\text{-}UserProxy}_1$, respectively. Processes $\mathsf{AP}^2(\mathtt{S}_1)$ and $\mathsf{RestSystem}^2$ are as follows:

$$\begin{aligned}\mathsf{AP}^2(\mathtt{S}_1) \;\equiv\;& [\![\mathtt{S}_1^1]\!] \mid (\nu\, g_{w1}, ce_1)(\mathsf{AP\text{-}Log}\langle g_{w1}, g_{r1}, \mathtt{queued}, \mathtt{null}\rangle \mid \\ & e_1(\widetilde{r}).\overline{g_{w1}}\langle \mathtt{state}, \mathtt{finished}\rangle.\overline{g_{w1}}\langle \mathtt{result}, \widetilde{r}\rangle \mid \mathsf{AP\text{-}UserProxy}_1 \mid \\ & \mathsf{AP\text{-}ProxyHandler}\langle ce_1, d_1, d_2\rangle) \\ \mathsf{ResSystem}^2 \;\equiv\;& \mathsf{AP}^{v_1}\langle y_1, d_1, d_2\rangle \mid \mathsf{AP}^{v_2}\langle y_2, d_2, d_3\rangle \mid \mathsf{AD}\langle d_2\rangle \mid \mathsf{AD}\langle d_3\rangle\end{aligned}$$

At this point, we may infer a reduction sequence which abstracts steps of resource selection and task execution. We indeed have $\mathsf{GRID}^2 \Longrightarrow \mathsf{GRID}^3$, where process GRID^3 is as follows:

$$\begin{aligned}\mathsf{GRID}^3 \;\equiv\;& (\nu\, y_1, y_2)\big(\mathsf{UsrMonitor}\langle\widetilde{c_1}, g_{w1}, a_1, y_1, \mathsf{P}\rangle \mid (\nu\, t_2, e_2)\mathsf{User}\langle\widetilde{c_2}, [\![\mathtt{S}_2]\!]^{t_2,e_2}, y_2\rangle \mid \\ & (\nu\, d_1, d_2, d_3)(\mathsf{AP}^3(\mathtt{S}_1) \mid \mathsf{AD}^2(\mathtt{S}_1) \mid \mathsf{ResSystem}^2)\big)\end{aligned}$$

where $\mathsf{AP}^3(\mathtt{S}_1)$ corresponds to residual process for the access point; process $\mathsf{AD}^2(\mathtt{S}_1)$ is its analogous for the representation of AD d_1. While process $[\![\mathtt{S}_1^2]\!] \equiv \overline{e_1}\langle\widetilde{res_1}\rangle$, process $\mathsf{AP}^3(\mathtt{S}_1)$ is as follows:

$$\begin{aligned}\mathsf{AP}^3(\mathtt{S}_1) \;\equiv\;& [\![\mathtt{S}_1^2]\!] \mid (\nu\, g_{w1}, ce_1)(\mathsf{AP\text{-}Log}\langle g_{w1}, g_{r1}, \mathtt{running}, \mathtt{null}\rangle \mid \\ & e_1(\widetilde{r}).\overline{g_{w1}}\langle \mathtt{state}, \mathtt{finished}\rangle.\overline{g_{w1}}\langle \mathtt{result}, \widetilde{r}\rangle \mid \mathsf{AP\text{-}UserProxy}\langle ce_1, a_1, t_1, g_{w1}\rangle\rangle \mid \\ & \mathsf{AP\text{-}ProxyHandler}\langle ce_1, d_1, d_2\rangle)\end{aligned}$$

Process $[\![\mathtt{S}_1^2]\!]$ stands for the residual process for the task process of user u_1; it notifies its completion through channel e_1. We obtain the reduction sequence $\mathsf{GRID}^3 \Longrightarrow \mathsf{GRID}^4$ after some reductions corresponding to task completion and log registering. Process GRID^4 is as follows:

$$\begin{aligned}\mathsf{GRID}^4 \;\equiv\;& (\nu\, y_1, y_2)\big(\mathsf{UsrMonitor}\langle\widetilde{c_1}, g_{w1}, a_1, y_1, \mathsf{P}\rangle \mid (\nu\, t_2, e_2)\mathsf{User}\langle\widetilde{c_2}, [\![\mathtt{S}_2]\!]^{t_2,e_2}, y_2\rangle \mid \\ & (\nu\, d_1, d_2, d_3)(\mathsf{AP}^4(\mathtt{S}_1) \mid \mathsf{AD}^3(\mathtt{S}_1 \mid \mathsf{RestSystem}^2)\big)\end{aligned}$$

where $\mathsf{AP}^4(\mathtt{S}_1)$ is as follows:

$$\begin{aligned}\mathsf{AP}^4(\mathtt{S}_1) \;\equiv\;& (\nu\, g_{w1}, ce_1)(\mathsf{AP\text{-}Log}\langle g_{w1}, g_{r1}, \mathtt{finished}, \widetilde{res_1}\rangle \mid \\ & \mathsf{AP\text{-}UserProxy}\langle ce_1, a_1, t_1, g_{w1}\rangle\rangle \mid \mathsf{AP\text{-}ProxyHandler}\langle ce_1, d_1, d_2\rangle)\end{aligned}$$

At last, we may infer the reduction sequence $\mathsf{GRID}^4 \Longrightarrow \mathsf{GRID}^5$, where process GRID^5 defined as

$$\begin{aligned}\mathsf{GRID}^5 \;\equiv\;& (\nu\, y_1, y_2)\big(\mathsf{P}\langle\widetilde{res_1}\rangle \mid (\nu\, t_2, e_2)\mathsf{User}\langle\widetilde{c_2}, [\![\mathtt{S}_2]\!]^{t_2,e_2}, y_2\rangle \mid \\ & (\nu\, d_1, d_2, d_3)(\mathsf{AP}^4(\mathtt{S}_1) \mid \mathsf{AD}^3(\mathtt{S}_1) \mid \mathsf{RestSystem}^2)\big)\end{aligned}$$

and where process $\mathsf{P}\langle\widetilde{res_1}\rangle$ denotes an unspecified, parameterized process that is to be executed by the user monitor with the task result $\widetilde{res_1}$.

6 Future Work

The process model of GC systems presented here describes basic interactions among grid main components, abstracting and enforcing essential static and dynamic properties of such systems. Establishing an operational correspondence result connecting the invariants in the static description and the HOπ reductions of the dynamic representation is part of ongoing work. We conjecture that HOπ processes representing the dynamic part preserve by construction the invariants defined by the static part. Slightly more formally, we conjecture that if process P respects the static invariants, and $P \longrightarrow P$ then either (a) P' preserves the static invariants, or (b) there is a P'' such that $P' \Longrightarrow P''$ and P'' preserves the static invariants. One of the challenges in the proof consists in giving a unified treatment to all invariants.

Our current model does not take into account certain aspects typical of GC infrastructures, such as time. Still, as already mentioned, we think our current model is already a good basis for extensions: the inherent compositionality of process specifications should ease orthogonal improvements and refinements. In this sense, as future work, we plan to refine the model with locations (i.e., computation sites) and process failures. To this end, an initial approach would be using a calculus of *adaptable processes* [2], which enables to incorporate forms of runtime adaptation over located, interacting processes.

A strong motivation for pursuing a process calculi model of GC systems is that of exploiting the proof techniques over processes (behavioral equivalences, type systems) so as to reason about grid systems. That is, we would like to explore how our process model allows us to reason about correctness properties of GC systems. This involves, for instance, exploiting our model's compositionality and well-established theories of behavioral equivalence to reason about arbitrary behaviors in the grid setting. Also, we would like to reason about task termination and resource delivery in the grid setting. These properties are intrinsically related to reachability problems, and to issues of deadlock- and cycle-detection. We believe that a process calculi model offers a suitable basis also for investigating such problems.

Acknowledgments We are grateful to Carlos Olarte and to the DCM'13 reviewers for their detailed comments and remarks. Also, exchanges with John Sanabria were crucial to improve our understanding of grid computing platforms. The research of first author has been supported by COLCIENCIAS (the Colombian Agency for Science, Technology and Innovation) under a "Francisco José de Caldas" doctoral scholarship (Convocatoria 494/2009). The research of the second author was supported by the Portuguese Foundation for Science and Technology (FCT) under grants SFRH / BPD / 84067 / 2012 and CITI.

References

[1] Wil M. P. van der Aalst, Carmen Bratosin, Natalia Sidorova & Nikola Trcka (2010): *A reference model for grid architectures and its validation*. *Concurrency and Computation: Practice and Experience* 22(11), pp. 1365–1385. Available at `http://dx.doi.org/10.1002/cpe.1505`.

[2] Mario Bravetti, Cinzia Di Giusto, Jorge A. Pérez & Gianluigi Zavattaro (2012): *Adaptable processes*. *Logical Methods in Computer Science* 8(4). Available at `http://dx.doi.org/10.2168/LMCS-8(4:13)2012`.

[3] Nadia Busi, Maurizio Gabbrielli & Gianluigi Zavattaro (2009): *On the expressive power of recursion, replication and iteration in process calculi*. *Mathematical Structures in Computer Science* 19(6), pp. 1191–1222. Available at `http://dx.doi.org/10.1017/S096012950999017X`.

[4] Y. Du, C. Jiang & Y. Guo (2006): *Towards a formal model for grid architecture via Petri Nets*. *Information Technology Journal* 5(5), pp. 833–841. Available at `http://dx.doi.org/10.3923/itj.2006.833.841`.

[5] I. Foster, Yong Zhao, I. Raicu & Shiyong Lu (2008): *Cloud Computing and Grid Computing 360-Degree Compared*. In: *Grid Computing Environments Workshop, 2008. GCE '08*, pp. 1–10. Available at `http://dx.doi.org/10.1109/GCE.2008.4738445`.

[6] Ian Foster, Carl Kesselman & Steven Tuecke (2001): *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. *Int. J. High Perform. Comput. Appl.* 15(3), pp. 200–222. Available at `http://dx.doi.org/10.1177/109434200101500302`.

[7] Ian T. Foster, Carl Kesselman, Gene Tsudik & Steven Tuecke (1998): *A Security Architecture for Computational Grids*. In Li Gong & Michael K. Reiter, editors: *ACM Conference on Computer and Communications Security*, ACM, pp. 83–92. Available at `http://doi.acm.org/10.1145/288090.288111`.

[8] Cinzia Di Giusto, Jorge A. Pérez & Gianluigi Zavattaro (2009): *On the Expressiveness of Forwarding in Higher-Order Communication*. In: *ICTAC*, *Lecture Notes in Computer Science* 5684, Springer, pp. 155–169. Available at `http://dx.doi.org/10.1007/978-3-642-03466-4_10`.

[9] Vasileios Koutavas & Matthew Hennessy (2011): *A Testing Theory for a Higher-Order Cryptographic Language - (Extended Abstract)*. In Gilles Barthe, editor: *ESOP*, *Lecture Notes in Computer Science* 6602, Springer, pp. 358–377. Available at `http://dx.doi.org/10.1007/978-3-642-19718-5_19`.

[10] Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi & Alan Schmitt (2011): *On the expressiveness and decidability of higher-order process calculi*. *Inf. Comput.* 209(2), pp. 198–226. Available at `http://dx.doi.org/10.1016/j.ic.2010.10.001`.

[11] Robin Milner (1989): *Comunication and Concurrency*. Prentice Hall.

[12] Robin Milner, Joachim Parrow & David Walker (1992): *A calculus of mobile processes, I*. *Inf. Comput.* 100(1), pp. 1–40. Available at `http://dx.doi.org/10.1016/0890-5401(92)90008-4`.

[13] Zsolt Németh & Vaidy S. Sunderam (2003): *Characterizing Grids: Attributes, Definitions, and Formalisms*. *J. Grid Comput.* 1(1), pp. 9–23. Available at `http://dx.doi.org/10.1023/A:1024011025052`.

[14] Carlos A. Ramírez, Jorge A. Pérez, Jesús Aranda & Juan F. Díaz (2013): *Towards Formal Interaction-based Models of Grid Computing Infrastructures (Extended Version)*. Available at `http://tinyurl.com/kkcadba`.

[15] Davide Sangiorgi (1992): *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, University of Edinburgh, Dept. of Comp. Sci.

[16] Davide Sangiorgi (1996): *Bisimulation for Higher-Order Process Calculi*. *Inf. Comput.* 131(2), pp. 141–178. Available at `http://dx.doi.org/10.1006/inco.1996.0096`.

[17] Davide Sangiorgi & David Walker (2001): *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA.

[18] K. Stanoevska-Slabeva, T. Wozniak & S. Ristol (2009): *Grid and Cloud Computing: A Business Perspective on Technology and Applications*. Springer.

[19] Chuliang Weng, Xinda Lu & Qianni Deng (2003): *Formalizing Service Publication and Discovery in Grid Computing Systems*. In: *GCC (1)*, *Lecture Notes in Computer Science* 3032, Springer, pp. 669–676. Available at `http://dx.doi.org/10.1007/978-3-540-24679-4_118`.

[20] Li Zhan-jun, Huang Yong-zhong & Guo Shao-zhong (2009): *Using Pi-Calculus to Formalize Grid Workflow Parallel Computing Patterns*. In: *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, ITNG '09, IEEE Computer Society, Washington, DC, USA, pp. 1568–1571. Available at `http://dx.doi.org/10.1109/ITNG.2009.63`.

[21] Jing Zhou & Guosun Zeng (2009): *A mechanism for grid service composition behavior specification and verification*. *Future Generation Comp. Syst.* 25(3), pp. 378–383. Available at `http://dx.doi.org/10.1016/j.future.2008.02.013`.