Virtual Organizations in Arigatoni

Michel Cosnard [a] Luigi Liquori [a] Raphael Chand [a]

[a] *INRIA, France*

Abstract

Arigatoni is a lightweight overlay network that deploys the *Global Computing Paradigm* over the Internet. Communication for over the behavioral units of the overlay is performed by a simple *resource discovery protocol* (RDP). Basic Global Computers Units (GC) can communicate by first registering to a brokering service and then by mutually asking and offering services.

Colonies and communities are the main entities in the model. A colony is a simple virtual organization composed by exactly one leader and some set (possibly empty) of individuals. A community is a raw set of colonies and global computers (think it as a *soup* of colonies and global computer without a leader).

We present an operational semantics via a labeled transition system, that describes the main operations necessary in the Arigatoni model to perform leader negotiation, joining/leaving a colony, linking two colonies and moving one GC from one colony to another. Our formalization results to be adequate w.r.t. the algorithm performing peer logging/delogging and colony aggregation.

1 Introduction

Effective use of computational grids via P2P systems requires *up-to-date* information about widely-distributed resources. This is a challenging problem for very large distributed systems particularly when taking into account the continuously changing state of resources. Discovering dynamic resources must be scalable in number of resources and users and hence, as much as possible, fully decentralized. It should tolerate intermittent participation and dynamically changing status/availability.

The Arigatoni Model is suitable to deploy, via the Internet the *Global Computing Communication Paradigm*, *i.e.* computation via a seamless, geographically distributed, open-ended network of bounded resources by agents acting with partial knowledge and no central coordination. The model can be deployed firstly in an intranet and further from intranet to intranet by overlapping an *Overlay Network* on the top of the *actual network*. An overlay network is an abstraction on top of a global network to yield another global network. Overlay examples are *resource* *discovery* services (notion of resource sharing in distributed networks), search engines (abstraction of information repository), or systems of trusted mobile agents (notion of autonomic, exploratory behavior) [5].

The Arigatoni model provides the necessary basic infrastructure necessary for a real deployment of the overlay network itself. Moreover, our work abstracts on *which kind of resource* the overlay network is playing with; pragmatically speaking, this work could be useful for Grid, or for distributed file/band sharing, or for more evolved scenarios like mobile and distributed object-oriented computation.

The Arigatoni communication model is organized in *colony* governed by a clear leader. Global computers belong to only one colony, and requests for resources located in the same or in another colony traverse a broker-2-broker negotiation whose security is guaranteed via PKI mechanisms.

The model is suitable to fit with various global scenarios from classical P2P applications, like file or band sharing, to more sophisticated Grid applications, like remote and distributed big (and small) computations, until possible, futuristic *migration computations*, *i.e.* transfer of a non completed local run in another GC, the latter scenario being useful in case of catastrophic scenarios, like fire, terrorist attack, earthquake etc., in the vein of Global Programming Languages *à la* Obliq or Telescript.

The Units in the Arigatoni model are:

- A *Global Computer Unit,* GC, *i.e.* the basic peer of the global computer paradigm; it is typically a small device, like a PDA, a laptop or a PC, connected via IP, unrelated to the media used, wired or wireless, etc.
- A *Global Broker Unit,* GB, is the basic unit devoted to register and unregister GCs, to receive service queries from client GCs, to contact potential servants GCs, to negotiate with the latter the given services, to trust clients and servers, and to send all the information necessary to allow the client GC and the servants GCs to communicate. Every GB controls a *colony* (denoted by COL) of collaborating global computers. Hence, communication intra-colony is initiated via only one GB, while communication inter-colonies is initiated through a chain of GB-2-GB message exchanges. In both cases, when a client GC receives an acknowledgment for a request service (with related trust certificate) from the proper GB, then the client will enjoy the service directly from the servant(s) GC, *i.e.* without a further mediation of the GB itself.
- A *Global Router Unit,* GR is a simple basic unit that is devoted to send and receive packets using a proper Resource Discovery Protocol [3] and to forward the "payload" to the units which are connected with this router. Every GC and every GB has one personal GR, with which it communicates via a suitable API. The connection between router and peer is ensured via a suitable API.

Colonies and Individuals are the main entities in the model. A colony is a simple virtual organization composed by exactly one leader and some set (possibly empty) of individuals. Individuals are global computers (think it as an *Amoeba*), or (sub)colonies (think it as a *Protozoa*). A formal definition of a colony is given

$\{\mathsf{GB}\}$ is a (small) colony

$\{\mathsf{GB}_1, \mathsf{GC}_1 \ldots \mathsf{GC}_m\}$ is a colony

$$\{\mathsf{GB}_1, \mathsf{GC}_1 \ldots \mathsf{GC}_m, \overbrace{\{\mathsf{GB}_2, \mathsf{GC}_{m+1} \ldots \mathsf{GC}_{m+n}\}}^{subcolony}\}$$

is a colony (it contains a subcolony)

$$\{\mathsf{GB}_1, \mathsf{GC}_1 \ldots \mathsf{GC}_m, \mathsf{GB}_2, \mathsf{GC}_{m+1} \ldots \mathsf{GC}_{m+n}\}$$

is not a colony (two GBs)

$$\{\mathsf{GB}_3, \overbrace{\{\mathsf{GB}_1, \mathsf{GC}_1 \ldots \mathsf{GC}_m\}}^{subcolony}, \overbrace{\{\mathsf{GB}_2, \mathsf{GC}_{m+1} \ldots \mathsf{GC}_{m+n}\}}^{subcolony}\}$$

is a colony (with two subcolonies)

$$\{\overbrace{\{\mathsf{GB}_1, \mathsf{GC}_1 \ldots \mathsf{GC}_m\}}^{subcolony}, \overbrace{\{\mathsf{GB}_2, \mathsf{GC}_{m+1} \ldots \mathsf{GC}_{m+n}\}}^{subcolony}\}$$

is not a colony (no leader in the toplevel colony) but it is a community

Figure 1. Some Colony's Examples

using this simple BNF syntax:

$$\mathsf{COL} ::= \{\mathsf{GB}\} \mid \mathsf{COL} \cup \{\mathsf{GC}\} \mid \mathsf{COL} \cup \{\, \mathsf{COL} \,\}$$

The two main characteristics of a colony are:

(i) a colony has *exactly* one leader GB and at least one individual (the GB itself);

(ii) a colony contains individuals (some GC's, or other colonies).

Some examples of colonies are shown in Figure 1.

A Community (denoted by COM) is a raw set of colonies and global computers (think it as a *soup* of colonies and GC without a leader). A formal definition of community is given using the BNF syntax:

$$\mathsf{COM} ::= \emptyset \mid \mathsf{COM} \cup \{\mathsf{GC}\} \mid \mathsf{COM} \cup \{\, \mathsf{COL} \,\}$$

A simple example of a community is shown in Figure 1. As one can see from the abstract syntax, a colony is a community but the reverse is not true.

Resource Discovery is one of the key issues in building overlay computer networks. Individuals (global computers) can register and unregister to a colony. The same holds true for the subcolonies that, in turn, can (un)register to another colony. The main difficulty in (un)registering is dealing with *Administrative Domains*: as well stated in the seminal Cardelli-Gordon's paper on Mobile Ambients [2]:

"In the early days of the Internet *one could rely on a flat name space given by* IP *addresses; knowing the* IP *address of a computer would very likely allow now to talk to that computer in some way. This is no longer the case: firewalls partition the* Internet *into administrative domains that are isolated from each other except for rigidly controlled pathways. System administrators enforce policies about what can move through firewalls and how [...]"*

(Un)Registering Modalities. There are essentially two ways of registering to a GB leader of a colony, the latter being not enforced by the Arigatoni model:

- registration of an individual (GC or colony) to a GB leader of a colony belonging to the same *current administrative domain*;
- registration via *remote tunnelling* of an individual (GC or colony) to another GB leader of a colony belonging to a *different administrative domain*. In this case, we say that the individuals *de facto* are working in local mode *in the current administrative domain* and in global mode *in another administrative domain*.

  In addition to this remote registration, the same individual can still register to the GB leader of the colony belonging to the same administrative domain in which it resides. As such, in its global mode, it will belong to the colony of the current administrative domain, and, in its local mode (via remote tunnelling), it will belong to another colony in another administrative domain.

Counterwise, an individual can unregister according to the following simple rules *d'étiquette*:

- unregistration is possible only when there are no pending services demanded or requested to the leader GB of the colony it belongs: it must wait for an answer of the leader GB or for a direct connection of the GC requesting the already offered service, or wait for a timeout. The colony accepts the unregistration only if the colony itself will not be *corrupted*;
- (as a corollary of the above) a GB cannot unregister from its own colony, *i.e.* it cannot discharge itself. However, for fault tolerance purposes, a GB can be faulty. In that case, the GCs will unregister one after the other and the colony will "disappear";
- once a GC (*e.g.* a laptop) has been disconnected from a colony belonging to any administrative domain, it can migrate in another colony belonging to any other administrative domain;

Summarizing, the original contributions of the paper are:

- a formalization of the *Virtual Intermittence Protocol* (VIP) in terms of a labeled transition system; in our modest opinion, this is the first attempt to capture the behavior of an intermittence protocol using formal methods and a labeled transition semantics mathematical tool. Advantages of this approach is that rely on robust mathematical basis as languages and concurrency theory. As such, formal and mechanical proof on protocols related with overlay networks is feasible.
- a *complete domain independence* of the model w.r.t. other models in the literature. In other words Arigatoni completely abstracts of its use, *i.e.* Grid, file/band sharing, web services, etc.
- some simulation results of the intermittent participation for a given network topology.

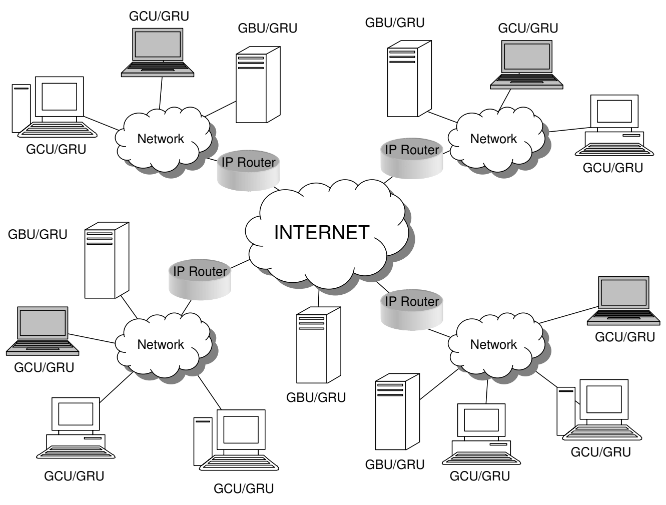

Figure 2. ArigatoNet

2 Units in a Nutshell

A complete description of all the functional units of the Arigatoni model is given in [1]; this section is an overview.

2.1 Global Computer Unit

In the Arigatoni model, a *Global Computer Unit (*GC*)* is a cheap computer device. The computer should be able to work in *Standalone Local Mode* for all the tasks that it can do locally or in *Global Mode*, by first registering itself in the Arigatoni overlay, and then by making a global request to the overlay network induced by the model. Figure 2 shows the Arigatoni model. The GC must be able to perform the following tasks:

- Discover, upon the physical arrival of the GC in a new colony, the address of a GB, representing the *leader* of the colony;
- Register/Unregister on the GB which manages the colony;
- Request some services to its GB, and respond to some requests from the GB;
- Upon reception from a GB of a positive response to a request, be able to connect directly with the servant(s) GC in a P2P fashion, and offer/receive the service.

2.2 Global Broker Unit

The *Global Broker Unit (*GB*)* performs the following tasks:

- Discover the address of another *super* GB, representing the *superleader* of the *supercolony*, where the GB's colony is embedded. We assume that every GB comes with its proper PKI certificate.

- Register/Unregister the proper colony to the *leader* GB which manages the supercolony;
- Register/Unregister clients and servants GC in its local base of global computers. By definition every GC can register to *at most* one GB;
- Acknowledge the request of service of the client GC;
- Discover the resource(s) that satisfies the GC's request in its local base (local colony) of GC;
- Delegate the request to another GB governing another colony;
- Perform a combination of the above two actions;
- Deal with all PKI intra- and inter-colony policies;
- Notify the client GC (or the delegating GB) that some servant(s) GCs have *accepted* to serve the request, or just notify a *failure* of the request.

Every GC in the colony sends its request to the GB which is the leader of the colony. There are different scenarios concerning the demanded resource for service discovery, namely:

(i) The broker finds all the resource(s) needed to satisfy the requested services of the GC client locally in the intranet. Then it will send all the information necessary to make the GC client able to communicate with the GC servants. This notification will be encoded using the RDP protocol. Then, the GC client will directly talk with GC servant(s), and the latter will manage the request, as in classical P2P systems;

(ii) The broker did not find all the resource(s) in its local intranet. In this case it will forward and delegate the request to another broker. For that purpose, it must first register the whole colony to another supercolony;

(iii) A combination of steps 1 + 2 could be envisaged depending on the capability of the GB to combine resources that it manages and resources that come from a delegate GB;

(iv) After a fixed *timeout period*, or when all delegate GBs have failed to satisfy the delegated request, the broker will notify the GC client of the *refusal of service*.

2.3 Global Router Unit

The last unit in the Arigatoni model is the *Global Router Unit (*GR*)*. The GR implements all the low-level network routines, those which really have access to the IP network. It is the only unit which effectively runs the RDP protocol. The GR can be implemented as a small daemon which runs on the same device as a GC or a GB, or as a shared library dynamically linked with a GC or a GB. The GR is devoted to the following tasks:

- Upon the initial startup of a GC it helps to register the unit to a GB;
- It checks the well-formedness and forwards RDP packets across the overlay

toward their destinations. RDP packets encode the requests of a GC or a GB in the Arigatoni network;

- Upon the initial startup of a GB it helps the unit with several other GBs that it knows or discovers.

2.4 Unit Semantics

The formal semantics of the three formal units was first presented in [1]: Figures 3 and 4 show the pseudo code embedded inside a GC and a GB. We write in blue the code not essential to the semantics of peer discovery and the virtual (un)growth of colonies, and we highlight in red the code which is essential.

```
inparallel
while true do                          // Registration loop
 GBU = Discover(MyCard)
 case (GlobalMode,RegMode) is
  (true,false):
   ServiceReg(MyCard,GBU,LOGIN)
  (false,true):
   ServiceReg(MyCard,GBU,LOGOUT)
  otherwise:                           // Do nothing
 endcase
endwhile
with
 while true do                         // Shell loop
  Data     = ListenLocal()
  Response = LocalServe(Data)
  case (Response,GlobalMode,RegMode) is
   (login,_,_):                        // Open global mode
    GlobalMode = true
   (logout,_,_):                       // Close global mode
    GlobalMode = false
   (true,true):                        // Ask to the GBU
    MetaData = PackScenario(Data)
    ServiceRequest(MyCard,GBU,MetaData)
   otherwise: LocalReply(Response)
  endcase
 endwhile
with
 while RegMode do                      // Global GBU listening
  MetaData = ListenGBU()
  case MetaData.CMD.SERVICE is
   SREG:// GBU responds if it accepts my registration
    if   CanJoin(MetaData)
    then RegMode = true
    endif
    if   CanLeave(MetaData)
    then RegMode = false
    endif
   SREQ:                               // GBU is asking for some resources
    if   CanHelp(MetaData)
    then ServiceResponse(MyCard,GBU,ACC)
    else ServiceResponse(MyCard,GBU,REJ)
    endif
   SRESP:                              // GBU responds if it found some resources
    if   CanServe(MetaData)
    then Peers    = GetPeers(MetaData)
         Response = GlobalServe(MyCard,
                                Peers,MetaData)
         ServiceResponse(MyCard,GBU,DONE)
         LocalReply(Response)
    else LocalReply(fail)
    endif
  endcase
 endwhile
with
 while RegMode do                      // Global GCU listening
  MetaData = ListenGCU()
  if   Verify(MetaData)
  then Data     = UnPackScenario(MetaData)
       Response = LocalServe(Data)
       if   Response == fail
       then ServiceResponse(MyCard,GBU,ERR)
       else ServiceResponse(MyCard,GBU,DONE)
            SendResult(MyCard,GCU,Response)
       endif
  else ServiceResponse(MyCard,GBU,SPOOF)
  endif
 endwhile
endinparallel
```

Figure 3. GC pseudocode

3 Formal Semantics of the Virtual Organization

Let $\{\ldots\}$ denotes a colony and not necessarily an administrative domain (like in Cardelli-Gordon ambients), and let every individual come with its own IP address and security certificate. Let $\{\mathsf{GB}, \ldots\}$ denotes a colony with its leader, *e.g.*

$$\{\mathsf{GB}, \mathsf{COL}_1, \mathsf{COL}_2, \mathsf{GC}_1, \mathsf{GC}_2, \ldots\}$$

```
inparallel
while true do                              // Registration loop
 GBU = Discover(MyCard)
 case (GlobalMode,RegMode) is
  (true,false):
   ServiceReg(MyCard,GBU,LOGIN)
  (false,true):
   ServiceReg(MyCard,GBU,LOGOUT)
  otherwise:                               // Do nothing
 endcase
endwhile
with
 while true do                             // Shell loop
  Data     = ListenLocal()
  Response = LocalServe(Data)
  case (Response,GlobalMode,RegMode) is
   (login,_,_):                            // Open global mode
    GlobalMode = true
   (logout,_,_):                           // Close global mode
    GlobalMode = false
   (fail,true,true):                       // You ask for you
    MetaData = PackScenario(Data)
    ServiceRequest(MyCard,MyCard,MetaData)
   otherwise: LocalReply(Response)
  endcase
 endwhile
with
 while true do                             // Intra-colony listening
  MetaData = ListenPeer()
  PushHistory(MetaData)
  case MetaData.CMD.SERVICE is
   SREG:          // A Peer is asking for (un)registration
    Update(Colony,MetaData)
   SREQ:              // A Peer is asking for some request
    SubColony = SelectPeers(Colony,MetaData)
    if SubColony == {}                     // Broadcast inter
    then
      ServiceRequest(MyCard,GBU,MetaData)
    endif
    foreach Peer in SubColony do
// Broadcast intra
     ServiceRequest(MyCard,Peer,MetaData)
    endforeach
   SRESP:                  // A GCU responds to a request
    Sort&PushPeers4Id(MetaData)
  endcase
 endwhile
with
 while true do                             // Spooling Peers4Id
  foreach (Id,Peers) in Peers4Id do
   if   Timeout(Id)
   then ServiceResponse(MyCard,{},NOTIME)
   else if Satisfy(Peers,History(Id))
        then
         ServiceResponse(MyCard,
                         GetBestPeers4Id(Id),
                         DONE)
        endif
   endif
   PopPeers4Id(Id)
  endforeach
 endwhile
with
 while RegMode do                          // Inter-colony listening
  MetaData = ListenGBU()
  PushHistory(MetaData)
  case MetaData.OPE is
   SREG:                                   // Registration inter GBU
     ... as for SREQ intra-colony
   SREQ:
     ... as for SREQ intra-colony
   SRESP:             // A leader GBU responds to a request
    Sort&PushPeers4Id(MetaData)
     endcase
  endcase
 endwhile
endinparallel
```

Figure 4. GB pseudocode

is a colony with two subcolonies and two GCs highlighted. A colony is virtually addressed by the IP of its GB leader. Let a community be denoted by $\{\ldots\}$, *e.g.*

$$\{\mathsf{COL}_1, \mathsf{COL}_2, \mathsf{GC}_1, \mathsf{GC}_2\}$$

is a community with two subcolonies and two GC's.

We present an operational semantics via a reduction relation "$\rightarrow$", between communities, that describes the main operations necessary in the Arigatoni model to perform leader discovery and colony's service registration, namely joining/leaving a colony, linking two colonies and moving one GC from one colony to another.

As usual in process algebras, the reduction is quotiented by a set theoretical equivalence between communities. As remarked by Michele Bugliesi during the workshop, we omit in the reduction rules all the imperative aspects related to the changing of *state* of individuals; we focus only on the functional rules of the protocol describing the intermittent participation of individuals. The reduction rules are listed below with a concise explication.

(i) A GC joins a colony *in the same Administrative Domain*

$$\frac{\begin{array}{c} \mathit{discover}(\mathsf{GC}) = \mathsf{GB} \\ \mathit{samedom}(\mathsf{GB},\mathsf{GC}) = \textit{true} \quad \mathit{gmode}(\mathsf{GC}) = \textit{true} \\ \mathit{accept}(\mathsf{GB},\mathsf{GC}) = \textit{true} \ \mathit{regmode}(\mathsf{GC}) = \textit{false} \end{array}}{\{\{\mathsf{GB},\ldots\},\mathsf{GC}\} \to \{\{\mathsf{GB},\mathsf{GC},\ldots\}\}} \ (\mathsf{JoinGCU})$$

- $\mathit{discover}(\mathsf{GC}) = \mathsf{GB}$ discovers the leader-GB unit, upon physical/logical insertion of the GC in the Arigatoni network;
- $\mathit{samedom}(\mathsf{GB},\mathsf{GC}) = \textit{true}$: both the broker and the global computer reside in the same administrative domain;
- $\mathit{accept}(\mathsf{GB},\mathsf{GC}) = \textit{true}$: the broker accepts the global computer in its colony;
- $\mathit{gmode}(\mathsf{GC}) = \textit{true}\ \&\ \mathit{regmode}(\mathsf{GC}) = \textit{false}$: the global computer is in global mode but not yet registered. The side effect of this rule is to set the registration mode to *true*.

(ii) A GC leaves a colony *in the same Administrative Domain*

$$\frac{\begin{array}{c} \mathit{pendingip}(\mathsf{GC}) = \textit{false} \\ \mathit{samedom}(\mathsf{GB},\mathsf{GC}) = \textit{true} \quad \mathit{gmode}(\mathsf{GC}) = \textit{false} \\ \mathit{accept}(\mathsf{GB},\mathsf{GC}) = \textit{false} \ \mathit{regmode}(\mathsf{GC}) = \textit{true} \end{array}}{\{\{\mathsf{GB},\mathsf{GC},\ldots\}\} \to \{\{\mathsf{GB},\ldots\},\mathsf{GC}\}} \ (\mathsf{LeaveGCU})$$

- $\mathit{pendingip}(\mathsf{GC}) = \textit{false}$: the global computer has no pending services to give to its leader;
- $\mathit{samedom}(\mathsf{GB},\mathsf{GC}) = \textit{true}$: both the broker and the global computer reside in the same administrative domain;
- $\mathit{accept}(\mathsf{GB},\mathsf{GC}) = \textit{false}$: the broker accepts to delog the global computer in its colony;
- $\mathit{gmode}(\mathsf{GC}) = \textit{false}\ \&\ \mathit{regmode}(\mathsf{GC}) = \textit{true}$: the global computer is in local mode but still registered. The side effect of this rule is to set its registration mode to *false*.

(iii) A subcolony joins a colony *in the same Administrative Domain*

$$\frac{\begin{array}{c} \mathit{discover}(\mathsf{GB}_2) = \mathsf{GB}_1 \\ \mathit{samedom}(\mathsf{GB}_1,\mathsf{GB}_2) = \textit{true} \quad \mathit{gmode}(\mathsf{GB}_2) = \textit{true} \\ \mathit{accept}(\mathsf{GB}_1,\mathsf{GB}_2) = \textit{true} \quad \mathit{regmode}(\mathsf{GB}_2) = \textit{false} \end{array}}{\{\{\mathsf{GB}_1,\ldots\},\{\mathsf{GB}_2,\ldots\}\} \to \{\{\mathsf{GB}_1,\{\mathsf{GB}_2,\ldots\},\ldots\}\}} \ (\mathsf{JoinCol})$$

- $\mathit{discover}(\mathsf{GB}_2) = \mathsf{GB}_1$: the broker GB_2 discovers the broker GB_1, upon physical/logical insertion in the Arigatoni network;
- $\mathit{samedom}(\mathsf{GB}_1,\mathsf{GB}_2) = \textit{true}$: both reside in the same administrative domain;

- $accept(\mathsf{GB}_1, \mathsf{GB}_2) = \textit{true}$: the broker GB_1 accepts the subcolony in its colony;
- $gmode(\mathsf{GB}_2) = \textit{true}$ & $regmode(\mathsf{GB}_2) = \textit{false}$: the broker GB_2 is in global mode but not yet registered. The side effect of this rule is to set its registration mode to *true*.

(iv) A subcolony leaves a colony *in the same Administrative Domain*

$$\frac{\begin{array}{rl} pendingip(\mathsf{GB}_2) = \textit{false} & \\ samedom(\mathsf{GB}_1, \mathsf{GB}_2) = \textit{true} & gmode(\mathsf{GB}_2) = \textit{false} \\ accept(\mathsf{GB}_1, \mathsf{GB}_2) = \textit{false} & regmode(\mathsf{GB}_2) = \textit{true} \end{array}}{\{\{\mathsf{GB}_1, \{\mathsf{GB}_2, \ldots\}, \ldots\}\} \rightarrow \{\{\mathsf{GB}_1, \ldots\}, \{\mathsf{GB}_2, \ldots\}\}}\ (\mathsf{LeaveCol})$$

- $pendingip(\mathsf{GB}_2) = \textit{false}$: the broker GB_2 has no pending services to give to its leader GB_1;
- $samedom(\mathsf{GB}_1, \mathsf{GB}_2) = \textit{true}$: both reside in the same administrative domain;
- $accept(\mathsf{GB}_1, \mathsf{GB}_2) = \textit{false}$: the broker GB_1 does not accept the subcolony in its colony;
- $gmode(\mathsf{GB}_2) = \textit{false}$ & $regmode(\mathsf{GB}_2) = \textit{true}$: the broker GB_2 is in local mode but still registered. The side effect of this rule is to set its registration mode to *false*.

(v) Linking two colonies *in different Administrative Domains*

$$\frac{\begin{array}{rl} & gmode(\mathsf{GB}_1) = \textit{true} \\ newgbu(\mathsf{GB}_1, \mathsf{GB}_2) = \mathsf{GB}_3 & gmode(\mathsf{GB}_2) = \textit{true} \\ samedom(\mathsf{GB}_1, \mathsf{GB}_2) = \textit{false} & regmode(\mathsf{GB}_1) = \textit{false} \\ agree(\mathsf{GB}_1, \mathsf{GB}_2) = \textit{true} & regmode(\mathsf{GB}_2) = \textit{false} \end{array}}{\{\{\mathsf{GB}_1, \ldots\}, \{\mathsf{GB}_2, \ldots\}\} \rightarrow \{\{\mathsf{GB}_3, \{\mathsf{GB}_1, \ldots\}, \{\mathsf{GB}_2, \ldots\}\}\}}\ (\mathsf{LinkCol})$$

- $newgbu(\mathsf{GB}_1, \mathsf{GB}_2) = \mathsf{GB}_3$: a new broker is created on behalf on GB_1 and GB_2;
- $samedom(\mathsf{GB}_1, \mathsf{GB}_2) = \textit{false}$: both reside in the same administrative domain;
- $agree(\mathsf{GB}_1, \mathsf{GB}_2) = \textit{true}$: an agreement between the two brokers is signed;
- $gmode(\mathsf{GB}_1) = \textit{true}$ & $gmode(\mathsf{GB}_2) = \textit{true}$ & $regmode(\mathsf{GB}_1) = \textit{false}$ & $regmode(\mathsf{GB}_2) = \textit{false}$: the brokers are in global mode but still not registered. The side effect of this rule is to set the registration mode of both brokers to *true*.

(vi) Unlinking two colonies *in different Administrative Domains*

$$\frac{\begin{array}{rr} pendingip(\mathsf{GB}_1) = \textit{false} & pendingip(\mathsf{GB}_2) = \textit{false} \\ pendingip(\mathsf{GB}_3) = \textit{false} & gmode(\mathsf{GB}_1) = \textit{false} \\ newgbu(\mathsf{GB}_1, \mathsf{GB}_2) = \mathsf{GB}_3 & gmode(\mathsf{GB}_2) = \textit{false} \\ samedom(\mathsf{GB}_1, \mathsf{GB}_2) = \textit{false} & regmode(\mathsf{GB}_1) = \textit{true} \\ agree(\mathsf{GB}_1, \mathsf{GB}_2) = \textit{false} & regmode(\mathsf{GB}_2) = \textit{true} \end{array}}{\{\{\mathsf{GB}_3, \{\mathsf{GB}_1, \ldots\}, \{\mathsf{GB}_2, \ldots\}\}\} \rightarrow \{\{\mathsf{GB}_1, \ldots\}, \{\mathsf{GB}_2, \ldots\}\}} \;(\mathsf{UnLinkCol})$$

- $newgbu(\mathsf{GB}_1, \mathsf{GB}_2) = \mathsf{GB}_3$: a new broker is created on behalf of GB_1 and GB_2;
- $samedom(\mathsf{GB}_1, \mathsf{GB}_2) = \textit{true}$: both reside in the same administrative domain;
- $agree(\mathsf{GB}_1, \mathsf{GB}_2) = \textit{false}$: an agreement between the two brokers is withdrawn;
- $pendingip(\mathsf{GB}_1) = \textit{false}\ \&\ pendingip(\mathsf{GB}_2) = \textit{false}\ \&$ $pendingip(\mathsf{GB}_3) = \textit{false}$: the brokers $\mathsf{GB}_{1,2,3}$ have no pending services;
- $gmode(\mathsf{GB}_1) = \textit{false}\ \&\ gmode(\mathsf{GB}_2) = \textit{false}\ \&\ regmode(\mathsf{GB}_1) = \textit{true}\ \&$ $regmode(\mathsf{GB}_2) = \textit{true}$: the brokers are in local mode but still registered. The side effect of this rule is to set their registration mode to *false*.

(vii) Contextual Rules and Congruence

We add the following congruence rules for set union and set minus, and structural equivalence rules, where COM denotes communities, COL denotes colonies and $=$ denotes the set theoretical equality. All symbols can be indexed.

$$\frac{\mathsf{COM}_1 \rightarrow \mathsf{COM}_2}{\mathsf{COM}_1 \cup \mathsf{COM}_3 \rightarrow \mathsf{COM}_2 \cup \mathsf{COM}_3} \;(\mathsf{CommCup})$$

$$\frac{\mathsf{COM}_1 = \mathsf{COM}_3 \cup \mathsf{COM}_4 \quad \mathsf{COM}_3 \cap \mathsf{COM}_4 = \emptyset \quad \mathsf{COM}_3 \rightarrow \mathsf{COM}_2}{\mathsf{COM}_3 \rightarrow \mathsf{COM}_2 \setminus \mathsf{COM}_4} \;(\mathsf{CommMinus})$$

$$\frac{\mathsf{COM}_1 = \mathsf{COM}_3 \quad \mathsf{COM}_3 \rightarrow \mathsf{COM}_4 \quad \mathsf{COM}_4 = \mathsf{COM}_2}{\mathsf{COM}_1 \rightarrow \mathsf{COM}_2} \;(\mathsf{StructEq})$$

Rule (CommCup) is the usual Contextual closure of the reduction rules, while rule (CommMinus) states that a reduction can drop in its right-hand side some individuals that are not essential to the firing of the reduction itself. Let $\rightarrow^*$ be the reflexive and transitive closure of $\rightarrow$.

4 Join/Leave a Colony in a Different Administrative Domain

The acute reader has observed that the above labeled transition system *forbids* an individual to join/leave another colony whose leader resides *in a different Administrative Domain*. This is sound in order to guarantee the integrity and the security of the virtual organization induced by the Arigatoni model. Crossing safely administrative domains is an important security problem that the model must take into account. However, the situation where one individual does not receive enough help from the local colony or, worst, where it is even rejected as an individual, could be very common. In this case, it is highly desirable that the model permits a mechanism to cross boundaries of the administrative domain in order to make a service request to another colony which resides in another administrative domain. This can be done in two ways:

(i) the individual resident in an administrative domain IP_1 knows some "friends" inhabitant of the colony resident in another administrative domain IP_2 (think of the individual as a laptop connected in a hot spot of an airport, and think of the "friend" as the desktop in its own office). Then, via an explicit `ssh`, the laptop can log into the desktop and send a global request to the "mother colony". As such, the laptop works in its *local mode* while the desktop works in *global mode*. The final result will be send, via `ssh-tunneling` to the laptop.

    This mechanism of tunneling is well-known in common practice of nomadic behaviors and it does not require any *ad hoc* rewriting rules in the Arigatoni virtual organization since the connection individual-friend is done explicitly and privately;

(ii) the individual resident in an administrative domain IP_1 knows no inhabitant of the colony resident in another administrative domain IP_2, but it knows the IP address of the leader of the colony. If the leader agrees, it can arrange an `ssh-tunnel` by creating from scratch a *virtual clone* of the remote individual and by registering it in the colony on behalf of the leader of the colony. As in the previous case, the laptop can log into the desktop and send a global request to the "mother colony". As such, the laptop works in *local mode* while the clone works in *global mode*. The final result will be sent, via `ssh-tunneling` to the laptop.

    This mechanism is well-known in common practice of nomadic behaviors and is reminiscent of the *Virtual Private Network* technology (VPN) [6]. To implement this VPN-like behavior, we must add four *ad hoc* rewriting rules in the labeled transition system showed in Figure 5. For obvious lack of space those rules are not commented but left as an easy exercise to the interested reader.

5 Firing Free Riders

Again, the acute reader has observed that the original labeled transition system allows *free riders* to become members of one colony.

> *"In economics and political science, free riders are actors who consume more than their fair share of a resource, or shoulder less than a fair share of the costs of its production. The free rider problem is the question of how to prevent free*

$$\dfrac{\begin{array}{rr} discover(\mathsf{GC}_1)=\mathsf{GB} & agree(\mathsf{GB},\mathsf{GC}_1)=\textit{true} \\ samedom(\mathsf{GB},\mathsf{GC}_1)=\textit{false} & gmode(\mathsf{GC}_1)=\textit{false} \\ newgcu(\mathsf{GB},\mathsf{GC}_1)=\mathsf{GC}_2 & regmode(\mathsf{GC}_1)=\textit{false} \\ samedom(\mathsf{GB},\mathsf{GC}_2)=\textit{true} & gmode(\mathsf{GC}_2)=\textit{true} \\ accept(\mathsf{GB},\mathsf{GC}_2)=\textit{true} & regmode(\mathsf{GC}_2)=\textit{false} \end{array}}{\{\{\mathsf{GB},\ldots\},\mathsf{GC}_1\}\rightarrow\{\{\mathsf{GB},\mathsf{GC}_2,\ldots\},\mathsf{GC}_1\}}\;(\mathsf{JoinTunnelGCU})$$

$$\dfrac{\begin{array}{rr} & agree(\mathsf{GB},\mathsf{GC}_1)=\textit{false} \\ samedom(\mathsf{GB},\mathsf{GC}_1)=\textit{false} & pendingip(\mathsf{GC}_2)=\textit{false} \\ newgcu(\mathsf{GB},\mathsf{GC}_1)=\mathsf{GC}_2 & gmode(\mathsf{GC}_1,\mathsf{GC}_2)=\textit{false} \\ samedom(\mathsf{GB},\mathsf{GC}_2)=\textit{true} & regmode(\mathsf{GC}_1)=\textit{false} \\ accept(\mathsf{GB},\mathsf{GC}_2)=\textit{false} & regmode(\mathsf{GC}_2)=\textit{true} \end{array}}{\{\{\mathsf{GB},\mathsf{GC}_2,\ldots\},\mathsf{GC}_1\}\rightarrow\{\{\mathsf{GB},\ldots\},\mathsf{GC}_1\}}\;(\mathsf{LeaveTunnelGCU})$$

$$\dfrac{\begin{array}{rr} discover(\mathsf{GB}_2)=\mathsf{GB}_1 & agree(\mathsf{GB}_1,\mathsf{GB}_2)=\textit{true} \\ samedom(\mathsf{GB}_1,\mathsf{GB}_2)=\textit{false} & gmode(\mathsf{GB}_3)=\textit{true} \\ newgbu(\mathsf{GB}_1,\mathsf{GB}_2)=\mathsf{GB}_3 & regmode(\mathsf{GB}_3)=\textit{false} \\ samedom(\mathsf{GB}_1,\mathsf{GB}_3)=\textit{true} & gmode(\mathsf{GB}_2)=\textit{false} \\ accept(\mathsf{GB}_1,\mathsf{GB}_3)=\textit{true} & regmode(\mathsf{GB}_2)=\textit{false} \end{array}}{\{\{\mathsf{GB}_1,\ldots\},\{\mathsf{GB}_2,\ldots\}\}\rightarrow\{\{\mathsf{GB}_1,\{\mathsf{GB}_3\},\ldots\},\{\mathsf{GB}_2,\ldots\}\}}\;(\mathsf{JoinTunnelCol})$$

$$\dfrac{\begin{array}{rr} & agree(\mathsf{GB}_1,\mathsf{GB}_2)=\textit{false} \\ samedom(\mathsf{GB},\mathsf{GB}_2)=\textit{false} & pendingip(\mathsf{GB}_3)=\textit{false} \\ newgbu(\mathsf{GB}_1,\mathsf{GB}_2)=\mathsf{GB}_3 & gmode(\mathsf{GB}_2,\mathsf{GB}_3)=\textit{false} \\ samedom(\mathsf{GB}_1,\mathsf{GB}_3)=\textit{true} & regmode(\mathsf{GB}_2)=\textit{false} \\ accept(\mathsf{GB}_1,\mathsf{GB}_3)=\textit{false} & regmode(\mathsf{GB}_3)=\textit{true} \end{array}}{\{\{\mathsf{GB}_1,\{\mathsf{GB}_3\},\ldots\},\{\mathsf{GB}_2,\ldots\}\}\rightarrow\{\{\mathsf{GB}_1,\ldots\},\{\mathsf{GB}_2,\ldots\}\}}\;(\mathsf{LeaveTunnelCol})$$

Figure 5. Extra Reduction Rules for Service Request via Tunnelling *à la* VPN

riding from taking place, or at least limit its negative effects. Because the notion of "fairness" is a subject of controversy, free riding is usually only considered to be an economic "problem" when it leads to the non-production or under-production of a public good, and thus to Pareto inefficiency, or when it leads to the excessive use of a common property resource" [From Wikipedia].

The selfish nodes in P2P networks, called free riders, only utilize other peers resources without providing any contribution in return, have greatly jeopardized the

$$\frac{\begin{array}{cc} \mathit{pendingip}(\mathsf{GC}) = \textit{false} & \mathit{gmode}(\mathsf{GC}) = \textit{true} \\ \mathit{samedom}(\mathsf{GB}, \mathsf{GC}) = \textit{true} & \mathit{regmode}(\mathsf{GC}) = \textit{true} \\ \mathit{fairness}(\mathsf{GB}, \mathsf{GC}) \leq \epsilon & \mathit{notifiring}(\mathsf{GB}, \mathsf{GC}) \end{array}}{\{\{\mathsf{GB}, \mathsf{GC}, \ldots\}\} \to \{\{\mathsf{GB}, \ldots\}, \mathsf{GC}\}} \ (\mathsf{FireGCU})$$

$$\frac{\begin{array}{cc} \mathit{pendingip}(\mathsf{GB}_2) = \textit{false} & \mathit{gmode}(\mathsf{GB}_2) = \textit{true} \\ \mathit{samedom}(\mathsf{GB}_1, \mathsf{GB}_2) = \textit{true} & \mathit{regmode}(\mathsf{GB}_2) = \textit{true} \\ \mathit{fairness}(\mathsf{GB}_1, \mathsf{GB}_2) \leq \epsilon & \mathit{notifiring}(\mathsf{GB}_1, \mathsf{GB}_2) \end{array}}{\{\{\mathsf{GB}_1, \{\mathsf{GB}_2, \ldots\}, \ldots\}\} \to \{\{\mathsf{GB}_1, \ldots\}, \{\mathsf{GB}_2, \ldots\}\}} \ (\mathsf{FireCol})$$

Figure 6. Extra Reduction Rules for Firing *Free Riders*

fairness attribute of P2P networks. Figure 6 presents the two rules that take into account the ratio between the number of services offered and the number of services demanded by an individual. If the leader of a colony finds that an individual ratio of fairness is too small ($\leq \epsilon$ for a given ϵ), it can arbitrarily decide to fire that individual without notice. Here, the function *pendingip* also checks that the individual has no pending services to offer, or that the timeout of some promised services has expired, the latter case means that the free rider promised some services but finally did not provide any service at all (not trustful). The function *notifiring* sends a message to the free rider, notifying it that it was definitively fired from the colony.

6 Examples

In [1], a Grid scenario for Seismic Monitoring was presented. In this section we briefly recall the scenario and we present, by means of labeled transition system reductions, the evolution of the given virtual organization. The

6.1 (Re)Setting the Scenario (from [1])

John, chief engineer of the SeismicDataCorp Company, Taiwan, on board of the seismic data collector ship, has to decide on the next data collect campaign. For this he would like to process the 100 TeraBytes of seismic data that have been recorded on the mass data recorder located in the offshore data repository of the company, to be processed and then analyzed.

He has written the processing program for modeling and visualizing the seismic cube using some *parallel library* like *e.g.* `MPI/PVM`: his program can be distributed over different machines that will compute a chunk of the whole calculus.

However, the amount of computation is so big that a supercomputer ($\mathsf{GC}_{\mathsf{SCU}}$) and a cluster of PC ($\mathsf{GC}_{\mathsf{CLU}}$) has to be *rented* by the SeismicDataCorp company. John will also ask for *bandwidth* via an ISP located in Taiwan ($\mathsf{GC}_{\mathsf{ISPTW}}$) in order to get rid of any bottleneck related to the big amount of data to be transferred.

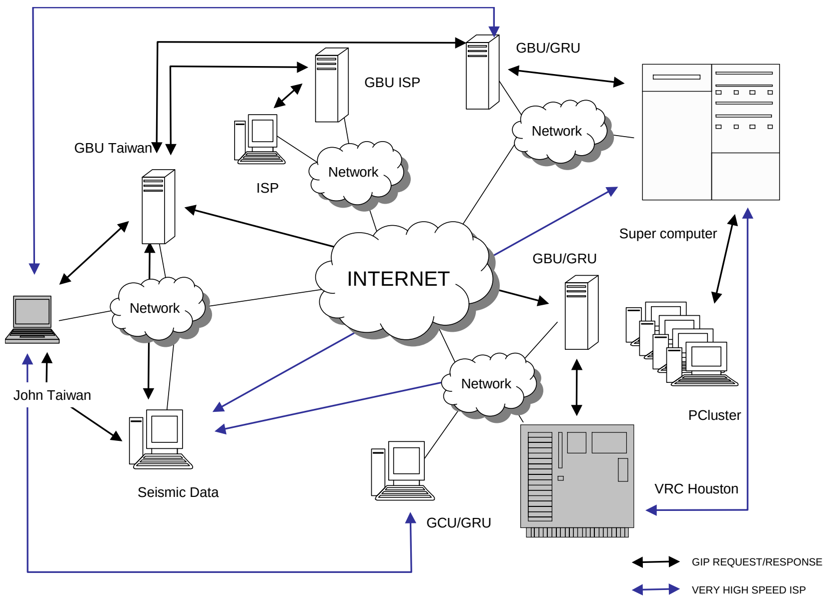

Figure 7. A Grid Scenario for Seismic Monitoring

Aftermath, the processed data should be analyzed using a *Virtual Reality Center*, VRC (GC_{VCRCPU}) based in Houston, U.S.A. by a specialist team and the resulting recommendations for the next data ($GC_{VRCSPEC}$) collect campaign have to be sent to John. Hence one would like the following scenario to happen:

- John logs with its laptop (GC_{John}) to the Arigatoni overlay network in a given colony in Taiwan, and sends a quite complicated service request in order for the data to be processed using his own code. Usually, the GB leader of the colony will receive and process the request;
- If the resource discovery performed by the GB succeeds, *i.e.* a supercomputer, a cluster and an ISP are found, then the data are transferred at a very high speed and processed;
- John will order to the GC_{SDTW} containing the seismic data to dispatch suitable chunks of data to the supercomputer and the cluster designated by the GB to perform some pieces of computation;
- John will assign to the supercomputer unit the task of collecting all intermediate results in order to compute the final result (*i.e.* it will play the role of *Maestro di Orchestra*);
- The processed data are then sent from the supercomputer, via the high speed ISP to the Houston center for being visualized and analyzed;
- Finally, the specialist team's recommendations have to be sent to John's laptop.

This scenario is pictorially presented in Figure 7.

6.2 *Formalizing the Scenario*

The initial community (the primitive *Soup*) will be composed of the following elements:

$$\begin{aligned}\mathsf{COM}_{\mathsf{Soup}} \triangleq \{&\{\mathsf{GB}_{\mathsf{SDTW}}\}, \mathsf{GC}_{\mathsf{SDTW}}, \{\mathsf{GB}_{\mathsf{ISPTW}}\}, \mathsf{GC}_{\mathsf{ISPTW}}, \{\mathsf{GB}_{\mathsf{CPU}}\},\\ &\mathsf{GC}_{\mathsf{SCU}}, \mathsf{GC}_{\mathsf{CLU}}, \{\mathsf{GB}_{\mathsf{VRC}}\}, \mathsf{GC}_{\mathsf{VRCPU}}, \mathsf{GC}_{\mathsf{VRCSPEC}}\}\end{aligned}$$

By applying five times the reduction rule $(\mathsf{JoinGCU})$ we obtain the new community:

$$\begin{aligned}\mathsf{COM}_1 \triangleq \{&\{\mathsf{GB}_{\mathsf{SDTW}}, \mathsf{GC}_{\mathsf{SDTW}}\}, \{\mathsf{GB}_{\mathsf{ISPTW}}, \mathsf{GC}_{\mathsf{ISPTW}}\}, \{\mathsf{GB}_{\mathsf{CPU}}, \mathsf{GC}_{\mathsf{SCU}}, \mathsf{GC}_{\mathsf{CLU}}\},\\ &\{\mathsf{GB}_{\mathsf{VRC}}, \mathsf{GC}_{\mathsf{VRCPU}}, \mathsf{GC}_{\mathsf{VRCSPEC}}\}\}\end{aligned}$$

and $\mathsf{COM}_{\mathsf{Soup}} \to^5 \mathsf{COM}_1$. Then by applying the reduction rule $(\mathsf{CommCup})$ we see John's laptop appear in the new community, $\mathsf{COM}_2 \triangleq \mathsf{COM}_1 \cup \{\mathsf{GC}_{\mathsf{John}}\}$:

$$\begin{aligned}\mathsf{COM}_2 \triangleq \{&\mathsf{GC}_{\mathsf{John}}, \{\mathsf{GB}_{\mathsf{SDTW}}, \mathsf{GC}_{\mathsf{SDTW}}\}, \{\mathsf{GB}_{\mathsf{ISPTW}}, \mathsf{GC}_{\mathsf{ISPTW}}\},\\ &\{\mathsf{GB}_{\mathsf{CPU}}, \mathsf{GC}_{\mathsf{SCU}}, \mathsf{GC}_{\mathsf{CLU}}\}, \{\mathsf{GB}_{\mathsf{VRC}}, \mathsf{GC}_{\mathsf{VRCPU}}, \mathsf{GC}_{\mathsf{VRCSPEC}}\}\}\end{aligned}$$

By applying again $(\mathsf{JoinGCU})$ we obtain the new community:

$$\begin{aligned}\mathsf{COM}_3 \triangleq \{&\{\mathsf{GB}_{\mathsf{SDTW}}, \mathsf{GC}_{\mathsf{SDTW}}, \mathsf{GC}_{\mathsf{John}}\}, \{\mathsf{GB}_{\mathsf{ISPTW}}, \mathsf{GC}_{\mathsf{ISPTW}}\},\\ &\{\mathsf{GB}_{\mathsf{CPU}}, \mathsf{GC}_{\mathsf{SCU}}, \mathsf{GC}_{\mathsf{CLU}}\}, \{\mathsf{GB}_{\mathsf{VRC}}, \mathsf{GC}_{\mathsf{VRCPU}}, \mathsf{GC}_{\mathsf{VRCSPEC}}\}\}\end{aligned}$$

Now, if the community whose leader is $\mathsf{GB}_{\mathsf{SDTW}}$ agrees to join the colony whose leader is $\mathsf{GB}_{\mathsf{ISPTW}}$ (both are supposed to live in the same administrative domain), by applying rule $(\mathsf{JoinCol})$, we obtain the new community:

$$\begin{aligned}\mathsf{COM}_4 \triangleq \{&\{\mathsf{GB}_{\mathsf{ISPTW}}, \mathsf{GC}_{\mathsf{ISPTW}}, \{\mathsf{GB}_{\mathsf{SDTW}}, \mathsf{GC}_{\mathsf{SDTW}}, \mathsf{GC}_{\mathsf{John}}\}\},\\ &\{\mathsf{GB}_{\mathsf{CPU}}, \mathsf{GC}_{\mathsf{SCU}}, \mathsf{GC}_{\mathsf{CLU}}\}, \{\mathsf{GB}_{\mathsf{VRC}}, \mathsf{GC}_{\mathsf{VRCPU}}, \mathsf{GC}_{\mathsf{VRCSPEC}}\}\}\end{aligned}$$

The colony in Taiwan and the colony whose leader is $\mathsf{GB}_{\mathsf{CPU}}$ (they are supposed to live in different administrative domain) sign an "agreement", by applying rule $(\mathsf{LinkCol})$, so giving the new community:

$$\begin{aligned}\mathsf{COM}_5 \triangleq \{\ &\{\mathsf{GB}_{\mathsf{ISP\&CPU}}, \{\mathsf{GB}_{\mathsf{ISPTW}}, \mathsf{GC}_{\mathsf{ISPTW}}, \{\mathsf{GB}_{\mathsf{SDTW}}, \mathsf{GC}_{\mathsf{SDTW}}, \mathsf{GC}_{\mathsf{John}}\}\},\\ &\{\mathsf{GB}_{\mathsf{CPU}}, \mathsf{GC}_{\mathsf{SCU}}, \mathsf{GC}_{\mathsf{CLU}}\}\}, \{\mathsf{GB}_{\mathsf{VRC}}, \mathsf{GC}_{\mathsf{VRCPU}}, \mathsf{GC}_{\mathsf{VRCSPEC}}\}\}\end{aligned}$$

Finally, the colony containing John's laptop is ready to receive *John's huge Service Request*, and, hopefully for John, the request will be accepted and performed ... It is now time for John to come back home and the community COM_5 could then (but this is not mandatory) disintegrate. By applying the "dual" reduction rules $(\mathsf{LeaveGCU})$, $(\mathsf{LeaveCol})$, and $(\mathsf{UnLinkCol})$ plus the congruence rules $(\mathsf{CommCup})$ and $(\mathsf{CommMinus})$, we come back to the initial soup, *i.e.* $\mathsf{COL}_5 \to^* \mathsf{COM}_{\mathsf{Soup}}$.

7 Properties

In this section we prove that our process algebra is able to model the virtual organization induced by an $\mathsf{Arigatoni}$ overlay network. Contextual equivalence [4] is the standard way of saying that two communities have *the same behavior* (they are equivalent) if and only if, whenever they are merged inside an arbitrary community, they admit the same elementary observations. In our setting and as usual in process algebras, contextual equivalence is formulated in terms of observing the presence of top-level colonies, as in the next definition.

Definition 7.1 [Colony Exhibition and Contextual Equivalence]

(i) a community COM must exhibit a colony COL, write $\mathsf{COM} \downarrow_{\mathsf{must}} \mathsf{COL}$, if COL is a community containing a top-level colony COL, *i.e.*

$$\mathsf{COM} \downarrow_{\mathsf{must}} \mathsf{COL} \triangleq \mathsf{COM} = \{\ldots, \mathsf{COL}, \ldots\}$$

(ii) a community COM may exhibit a colony COL, write $\mathsf{COM} \downarrow_{\mathsf{may}} \mathsf{COL}$, if after a number of reductions, COL is a community containing a top-level colony COL, *i.e.*

$$\mathsf{COM} \downarrow_{\mathsf{may}} \mathsf{COL} \triangleq \mathsf{COM} \rightarrow^* \mathsf{COM}' \text{ and } \mathsf{COM}' = \{\ldots, \mathsf{COL}, \ldots\}$$

(iii) let the context $\mathsf{C}[\cdot]$ be a community containing zero or more holes, and for any community COM let $\mathsf{C}[\mathsf{COM}]$ be the community obtained by filling each hole in $\mathsf{C}[\cdot]$ with a copy of COM. The contextual equivalence between community, write $\mathsf{COM} \simeq \mathsf{COM}'$, is defined as

$$\begin{aligned} \mathsf{COM} \simeq \mathsf{COM}' \triangleq\ & \text{for all } \mathsf{COL} \text{ and } \mathsf{C}[\cdot] \text{ we have} \\ & \mathsf{C}[\mathsf{COM}] \downarrow_{\mathsf{may}} \mathsf{COL} \Leftrightarrow \mathsf{C}[\mathsf{COM}'] \downarrow_{\mathsf{may}} \mathsf{COL} \end{aligned}$$

(iv) let $\mathsf{COM} \rightarrow^* \simeq \mathsf{COM}'$ if there exists COM'' such that $\mathsf{COM} \rightarrow^* \mathsf{COM}''$ and $\mathsf{COM}'' \simeq \mathsf{COM}'$.

Let $\mathcal{COM}$ be the set of communities generated by the BNF syntax.

Theorem 7.2 (Closure Under Reduction)

(i) *If* $\mathsf{COM} \in \mathcal{COM}$*, and* $\mathsf{COM} \rightarrow^* \mathsf{COM}'$*, then* $\mathsf{COM} \in \mathcal{COM}$*;*

(ii) *If* $\mathsf{COM} \simeq \mathsf{COM}'$*, then* $\mathsf{COM}, \mathsf{COM}' \in \mathcal{COM}$*;*

(iii) *If* $\mathsf{COM} \rightarrow^* \simeq \mathsf{COM}'$*, then* $\mathsf{COM}, \mathsf{COM}' \in \mathcal{COM}$

Proof

1) By observing the reduction rules of the labeled transition system, one can verify that if the left-hand side belongs to $\mathcal{COM}$*, then it is also the case for the right-hand side. The final result can be obtained by induction on the number of reduction.*

2,3) By point 1) using Definition 7.1.

□

Theorem 7.3 (Inversion)

(i) *If* $\mathsf{COM} \rightarrow_{(\mathsf{JoinGCU/COL})} \mathsf{COM}'$ *on the individual* (GC *or* COL), *and* $\mathsf{COM}' \rightarrow_{(\mathsf{LeaveGCU/COL})} \mathsf{COM}''$ *on the same individual, then* $\mathsf{COM} = \mathsf{COM}''$;

(ii) *If* $\mathsf{COM} \rightarrow_{(\mathsf{LinkCol})} \mathsf{COM}'$ *on two colonies, and* $\mathsf{COM}' \rightarrow_{(\mathsf{UnLinkCOL})} \mathsf{COM}''$ *on the same colonies, then* $\mathsf{COM} = \mathsf{COM}''$.

Proof *By observing the reduction rules, one can observe that the right-hand side of the reduction rules* (JoinGCU), (JoinCOL), *and* (LinkCOL) *corresponds to the left-hand side of the dual reduction rules* (LeaveGCU), (LeaveCOL), *and* (UnLinkCol), *and conversely the left-hand side of the reduction rules* (JoinGCU), (JoinCOL), *and* (LinkCOL) *corresponds to the right-hand side of the dual reduction rules* (LeaveGCU), (LeaveCOL), *and* (UnLinkCol). *Applying one rule after the other clearly corresponds to an identity operation.* □

We conclude this section by a conjecture that links our formal presentation of the VIP protocol with the actual pseudo-implementation of the protocol, as described in Figures 3 and 4.

Conjecture 7.4 (Adequacy of the labeled transition system w.r.t. the pseudocode) *The labeled reduction system is adequate with the pseudocode of the* GB *and of the* GC *shown in Figure 3 and 4.*

Proof *(Sketch) Observe that the red parts of the pseudocode of the* GC *concerning the set and unset of the variables* $globalmode/regmode$ *leads to the firing of the two rules* (JoinGCU) *and* (LeaveGCU). *Moreover, the red parts of the pseudocode of the* GB *concerning the set and unset of the variables* $globalmode/regmode$ *leads to to the firing of the two rules* (JoinGCU) *and* (LeaveGCU). *The last two rules of the transition systems, namely* (LinkCol) *and* (UnLinkCol) *are encapsulated (hence hidden) in the function calls* `Update(Colony,Metadata)`. □

As suggested by one referee, further work will focus on study bisimulations for reasoning about contextual equivalence of programs. This would greatly benefit in the field of overlay networks where, quite often, protocol are defined *manu militari* via implementations and evaluated via experimental simulations (as in the next section).

8 Experimental Evaluation

In this section, we provide results from experimental evaluation. We have conducted simulations using large numbers of units and service requests. In this paper, we specifically focus on the effect of individuals disconnections on the average service acceptation ratio.

More precisely, we have implemented reduction rules (JoinGCU), (LeaveGCU), (JoinCol), and (LeaveCol), that represent the "core" rewriting set to simulate the dynamic behavior in the Arigatoni overlay network. We expect to implement the full set of rewriting rules defining the operational semantics soon.

8.1 Simulation Setup

We have generated a network topology using the transit-stub model of the Georgia Tech Internetwork Topology Models package [7], on top of which we added the Arigatoni overlay network. The resulting network topology, shown in Figure 8, contains 103 GBs. GB$_2$ (highlighted with a square in Figure 8) was chosen as the root of the topology. We considered a finite set of resources R$_1 \cdots$ R$_r$ of variable

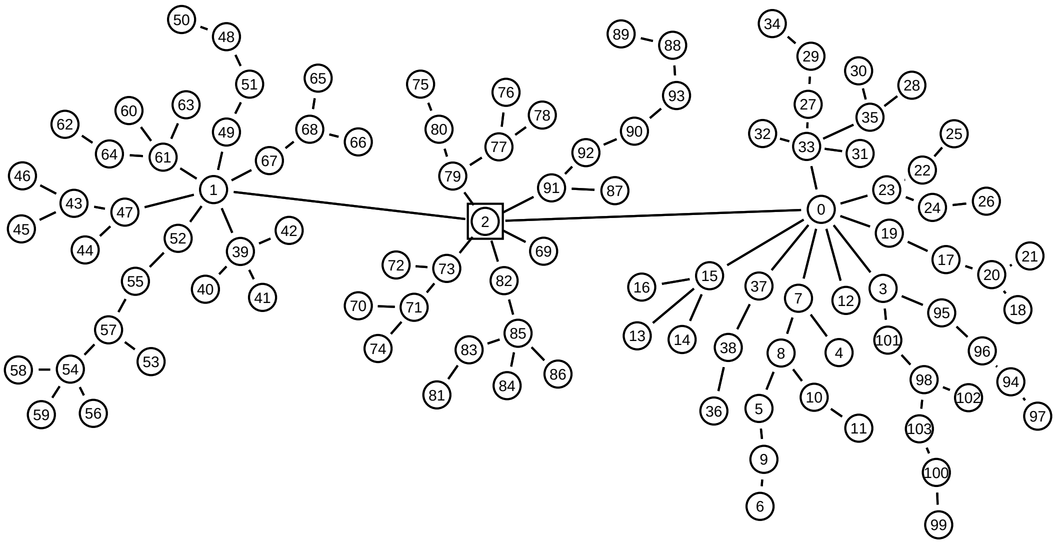

Figure 8. Simulated network topology with 103 GBs

size r, and represented a service by a direct mapping to a resource. In other words, a service expresses the conditional presence of a single resource. We have a set of r services $\{$S$_1 \cdots$ S$_r\}$, where service S$_i$ expresses the conditional presence of resource R$_i$. A GC declaring service S$_i$ means that it can provide resource R$_i$. This model, while quite simple, is still generic enough, and is sufficient for the main purpose of our experiments, which is to study the impact of individuals disconnections on the average service acceptation ratio. Results are illustrated in Figure 9.

To simulate GC load, we attached 50 GCs to each GB; we then randomly added each service S$_i$ with probability ρ at each GC and had it registered via the registration service of Arigatoni. The routing tables of the GBs were updated starting at the initial GB and ending at the root of the topology, GB$_2$.

We then issued n service requests at GCs chosen uniformly at random. Each request contained one service also chosen uniformly at random. Each service request was then handled by the Resource Discovery mechanism of Arigatoni (described in [3]). We used a service acceptation probability of $\alpha = 75\%$, which corresponds to the probability that a GC that receives a service request *and* that declared itself as a potential individual for that service (*i.e.* that registered it), accepts to serve it.

Upon completion of the n requests, we computed the average service acceptation ratio as follows. For each GC, we computed the local acceptation ratio as the number of service requests that yielded a positive response (*i.e.* the system found at least one individual), over the number of service requests issued at that GC. We then computed the average acceptation ratio as the average value over the number of GCs (that issued at least one service request).

To study the impact of GBs disconnections (*i.e.*, rewriting rules $(\mathsf{JoinCol})$ and $(\mathsf{LeaveCol})$), we used a disconnection probability variable δ that indicates a fraction of disconnected individuals ($\delta = 0\%$ means all individuals are connected, while $\delta = 100\%$ means all individuals are disconnected). We then repeated the same experiment when δ of the GBs population, chosen uniformly at random, have been disconnected from their leader. When a subcolony has been disconnected from its GB leader, it continues to operate in *standalone* mode, *i.e.* with its local GB leader as the current broker. Therefore, the services offered by the other colonies are unavailable inside, while services offered by the colony itself are not available outside. For each value of $\delta \in [0 \cdots 100]\%$, we repeated the same experiment 10 times, and measured the average value of the acceptation ratio. In each of the 10 runs, the disconnected GBs were chosen uniformly at random, *independently* of the previous runs (*i.e.*, with a different random seed). We then computed the standard deviation of the average service acceptation ratio (over the 10 values).

Starting from the fully connected topology COM_1 of Figure 8, the rationale of the simulation corresponds to applying a number of $(\mathsf{JoinCol})$ rewriting rules to have some subcolonies join the colony, and then applying a number of $(\mathsf{LeaveCol})$ rewriting rules to have some other subcolonies leave the colony, and then performing the experiment 10 times.

$$\mathsf{COM}_i \rightarrow^*_{(\mathsf{JoinCol})} \mathsf{COM}'_{i+1} \rightarrow^*_{(\mathsf{LeaveCol})} \mathsf{COM}_{i+1} \qquad i = 1 \dots 10$$

We also studied the effect of GCs disconnections (rewriting rules $(\mathsf{JoinGCU})$ and $(\mathsf{LeaveGCU})$), by repeating the same experiment when δ of the GCs population have been disconnected from their leader. Also in this case, a disconnected GC continues to work in standalone mode using only their own resources.

As for the GB case, we have

$$\mathsf{COM}_i \rightarrow^*_{(\mathsf{JoinGCU})} \mathsf{COM}'_{i+1} \rightarrow^*_{(\mathsf{LeaveGCU})} \mathsf{COM}_{i+1} \qquad i = 1 \dots 10$$

The resource discovery protocol taking into account virtual intermittence of individuals was implemented in C++ and compiled using GNU C++ version 2.95.3. Experiments were conducted on a 3.0 Ghz Intel Pentium machine with 2 GB of main memory running Linux 2.4.28. The different experimental parameters are summarized in Table 1. The service availability ratio, ρ, was fixed to a value of 12%, which yields an average service acceptation ratio of almost 100% with no subcolonies disconnections. Figure 9(a) shows that the average service acceptation ratio decreases exponentially with the number of subcolonies (*i.e.*, GBs) disconnections. This is not surprising, since when a subcolony has been disconnected,

Parameter	**Description**	**Value**
K	Number of GBUs	103
r	Size of services pool	128
ρ	Service availability	12%
α	Service acceptation probability	75%
n	Number of service requests issued	50000
δ	Fraction of disconnected individuals	$[0 \cdots 100]\%$

Table 1
Parameters of the experiments

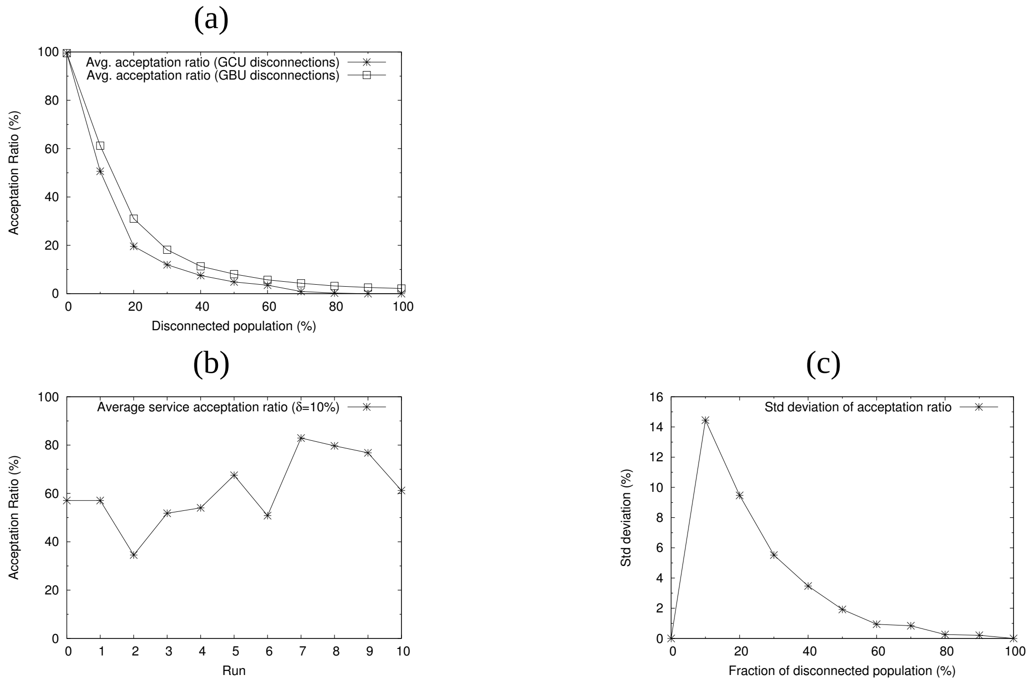

Figure 9. (a) Average service acceptation ratio w.r.t. fraction of disconnected population. (b) Average service acceptation ratio for the different runs of the value $\delta = 10\%$. (c) Standard deviation of the service acceptation ratio w.r.t. fraction of disconnected population.

all the services offered by the other colonies are unavailable. Conversely, all the services offered by the subcolony are unavailable for the other colonies. Note that when all subcolonies (GB) have been disconnected ($\delta = 100\%$), then the average service acceptation ratio is not null. Indeed, the local colony of a GB (*i.e.*, the GCs directly connected to the GB) remains operational, *i.e.*, the services offered by a GC are available for the other GCs of the same colony.

We observe that GC disconnections have more impact on the average service acceptation ratio than GB disconnections. This is due to the fact that when a GC is disconnected, all the services that it provided are unavailable for the entire system and, conversely, all the services provided by the system are unavailable for it. As expected, for a value of $\delta = 100\%$, the average acceptation ratio is 0, as no service at all is unavailable.

Figure 9(a) shows the different values of the average service acceptation ratio obtained for a value of $\delta = 10\%$ of the fraction of disconnected population. As previously explained, for each run, we have chosen 10 GBs ($\sim 10\%$ of 103) uniformly at random, and *independently* of the previous runs, *i.e.*, with a different random seed. In other words, the disconnected subcolonies are different in each run. Figure 9(b) shows that subcolonies disconnections can have a very different impact on the acceptation ratio. In fact, "low-level" subcolonies disconnections have a dramatic impact whereas "high-level" subcolonies disconnections have a very limited, local impact. Figure 9(c) shows that, unsurprisingly, the level of the disconnected subcolony has less impact on the service acceptation ratio for higher values of δ.

Acknowledgment

The authors ack Aeolus FP6-2004-IST-FET Proactive, the French grant ACI Modulogic, and the anonymous referees for their useful comments and suggestions.

References

[1] D. Benza, M. Cosnard, L. Liquori, and M. Vesin. Arigatoni: A Simple Programmable Overlay Network. In *Proc. of John Vincent Atanasoff International Symposium on Modern Computing*. IEEE, 2006.

[2] L. Cardelli and A. D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[3] R. Chand, M. Cosnard, and L. Liquori. Resource Discovery in the Arigatoni Overlay Network. In *I2CS: International Workshop on Innovative Internet Community Systems*, volume LNCS. Springer, 2006. To appear. Also available as RR INRIA 5928.

[4] J. H. Morris. *Lambda-calculus models of programming languages*. PhD thesis, MIT, 1968.

[5] V. Sassone. Global Computing II: A New FET Program for FP6. Talk, Bruxelles, 4/6/04.

[6] Virtual Private Network Consortium. Virtual Private Network Home Page. `http://www.vpnc.org/`.

[7] E.W. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proc. of INFOCOM*, 1996.