Arigatoni: A Simple Programmable Overlay Network

Didier Benza, Michel Cosnard, Luigi Liquori, Marc Vesin

Arigatoni: A Simple Programmable Overlay Network

Didier Benza Michel Cosnard Luigi Liquori Marc Vesin
INRIA, France
[Didier.Benza,Michel.Cosnard,Luigi.Liquori,Marc.Vesin]@inria.fr

Abstract

We design a lightweight Overlay Network, called Arigatoni*, that is suitable to deploy the* Global Computing Paradigm *over the Internet. Communications over the behavioral units of the model are performed by a simple communication protocol. Basic Global Computers can communicate by first registering to a brokering service and then by mutually asking and offering services, in a way that is reminiscent to Rapoport's* "tit-for-tat" *strategy of cooperation based on reciprocity. In the model, resources are encapsulated in the administrative domain in which they reside, and requests for resources located in another administrative domain traverse a broker-2-broker negotiation using classical* PKI *mechanisms. The model is suitable to fit with various global scenarios from classical* P2P *applications, like file sharing, or band-sharing, to more sophisticated* Grid *applications, like remote and distributed big (and small) computations, to possible, futuristic real migrating computations. Indeed, our model fits some of the objectives suggested by the* CoreGrid *Network of Excellence, as described in Schwiegelshohn et al. [?].*

1. Introduction

This paper presents the first light-weight overlay network called Arigatoni[1] that 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 owned by agents (called *Global Computers*) acting with partial knowledge and no central coordination. The paradigm provides uniform services with variable guarantees. Aggregating many Global Computers sharing similar or different resources leads to a *Virtual Organization*, sometimes called *Overlay Computer*. Finally, organizing many Overlay Computers, using, *e.g.* tree- or graph-based topology leads to an *Overlay Network*, *i.e.* the possibility of programming a *collaborative Global Internet* over the *plain Internet*.

The main challenge in this research field is how single resources, offered by the Global/Overlay Computers are discovered. The process is often called *Resource Discovery*: it requires an *up-to-date* information about widely-distributed resources. This is a challenging problem for large distributed systems when taking into account the continuously changing state of resources offered by Global/Overlay Computers and the possibility of tolerating intermittent participation and dynamically changing status/availability of the latter.

Entities in Arigatoni are organized in *Colonies*. A colony is a simple virtual organization composed by exactly one *Leader*, offering some broker-like services, and some set of *Individuals*. Individuals are Global Computers (think it as an *Amoeba*), or subcolonies (think it as a *Protozoa*). Global Computers communicate by first registering to the colony and then by mutually asking and offering services. The leader, called *Global Broker* analyzes service requests/responses, coming from its own colony or arriving from a surrounding colony, and routes requests/responses to other individuals. After this discovery phase, individuals get in touch with each other without any further intervention from the system, in a P2P fashion.

Symmetrically, the leader of a colony can arbitrarily unregister an individual from its colony, *e.g.*, because of its bad performance when dealing with some requests, or because of its high number of "embarrassing" requests for the colony. This mechanism/strategy reminiscent of the Roman *"do ut des"*, is nowadays called, in Game Theory, *"tit-for-tat"* [?]. This strategy is commonly used in economics, social sciences, and it has been implemented by a computer program as a winning strategy in a chess-play challenge against humans (see also the well known *prisoner dilemma*). In computer science, the tit-for-tat strategy is the main principle of Bittorrent P2P protocol [?]. Once a Global Computer has issued a request for some services, the

[1] The Arigatoni model, protocol and middleware, is copyrighted by Luigi Liquori (INRIA) under the CECIL License.

system finds some Global Computers (or, recursively, some subcolonies) that can offer the resources needed, and communicates their identities to the (client) Global Computer as soon as they are found.

The model also offers some mechanisms to dynamically adapt to *dynamic topology changes* of the Overlay Network, by allowing an individual (Global Computer or subcolony) to log/delog in/from a colony. This essentially means that the process of routing request/responses may lead to failure, because some individuals delogged or because they are temporarily unavailable (recall that Individuals are not *slaves*) [**?**]. This may also lead to temporarily *denials of service* or, more drastically, to the complete delogging of an individual from a given colony in the case where the former does not provide enough services to the latter.

Indeed, dealing only with Resource Discovery has one important advantage: the complete generality and independence of any given requested resource. Arigatoni can fit with various scenarios in the Global Computing arena, 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 GCU, 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 main ingredients of Arigatoni are one protocol, the *Global Internet Protocol,* GIP, and three main units:
• A *Global Computer Unit,* GCU, *i.e.* the basic peer of the Global Computer paradigm; typically it is a small device, like a PDA, a laptop or a PC, connected via IP.
• A *Global Broker Unit,* GBU, is the basic unit devoted to register and unregister GCUs, to receive service queries from client GCUs, to contact potential servants GCUs, to negotiate with the latter the given services, to trust clients and servers, and to send all the informations useful to allow the client GCU, and the servants GCUs to be able to communicate. Every GCU can register to *only one* GBU, so that every GBU controls a *colony* of collaborating Global Computers. Hence, communication intra-colony is initiated via only one GBU, while communication inter-colonies is initiated through a chain of GBU-2-GBU message exchanges. In both cases, when a client GCU receives an acknowledgment for a request service (with related trust certificate) from the leader GBU, then the client enjoys the service directly from the servant(s) GCU, *i.e.* without a further mediation of the GBU itself.
• A *Global Router Unit,* GRU, is a simple basic unit that is devoted to send and receive packets of the Global Internet Protocol GIP and to forward the payload to the units which is connected with this router. Every GCU and every GBU have one personal GRU. The connection between router and peer is ensured via suitable API.

Effective use of computational grids via Overlay Networks requires *up-to-date* information about widely-distributed resources. This is a challenging problem for very large distributed systems particularly taking into account the continuously changing state of the 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 overlay network is, by construction, independent from any given resource request. We could envisage at least the following scenarios to be completely full-fitted in our model (list not exhaustive)
• Ask for computational power (*i.e.* the Grid).
• Ask for memory space.
• Ask for bandwidth (*i.e.* VoIP).
• Ask for file retrieving (*i.e.* P2P).
• Ask for web service (*i.e.* Google).
• Ask for a computation migration (*i.e.* transfer one partial run in another GCU saving the partial results, as in a truly mobile ubiquitous computations).
• Ask for a *Human Computer Interaction* ...

Our paper tries to fill some of the objectives fixed in the seminal paper of [?], where the requirements and the resource management for future generation Grids are discussed. More generally, Arigatoni is *parametric* in a given application, or *universal* in the sense of Universal Turing Machine, or *generic* as the Von Neumann Computer Model. Summarizing, the original contributions of the paper are:
• A simple distributed communication model that is suitable to make Resource Discovery transparent.
• A Global Internet Protocol that allows Global Computers to negotiate resources.
• A *complete independence* of the classical scenarios of the arena, *i.e.* Grid, file/band sharing, web services, etc. This domain independence is a key feature of the model and of the protocol, since it allows the Overlay Network to be programmable.

We hope that Arigatoni could represent a little step toward a natural integration of different scenarios under the common paradigm of Global Computing.

Road Map. The paper is structured as follows: Section 2 describes in an high level fashion, the Arigatoni Overlay Network and its functional units. Section 3 presents one possible semantic of the three units (via one "reference" implementation). Section 4 describes

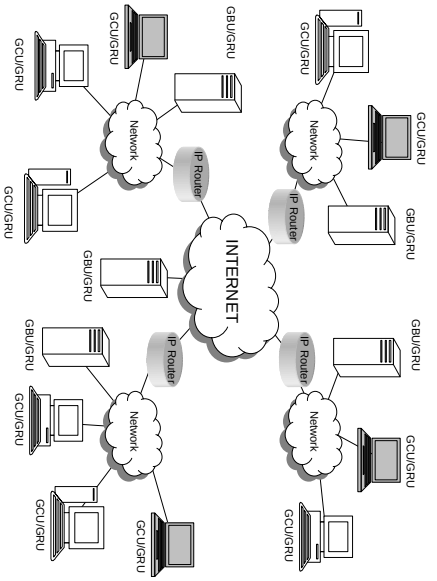

Figure 1. ArigatoNet

the protocol used by all the units to communicate. Section 5 puts Arigatoni@work in a Grid arena, while Section 6 concludes. The companion papers [**?**, ?] present an analysis of Resource Discovery, scalability and experimental issues, simulations and performance evaluation, and a semantic of the Virtual Organization.

2. Arigatoni Units: Informal Description

The Global Computer Unit (GCU) can be, *e.g.* a cheap computer device composed by a small RAM-ROM-HD memory capacity, a modest CPU, a ≥ 20 keystrokes keyboard, a ≥ 1.5 inch screen, an IP connection, an USB port, and very few programs installed inside (one simple editor, one or two compilers, a `mail` client, a mini browser, a GSM module, etc). Of course a GCU can be a big computer or a PC-cluster. The operating systems installed in the GCU is not important. The computer should be able to work in *Standalone Local Mode* for all the tasks that it could do locally or in *Global Mode*, by first registering itself in Arigatoni, and then by making a global request to the Overlay Network. Figure **??** shows the Arigatoni Overlay Network. The task of a GCU are:

- Discover, upon the physical arrival of the GCU in a new colony, the address of a GBU (*colony leader*).
- Register/Unregister on the GBU, leader of the colony.
- Request some services to its GBU, and respond to some requests from the GBU.
- Connect directly with the servant(s) GCU in a P2P fashion, and offer/receive the service.

It is worth noticing that a GCU can also be a resource provider. Hence, a GCU can also be a supercomputer, a high performance parallel cluster, a large database server, an high performance visualizer (*e.g.* connected to a virtual reality center), or any particular resource provider, that is linked to Internet. This symmetry is another key feature of Arigatoni. We assume that every GCU comes with its proper PKI certificate.

The Global Broker Unit (GBU) is devoted to:

- Discover, the address of another *super* GBU, representing the *superleader* of the *supercolony*, where the GBU's colony is embedded. We assume that every GBU comes with its proper PKI certificate. The policy to accept or refuse the registration of an individual with a different PKI are left open to the level of security requested by the colony.
- Register/Unregister the proper colony on the *leader* GBU which manages the supercolony.
- Register/Unregister clients and servants GCU in its local base of Global Computers. We assume by definition that every GCU can register to *at most* one GBU.
- Acknowledge the request of service of the client GCU.
- Discover the resources that satisfy the GCU's request in its local base (local colony) of GCU.
- Delegate the request to a GBU leader of another colony.
- Perform a combination of the above two actions.
- Deal with all PKI intra- and inter-colony policies.
- Notify, to the client GCU or to a delegating GBU, the servant(s) GCUs that have accepted to serve its request, or notify a *failure* of the request.

Every GCU in the colony sends its request to the GBU which is the leader of the colony. There are different scenarios concerning service discovery, namely:

- The broker finds all the resource(s) needed to satisfy the requested services of the GCU client locally in the intranet. Then, it will send all the information necessary to make the GCU client able to communicate with the GCU servants. This notification will be encoded using the GIP protocol. Then, the GCU client will directly talk with GCU servant(s).
- The broker did not find all the resource(s) in the local intranet. In this case it will forward and delegate the request to another broker. To do this, it must first

register the whole colony to another supercolony.
• A combination of steps 1 + 2 could be envisaged depending on the capability of the GBU to combine resources that it manages and resources coming from a delegate GBU.
• After a fixed *timeout period*, or when all delegate GBUs failed to satisfy the delegated request, the broker will notify to the GCU client the *refusal of service* requested by the GCU client.

The Global Router Unit (GRU) 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 GIP protocol. The GRU can be implemented as a small daemon which runs on the same device as a GCU or a GBU, or as a shared library dynamically linked with a GCU or a GBU. The GRU is devoted to the following tasks:
• Upon the initial startup of a GCU it helps to register the unit with one GBU.
• It checks the well-formedness and forwards GIP packets across the Arigatoni toward their destinations. GIP packets encode the requests of a GCU or a GBU.
• Upon the initial startup of a GBU it helps the unit with several other GBUs that it knows or discovers.

Resource Discovery. The are mostly three mechanisms of *Resource Discovery* in Arigatoni, namely:
• The process of a GBU to find and negotiate resources to serve a GCU's request in its own colony.
• The process of a GCU to discover a GBU, upon physical insertion in a colony.
• The process of a GBU to discover other *friend* GBU, upon physical insertion in the Overlay Network.

3. Arigatoni's Units: Formal Description

We present a prototype implementation of the three units of Arigatoni. As any pseudocode, this encoding does not bring into light all the details which are usually swept under the carpet. We try to get the encoding as clean and compact as possible, by abstracting as much as possible on all "bureaucracy" concerning synchronization between processes.

In what follows, everything in italic denotes a constant; in particular, *MyId* denotes the name of the current unit (like, *e.g.* `this` in object-oriented languages), and *MyGRU* denotes the name of the Global Router which is uniquely attached, via API to *MyId*, and *MyPKI* denotes my security certificate, and *MyRes* denotes the set of resources that the individual can offer to the community. Those values are packaged in a record (the *identity card*) called *MyCard*. The **inparallel**...**with**...**endinparallel** control structure allows two or many processes to be executed concurrently and independently [**?**, **?**].

The GCU's Semantics is described in the pseudo-code in Figure **??**. The key functions of the algorithm are explained below. It is composed by four processes:
• **Un/Registering**: implements the un/registration of a GCU to a GBU leader of a given colony.
• **Basic shell:** read-eval-print loop. In the case of a local failure of a request, if the GCU is working in global mode, then the request is forwarded to the GBU leader of the colony.
• **Global GBU Listening:** listens for any communication (service request/response) from the GBU.
• **Global GCU Listening:** deals with the (P2P like) interaction between GCUs. This interaction takes place after a clear phase of negotiation with the GBU.

We let the following variables shared by all processes, via classical semaphores *à la* Dijkstra:
• `GBU` holds all the security and network informations of the leader of the colony.
• `GlobalMode` is *true* iff the GCU works in global mode.
• `RegMode` is *true* if and only if the GCU has been registered in a given colony. Until registered, the GCU must keep the dialog with the GBU.

A short explanation of the GCU pseudocode follows.
• `Discover(`*MyCard*`)` discovers the GBU leader of the colony, where the GCU is going to connect.
• `ServiceReg(`*MyCard*`,GBU,LOGIN)` tries to register the GCU to GBU on the local colony. The registration can fail depending of different parameters (like the fact that the PKI is not trustful, or that the GCU will offer insufficient resources to the colony, etc.); this function will set `RegMode` to `true`.
• `ServiceReg(`*MyCard*`,GBU,LOGOUT)` unregisters the GCU to the GBU leader of the local colony he is actually connected; the GCU will now work in local standalone mode; this function will set `RegMode` to `false`.
• `ListenLocal()` waits for a request from local API.
• `LocalServe(Data)` executes the `Data` on the local machine. It can fail.
• `PackScenario(Data)` encodes the scenario request with the `Data` to be sent, in the payload part of the GIP protocol, within the service request.
• `ServiceRequest(`*MyCard*`,GBU,MetaData)` sends a request of service to the leader GBU.
• `LocalReply(Response)` forwards locally `Response`.
• `ListenGBU()` waits for a request from the GBU.
• `CanHelp(MetaData)` checks if the request can be served.
• `ServiceResponse(`*MyCard*`,GBU,COMMAND)` answer to the GBU concerning the requested service.

```
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):                                 // 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.OPE 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 has 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 2. GCU **pseudocode**

- `CanServe(MetaData)` analyzes the request.
- `GetPeers(MetaData)` fetch some candidates peers.
- `GlobalServe(MyCard,Peers,Data)` forwards the request to the peers that the GBU found in his colony. The request will be processed remotely.
- `CanJoin/CanLeave(MetaData)` checks if the GCU can join/leave the colony.
- `ListenGCU()` waits for a request from GCU.
- `Verify(MetaData)` verifies if the request is well formed. It also verifies the PKI of the GCU, or it checks if the demanded service was already asked, etc.
- `UnPackScenario(MetaData)` decodes the scenario request from the `Data` received in the payload part of the GIP protocol, within the service request.
- `SendResult(MyCard,GBU,Response)` sends the results of the request to the requesting GCU.

The GRU's Semantics is described below.

```
while true do
 inparallel
  GIPacket = ListenLocal()                    // Local listening
  Route(MyCard,MyPeerCard,GIPacket)
 with
  GIPacket = ListenGlobal()                   // Global listening
  if GIPacket.TTL != 0
  then GIPacket.TTL --
       Deliver(MyCard,MyPeerCard,GIPacket)
  endif
 endinparallel
endwhile
```

The key functions of the algorithm are explained below. Let *MyPeerCard* denotes the name of the GCU (resp. GBU) which is uniquely attached, via a suitable API to the GRU, denoted by *MyCard*. This unit is the only units that *de facto* understands the GIP protocol; it will deals with Resource Discovery (function `Discover()` of the GCU (resp. GBU). The TTL slot in a GIP packet will be used to *count* the maximum number of *hops* from one unit to another: this value is useful to limit the number of request forwarded from one GBU to another one. This field help the GRU to discard some packets (typically service request) that "surfs" the Overlay Network looking for some "charitable" GCU that could help him.

- `ListenLocal()` waits for a request from the local API.
- `ListenGlobal()` waits for a request from Arigatoni.
- `Route(MyCard,MyPeerCard,GIPacket)` routes a GIP packet to its destination (defined in the `GIPacket`).
- `Deliver(MyCard,MyPeerCard,GIPacket)` unpacks and delivers a GIP packet to the peer (GCU or GBU) to which the GRU is uniquely attached.

The GBU's Semantics is described in the pseudo-code in Figure **??**. The key functions of the algorithm are explained below. It is composed by five processes:

- **Un/Registering**: implements the (un)registration of a GBU to a leader-GBU of a given supercolony.

```
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)://To ask something you and 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.OPE is
   SREG:                  // A GCU is asking for (un)registration
    Update(Colony,MetaData)
   SREQ:                  // A GCU 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
    case MetaData.ROLE is
     LEADER://A GBU is trying to register in a leader GBU
     if   CanJoin(MetaData)
     then RegMode = true
     endif
     if   CanLeave(MetaData)
     then RegMode = false
     endif
     INHABITANT://A GBU is asking for (un)registration
      Update(Colony,MetaData)
   SREQ:
     ... as for SREQ intra-colony
   SRESP:                 // A leader GBU responds to a request
    Sort&PushPeers4Id(MetaData)
    endcase
  endcase
 endwhile
endinparallel
```

Figure 3. GBU **pseudocode**

• **Basic shell:** read-eval-print loop. The GBU itself can work in local standalone mode (*i.e.* it does not forward any requests to other brokers), or in global mode (any request that cannot be completely served intra-colony is forwarded to the leader-GBU of the supercolony).
• **Spool:** associative list composed by an unique identifier of a service request and a list of GCUs that have accepted to serve the task associated with the identifier.
• **Intra-colony Listening:** listens for any communication (service request or service response) from the local colony.
• **Inter-colony Listening:** deals with the interaction between the leader-GBU of the colony and the superleader-GBU of the supercolony where the colony is registered: this interaction takes place after a clear phase of negotiation between both leaders of colonies.

We assume, that the following variables are shared by all processes, via classical semaphores *à la* Dijkstra:
• `Colony` is the set of colony's inhabitants.
• `Peers4Id` is a dictionary of the shape `[(Id, Peers)]*` denoting, for each service request `Id`, the list of potential `Peers` that have accepted to serve `Id`.
• `History` is a dictionary of the shape `[(Id, MetaData)]*`, where `MetaData` contains all the informations about the kind of request.
• `GlobalMode` is *true* if and only if the GCU works in global mode; is *false* otherwise.
• `RegMode` is *true* iff the GCU has been registered in a given colony; it holds *false* otherwise. Unless unregistered, the GCU must keep the dialog with the GBU.
• GBU holds all the security and network informations of the leader of the colony.

A short explanation of the GBU pseudocode follows.
• `Discover(MyCard)` discovers the leader-GBU, upon physical/logical insertion in the Overlay Network.
• `ListenPeer()` waits for a request from an individual of the colony.
• `PushHistory(MetaData)` push the pair `(Id, MetaData)` on the `History` dictionary (`Id` is contained in `MetaData` as well).
• `SelectPeers(Colony,MetaData)` performs a *static analysis* about the possibility to fully satisfy the service request *inside* the local colony, *i.e.* without forwarding the request out of the colony; if the function returns {},

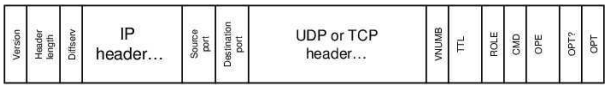

Figure 4. A GIP packet on UDP or TCP

then the request *a priori* cannot be satisfied internally.
• `Sort&PushPeers4Id(MetaData)` inserts and sort the peers of `GetPeers(MetaData)` in the list of peers identified by `Peers4Id(GetId(MetaData))`: sorting is done following *ad hoc* criteria w.r.t. the resources requested for a given scenario.
• `Update(Population,MetaData)` logs and delogs one GCU (resp. GBU), whose coordinates are contained in `MetaData`, from the colony (denoted by `Population`); the criteria of logging/delogging depend on the security policy the colony has adopted.
• `Timeout(Id)` is true when a service request, labeled with a given `Id`, oversize a fixed time of waiting.
• `Satisfy(Peers,History(Id))` checks for a service request `Id` (in `History`), the `Peers` capabilities.
• `GetBestPeers4Id(Id)` selects the "best" peers for the request with `Id` key, from a list of potential peers: the selection criteria depends, among others, on the peculiar scenario we are dealing with.
• `PopPeers4Id(ID)` pops the pair `(ID,PEERS)` in the `Peers4Id` dictionary.
• `CanJoin(MetaData)` checks if the GBU can join the colony; it also verifies that the registration does not induce *cycles* in the colony the GBU he is trying to join.

4. The GIP Protocol

For obvious lack of space, many details of the protocol are left implicit. As shown in Figure **??**, the GIP packet resides in the payload of a UDP datagram, or eventually of a TCP packet. We let the common datatypes, like *Byte*, *Int*, *Bool*, *Set*, etc. plus the *Variable-Length* (recursive)-type *Vlt* defined as follows.

Definition 1 (*Vlt* Type) *Any element of type Vlt has the following two fields:*
1) LENGTH *: Int is the length of the* Payload *in bytes.*
2) PAYLOAD *: Vlt contains the data to be interpreted.*

The fields of the GIP protocol are:
• VNUMB : *Int*: version number of the protocol.
• TTL : *Int*: *"time to live"* of the packet protocol introduced to avoid that packets "lives" too much in the Network jumping from one GBU to another GBU.
• ROLE : *Bool*: the role of the sender of the packet, either a LEADER or INHABITANT.
• CMD : *2Byte* command carried by the packed. It is composed by the two subfields SERVICE,VALUE : *Byte*.
• OPE : *Vlt* describes, for each command, a particular operation and its parameters.
• OPT? : *Bool* indicates that options are present at the end of the GIP packet.
• OPT : *Vlt* describes the optional fields.

For each command described in the CMD field, the OPE field contains, in its payload field, all data necessary to perform the command. For lack of space, we only describe the CMD and the OPE fields.

The CMD Field. The GIP allows the three services, namely SREG, SREQ, and SRESP.
• (SREG : *Byte*,VALUE : *Byte*), a.k.a. *Service Register* is used for the registration of either a GCU to a GBU, or a GBU (leader of a subcolony working in local mode) to another GBU leader of another colony that physically (or logically) contains the subcolony. Registration is acknowledged by both units. Values are:
· LOGIN applies when a GCU wants to register to a GBU, or when a GBU (representing a subcolony) wants to register to another GBU.
· LOGOUT applies when a GCU wants to unregister to a GBU or when a GBU (representing a subcolony) wants to unregister to another GBU.
· LOGGED applies when a GBU notifies a successful registration to an individual.
· UNLOGGED applies when a GBU notifies a failed registration to an individual.
• (SREQ : *Byte*,VALUE : *Byte*), a.k.a. *Service Request* is sent by a GCU in global mode to request a service to its GBU. A GBU in global mod forwards this request to another super GBU, in case it did not find in its own colony all the needed resources to serve the request. A GBU also can sends this request to every registered inhabitant of his colony, namely GCUs or GBUs leader of some subcolonies. Every bit of VALUE represents any possible distributed resource that can be asked, *i.e.*:
· (bit 0) CPU: we ask for computational power.
· (bit 1) MEM: we ask for memory space.
· (bit 2) DATA: we ask for some (distributed) files.
· (bit 3) BAND: we ask for some bandwidth (the GCU is usually an ISP).
· (bit 4) WEB: we ask for web services.
· (bit 5) RUN: we ask to abort a run, pack everything (complete dump of the registers, stack, etc.) in a closure and migrate somewhere the computation.
· (bit 6): left for future use.
· (bit 7): parity bit.

Of course, a combination of different requests can be done, like the following one that ask for CPU, Memory, Data, and Bandwidth like in `1 1 1 1 0 0 0 0`.
• (SRESP : *Byte*,VALUE : *Byte*), a.k.a. *Service Response* is sent by a GCU to a GBU, to answer a received SREQ.

It is also exchanged between two GBUs or from a GBU to a GCU, following the reverse path of the SREQ. It indicates whether or not the individual may process the request of the leader GBU. A service response is also exchanged between two GCU when one servant GCU acknowledge the reception of the request from a client GCU, or to inform the client that it has to wait since the request is still processing on the servant, or to send the result or informations on how to retrieves the result to the client. Possible kinds of values are

· ACC: the request is accepted. Sent by a GBU to the individual which transmitted the request.

· REJ: the request cannot be processed. Sent by a GBU to the individual which transmitted the request.

· DONE: the request has been processed. Sent by an individual to the GBU, leader of the colony.

· ERR: the request has been processed, but some errors occurs (*i.e.* a `core dump` in a run). Sent by an individual to the GBU, leader of the colony.

· SPOOF: the request cannot be processed, because of some problems in the authentication. Sent by an individual to the GBU, leader of the colony.

· NOTIME: the request has expired its time-frame. Sent by the GBU to individuals of its colony.

· RES: the request is processed and the result is going to be transmitted. Sent by an individual to the GBU, leader of the colony.

The OPE Field. The OPE field of type *Vlt* is used to encode in its payload part all the information necessary to execute the command.

• For an SREQ command:

· ID :*4Byte* is the unique ID identifying the request carried by this command. This field is created by the original individuals which emitted the request and is left unchanged by all the nodes forwarding the request.

· CARD: *Vlt* contains all the informations necessary for the exchange between the client and a servant (*i.e.* Protocol, IP Address, Port number, PKI, etc.).

· REQNUMB:*Int* is the number of request units follow in the packet. This number *must not* be equal to zero.

· REQDATA: *Vlt** (a list of *Vlt*) describes all informations necessary to deal with a simple request.

• For an SRESP command:

· ID :*4Byte* contains the unique ID identifying the request carried by this command.

· CARD: *Vlt* contains all the informations necessary for the exchange between the client and a servant (*i.e.* Protocol, IP Address, Port number, PKI, etc.).

· RET: *Vlt* contains the result of the request.

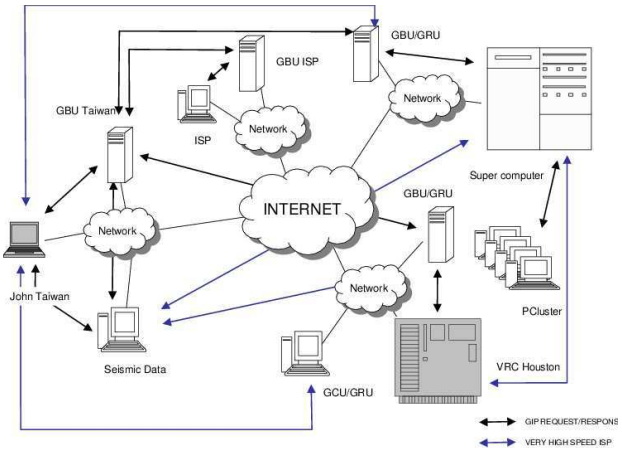

Figure 5. Scenario for Seismic Monitoring

5. Scenario for Seismic Monitoring

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 data mass 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` or `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 and a cluster of PC has to be *rented* by the SeismicDataCorp company. John will ask also for *bandwidth* in order to get rid of any bottleneck related to the big amount of data to be transferred.

Then, the processed data should be analyzed using the *Virtual Reality Center, (*VRC*)* based in Houston, U.S.A. by a specialist team and the resulting recommendations for the next data collect campaign have to be sent to John. As such:

1) John logs on 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 GBU leader of the colony will receive and process the request.

2) If the Resource Discovery performed by the GBU succeeds, *i.e.* a supercomputer and a cluster and an ISP are found, then the data are transferred at a very high speed and the *"Sinfonia"* begins.

3) John will also ask (in the GIP query) to the GCU containing the seismic data to dispatch suitable chunks of data to the supercomputer and the cluster designated by the GBU to perform some pieces of computation.

4) John will also ask (in the GIP query) to the global

supercomputer the task of collecting all intermediate results so calculating the final result of the computation, like a *"Maestro di Orchestra"*.
5) The processed data are then sent from the supercomputer, via the high speed ISP to the Houston center for being visualized and analyzed.
6) Finally, the specialist team's recommendations will be sent to John's laptop.

This scenario is pictorially presented in Figure **??** (we suppose a number of subcolonies with related leaders GBU, all registered as individuals to a superleader-GBU (for example the John's GBU could be elected as the superleader). For simplify security issues, all GBU's are trusted using the same PKI, making *de facto* in common all resources of their colonies.

6. Related and Future Work

Related work. Many technologies, algorithms, and protocols have been proposed recently on Resource Discovery in Overlay Networks. Some of them focus on Grid or P2P applications, but none of those targets the full generality of the Arigatoni. Our model deals only on generic Resource Discovery for building an Overlay Network of Global Computers, structured in a Virtual Organization with clear and distinct roles between leader and individuals. This section briefly discusses some of the closest technologies and architectures found recently in the literature.

The Globus Toolkit [**?**], is an open source set of technology, protocols, middleware, used for building Grid systems and applications. Possible applications range from sharing computing power to distributed databases in a heterogeneous overlay network, where security is taken seriously into account. The toolkit includes stand alone software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. The analogies with the Arigatoni model lies in the *Community Scheduler Framework* component and the *Web Service Grid Resource Allocation and Management* of the toolkit concerning the Resource Discovery, and the *Globus Teleoperations Control Protocol* to allows units to cooperate (analogy with our GIP protocol).

Puppin *et al.* [**?**,**?**], following the lines of [**?**], designed and implemented a super-peer overlay network, using the Globus technology, as a trade-off between totally distributed systems and cache based services. Their network shares some similarities with our tree-based architecture of the Virtual Organization induced by the Arigatoni model, especially in case of network with no-redundancy.

Promoted by Sun, the JXTA [**?**] technology is a set of open peer-to-peer protocols that enable any device to communicate, collaborate and share resources. After a peer discovery process, any peer can interact *directly* with other peers. Hence the overlay network of peers induced by the JXTA technology is *flat*. In fact the main concern of Arigatoni model is Resource Discovery, while the main concern of the JXTA technology is to offer some tools to implement a P2P model. In Arigatoni, any individual *first* asks to the GBU leader of the colony it belongs and *then* collaborates with any peers suggested by the GBU. Consequently, the JXTA flat set of individuals is replaced in Arigatoni by a hierarchy of colonies, and subcolonies, that can dynamically change upon registration or unregistration of the different individuals, *i.e.* the topology induced by the model is a dynamic tree. Moreover, Arigatoni focuses on the evolution/devolution of colonies and the mechanism of resource discovery, while JXTA technology allows peers to communicate using an already existing overlay network of peers. Arigatoni aims are dynamicity of the overlay network while JXTA aims are freedom of connectivity between peers. Finally peers in the JXTA architecture come with their proper JXTA-ID (logical JXTA peers addressing) while Arigatoni relies on the more conventional IP addresses. As such, a peer in a JXTA network is uniquely identified by its *peer* ID allowing the peer to be addressed independently of its physical addresses.

NaradaBrokering [**?**] is an open-source, distributed messaging infrastructure based on the Publish/Subscribe paradigm. A broker distributes and routes messages, while working with multiple underlying communication protocols. The broker network in NaradaBrokering is based on hierarchical, cluster-based structure which can support large heterogeneous client configurations. The routing of events within the substrate is very efficient since for every event, the associated targeted brokers are usually the only ones involved in dissemination. Furthermore, every broker computes the shortest path to reach target destinations while eschewing links and brokers that have failed or are suspected to fail. Various data structures are used to encode topic descriptors in order to implement efficient search procedures. Arigatoni is very complementary to NaradaBrokering since it mainly concentrates on Resource Discovery and peer selection based on service requests.

The OurGrid architecture [**?**] is oriented to share computational power and does not match with the complete genericity of Arigatoni. From this point of view Arigatoni is a generalization of the OurGrid architecture. Arigatoni is based on the formal model of colonies, the dynamic tree of brokers and a trade off between P2P

and Grid models thanks to an extended version of the Publish/Subscribe paradigm.

In [**?**], a P2P approach for Resource Discovery in Grid environments is proposed. The authors present a framework that drives a design of any Resource Discovery architecture. In [**?**], non-uniform information dissemination protocols are used to efficiently propagate information to distributed repositories, without requiring flooding or centralized approaches. Results indicate a significant reduction in the overhead compared to uniform dissemination to all repositories.

In [**?**], a semantic Resource Discovery in the Grid is proposed using a P2P network to distribute and query to the resource catalog. Each peer can provide resource descriptions and background knowledge, and each peer can query the network for existing resources.

In [**?**], the authors investigate the applicability of a structured overlay network for the discovery of Grid resources based on the P-GRID overlay network and presents experimental results from a large-scale deployment on PlanetLab [**?**]. We do believe that our approach is complementary to this overlay network in the sense that it provides the necessary basic infrastructure necessary to a real deployment of the overlay network itself. Moreover, our work abstract on *which kind of resource* the overlay network is playing with; pragmatically, 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 in the style of the language Obliq [?].

However, all these papers propose high level mechanisms or algorithms and do not address the full generality of Arigatoni.

Future work. We are improving our model with several new features, such as the possibility to ask a certain number of instances of a service (*i.e.*, the system should find the specified number of GCUs capable of providing that service), or the possibility to embed services in conjunctions (*i.e.*, the services in a conjunction should be provided by the same GCU), or load balancing issues. We are working on the implementation of a real prototype and the subsequent deployment on the PlanetLab experimental platform, and/or on GRID5000, the platform available at the INRIA. As part of our ongoing research, we are also working on a more complete statistical study of our system, based on more elaborate statistical models and realistic assumptions. Future works will also focus on security issues as for example using many PKI instead of a unique PKI, the study of trust models based on reputation and more advanced security models and techniques.

Acknowledgments. We warmly thanks Nicolas Bonneau for the careful reading of the paper. This work is supported by Aeolus FP6-2004-IST-FET Proactive.

References

[1] BitTorrent, Inc. The Bittorrent Home Page. `http://www.bittorrent.com`.

[2] L. Cardelli. A language with distributed scope. *Computing Systems*, 8(1):27–59, 1995.

[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] Community Grid Labs. Narada Brokering Home Page. `http://www.naradabrokering.org/`.

[5] M. Cosnard, L. Liquori, and R. Chand. Virtual Organizations in Arigatoni. *DCM: International Workshop on Developpment in Computational Models. Electr. Notes Theor. Comput. Sci.*, 2006. To appear.

[6] Globus Alliance. Globus Home Page. `http://www.globus.org/`.

[7] M. Hauswirth and R. Schmidt. An Overlay Network for Resource Discovery in Grids. In *Proc. of International Workshop on Database and Expert Systems Applications, DEXA*, pages 343–348. IEEE, 2005.

[8] F. Heine, M. Hovestadt, and O. Kao. Towards Ontology-Driven P2P Grid Resource Discovery. In *Proc. of International Workshop on Grid Computing, GRID*, pages 76–83. IEEE/ACM, 2004.

[9] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[10] A. Iamnitchi, I. T. Foster, and D. Nurmi. A Peer-to-Peer Approach to Resource Location in Grid Environments. In *Proc. of High Performance Distributed Computing, HPDC*, page 419, 2002.

[11] V. Iyengar, S. Tilak, M. J. Lewis, and N. B. Abu-Ghazaleh. Non-Uniform Information Dissemination for Dynamic Grid Resource Discovery. In *Proc. of Network Computing and Applications, NCA*. IEEE, 2004.

[12] JXTA Community. JXTA Home Page. `http://www.jxta.org/`.

[13] R. Milner. *A Calculus of Communicating Systems*. Springer, 1980.

[14] D. P. S. Moncelli, R. Baraglia, N. Tonellotto, and F. Silvestri. A Grid Information Service Based on Peer-to-Peer. In *Proc. of Euro-Par*, pages 454–464, 2005.

[15] OurGrid Project. OurGrid Home Page. `http://www.ourgrid.org`.

[16] Planet Lab Consortium. Planet Lab Home Page. `http://www.planet-lab.org/`.

[17] D. Puppin, F. Silvestri, and D. Laforenza. Component Metadata Management and Publication for the Grid. In *Proc. of ITCC*, pages 187–192, 2005.

[18] A. Rapoport. Mathematical models of social interaction. In *Handbook of Mathematical Psychology*, volume II, pages 493–579. John Wiley and Sons, 1963.

[19] U. Schwiegelshohn, R. Yahyapour, and P. Wieder. Resource Management for Future Generation Grids. Technical Report TR-0005, CoreGRID, 2005.

[20] J. White. Telescript technology: the foundation for the electronic marketplace. White Paper. General Magic, Inc., 1994.

[21] B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. In *Proc. of ICDE*, 2003.