Resource Discovery in the Arigatoni Model.

Raphael Chand, Luigi Liquori, Michel Cosnard

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Resource Discovery in the Arigatoni *Model*

Raphael Chand — Luigi Liquori — Michel Cosnard

N° 5924

Mai 2006

Thème COM

ISSN 0249-6399

Resource Discovery in the Arigatoni Model

Raphael Chand , Luigi Liquori , Michel Cosnard

Thème COM — Systèmes communicants
Projets Mascotte

Rapport de recherche n° 5924 — Mai 2006 — 22 pages

Abstract: Arigatoni is a lightweight communication model for dynamic Resource Discovery. Inspired by the Publish/Subscribe paradigm, the Arigatoni model implements a *Resource-Discovery Oriented* Overlay Network. 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 SubColonies of Individuals, or basic units called *Global Computers*. Global Computers communicate by first registering to the Colony and then by mutually asking and offering services. The leader, called *Global Broker*, has the job to analyze service requests/responses coming from its own Colony or arriving from a surrounding Colony, and to route requests/responses to other Individuals. After this discovery phase, Individuals get in touch with each others without any further intervention from the system, typically in a P2P fashion. Communications over the behavioral units of the model are performed by a simple *Global Internet Protocol* on top of the TCP or UDP protocol. Arigatoni provides fully decentralized, asynchronous and scalable Resource Discovery, that can be used for various purposes from P2P applications to more sophisticated Grid applications. The main focus of this paper is to present the Resource Discovery mechanism used in the Arigatoni model, along with some simulations that show that Resource Discovery in Arigatoni is efficient and scalable.

Key-words: Resource discovery, Overlay Networks, Global computing

Découverte de ressources dans le modèle Arigatoni

Résumé : Arigatoni est un modèle de communication léger pour la découverte de ressource dynamique. Inspiré du paradigme Publier/Souscrire, le modèle Arigatoni implémente un réseau recouvrant pour la découverte de ressource. Les entités dans Arigatoni sont organisées dans des *Colonies*. Une colonie est une organisation virtuelle simple composée d'exactement un leader, qui offre des services de type courtier, et un ensemble d'*Individus*. Les Individus sont des sous-colonies d'Individus, ou des unités basiques appelées des *Ordinateurs Globaux*. Les ordinateurs globaux communiquent en s'enregistrant tout d'abord à la Colonie, ils peuvent ensuite demander et offrir des services de manière interchangeable. Le leader, appelé *Routeur Global*, doit analyser les requêtes ou les réponses arrivant de sa propre Colonie ou d'une Colonie voisine, et router les requêtes ou réponses vers d'autres Individus. Une fois cette phase de découverte achevée, les Individus entrent en contact les uns avec les autres sans d'avantage d'intervention de la part du système, suivant le modèle pair-à-pair. Les communications entre les unités actives du modèle s'effectuent au moyen d'un protocole simple appelé GIP, qui utilise le protocole TCP ou UDP. Arigatoni offre une découverte de ressources entièrement décentralisée, aynchrone et extensible, et peut être utilisé à des fins divers, depuis les applications pair-à-pair jusqu'aux applications plus sophistiquées utilisées dans les grilles. Le principal objectif de cet article est de présenter le mécanisme de découverte de ressources utilisé dans le modèle Arigatoni, accompagné de simulations qui montrent que la découverte de ressources dans Arigatoni est efficace et extensible.

Mots-clés : Decouverte de ressources, Reseau recouvrant, Informatique globale

1 Introduction

Motivations. The *Global Computing Communication Paradigm*, *i.e.* computation via a seamless, geographically distributed, open-ended network of bounded resources by agents (called *Global Computers*) acting with partial knowledge and no central coordination is probably one of the most interesting challenges for the next decade. The paradigm provides uniform services with variable guarantees. Aggregating many Global Computers sharing similar or different resources leads to a *Virtual Organization*, also called *Overlay Computer*[1]. 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 new field of research is how single resources, offered by the Global/Overlay Computers are discovered. The process is called *Resource Discovery*: it requires an *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 offered by Global/Overlay Computers and the possibility of tolerating intermittent participation and dynamically changing status/availability of the latter.

The first presentation of the Arigatoni model was given in [2]. In this paper, we show the insights of the model concerning the dynamic Resource Discovery. Reciprocity and hierarchical organization of the Virtual Organization in *Colonies*, governed by a clear leader (called *Global Broker*) are the main achievements of the Arigatoni model. Global Computers belong to only one Colony, and requests for services and resources located in the same or in another Colony traverse a broker-2-broker negotiation whose security is guaranteed via standard PKI mechanisms. Once the resource offered by a Global Computer has been found by the Overlay Network, the real resource exchange will be performed out of the Arigatoni model itself, *e.g.* in a P2P fashion.

As such, the main concern of the Overlay Network induced by the Arigatoni model is *Resource Discovery*. In this paper, we explain how Arigatoni offers decentralized, asynchronous, and generic Resource Discovery. Once a Global Computer has issued a request for some services, the 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 fact that the Arigatoni model only deals 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 sharing, 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

[1] Overlay Computer: abstraction that can be implemented on top of a Global Computer to yield another Global Computer [18].

being useful in case of catastrophic scenarios, like fire, terrorist attack, earthquake etc., in the vein of global programming languages *à la* Obliq [3] or Telescript [19].

Related work. Many technologies, algorithms, and protocols have been proposed recently on Resource Discovery. Some of them focus on GRID or P2P oriented applications, but none of those targets the full generality of the Arigatoni model that deals only with Generic Resource Discovery for buiding an Overlay Network of Global Computers, structured via a Virtual Organization and clear distint roles between leader and Individuals (GCUs or SubColonies).

This section briefly discusses some of the closest technologies and architectures found recently in the literature.

The Globus Toolkit [11], is an open source set of technology, protocols and 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 seriously taken 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 lie 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 allow units to cooperate (analogy with our *ad hoc* protocol).

Promoted by Sun, the JXTA [16] 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*. Moreover, the main concern of the Arigatoni model is Resource Discovery, while the main concern of the JXTA technology is to offer some tools to implement a P2P model. In addition, the Arigatoni model 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.

NaradaBrokering [8] 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 a hierarchical, cluster-based structure which can support large heterogeneous client configurations. Furthermore, every broker computes the shortest path to reach target destinations while eschewing links and brokers that have failed or are suspected to failure. Arigatoni is very complementary to NaradaBrokering since it mainly concentrates on Resource Discovery and peer selection based on service requests.

The OurGrid architecture [17] aims at sharing computational power and does not match with the complete genericity of the Arigatoni model. 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 [14], a P2P approach for Resource Discovery in Grid environments is proposed. More precisely, the authors present a framework that drives a design of any resource discovery

architecture. In [15], 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 [13], 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.

Several publish/subscribe (pub/sub) have been developed recently, such as XNet [7, 6], Siena [4] or IBM Gryphon [1]. In [12], the authors propose to adapt the Siena publish/subscribe system to achieve Gnutella-like Resource Discovery. Their work resembles ours in the sense that Arigatoni also extends the pub/sub paradigm. However, in [12], Resource Discovery is achieved by publishing queries to the notification service. In contrast, Arigatoni implements its own Resource Discovery algorithm, especially designed for generic and scalable resource lookup.

2 System model

2.1 Units, topology

Two different kinds of units compose the Arigatoni system: *Global Computer Units* (GCU), and *Global Broker Units* (GBU).

A GCU is the basic peer of the Global Computing paradigm. It is typically a small device, like a PDA, or a PC, connected with any IP network, unrelated to the media used, wired or wireless, etc.

A GBU is the basic unit devoted to register and unregister GCUs, to receive service queries from client GCUs, to contact potential servant GCUs, to negotiate with the latter the given services, to trust clients and servers and to send all the information necessary to allow the client GCU, and the servants GCUs 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 requested service (with related trust certificate) from the proper GBU, then the client will enjoy the service directly from the servant(s) GCU, *i.e.* without a further mediation of the GBU itself.

A *Colony* is a simple virtual organization composed by exactly one leader and some set (possibly empty) of Individuals. Colonies are organized in a tree structure (at least for the duration of a request) where the parent of a Colony is its *leader* in the Arigatoni model. Individuals are Global Computers (think it as an *Amoeba*), or SubColonies (think it as a *Protozoa*). An Individual can be a GCU or a GBU (representing the leader of a a SubColony). GCUs cannot have children in the hierarchy. As such, GBUs can have both GBUs and GCUs as their children. The two main characteristics of a Colony are that it has *exactly* one leader

GBU and at least one Individual (the GBU itself), and that it contains Individuals (some GCU's, or other colonies).

A *Community* is a raw set of colonies and Global Computers (think it as a *soup* of colonies and GCU without a leader). Starting from a community, the Arigatoni protocol allows Individuals to dynamically aggregate in colonies. This topic has been addressed in [9].

The possibility for Individuals to log in/delog from a Colony or the possibility for a Colony's leader to delog some "lazy" Individuals makes *de facto* the network topology *dynamic*. This dynamicity implies that if GBUs hold routing tables about the services provided by their Colony, particular care must be taken to maintain consistency when Individuals log/delog. Moreover, due to the fact that Individuals are not *slaves* but Global Computers with their own proper activity, a service request may lead to run-time failures. This happens when an Individual gets busy by a local request, or when it suddenly delogs from the Colony during the routing of the service request, or worst, when it gets hardware failures.

2.2 Arigatoni extends the Pub/Sub Paradigm for Resource Discovery

In the Arigatoni model, Resource Discovery works by asynchronously disseminating request messages in the system until some Individuals have been found. More precisely, when Global Computers log in the system (a Colony), they declare the list of services that they can offer. When a Global Computer asks for some services, it issues a service request message to its leader, without addressing it to any particular receiver. The system disseminates the message according to the services included in it *and* according to the services that the other Global Computers have declared. As a consequence, the communication model underlying the Arigatoni model extends conservatively the *Publish/Subscribe* (pub/sub) paradigm [10]. Indeed, in the pub/sub paradigm, consumers subscribe to the system (typically called the *Notification Service*) to specify the type of information that they are interested in receiving. Producers publish data to the system. The notification service disseminates the data to all (if possible) the consumers that are interested in receiving it, according to the *content* of the data *and* the interests declared by the consumers. In Arigatoni, Global Computers "subscribe" to the system by declaring the services that they offer to serve. The same Global Computers also "publish" data in the system when they issue service requests. Arigatoni disseminates the data according to the services included in the requests and the services that the other Global Computers have declared.

The pub/sub like communication form used in Arigatoni for Resource Discovery has several advantages. First, it allows Arigatoni to realize a full decoupling, in time, space, and synchronization, between the Global Computers. Second, due to its asynchronous nature, the Arigatoni model is, potentially, more scalable and can work in "disconnected" mode (*e.g.*, for mobile users and wireless devices). Third, indirect addressing makes it possible for the infrastructure to implement reliability, load balancing, fault-tolerance, persistence, or transactional semantics. More practically, since Arigatoni has a tree-like topology, we can use the pub/sub subscription mechanisms described in existing tree-based pub/sub systems such as XNet [6, 7, 5] or Siena [4], for subscription management, *i.e.*, for the construction and the

update of *consistent* routing tables in the system. In addition, we can use the reliability mechanisms described in [7] to allow Arigatoni to be fault-tolerant or to adapt to dynamic topology changes.

However, one major difference between Arigatoni and classic pub/sub systems lies in their *functionality*. Indeed, the classic pub/sub paradigm deals with the publication of messages whereas Arigatoni focuses on *pure* Resource Discovery. More precisely, classic pub/sub systems aim at disseminating published messages to *all* interested consumers. In contrast, in Arigatoni, when a service request is issued, the goal is to find one (or maybe "some") Individuals able to provide the services included in the request, but not *all* the potential Individuals. As a consequence, a much smaller fraction of the system is traversed. Besides, the routing strategy used by a GBU consists in always trying to find potential GCU in its own Colony first. If it fails, it then delegates the request to its leader. This strategy is reminiscent of the *dynamic method lookup* employed in all Object-Oriented languages, and increases *resource encapsulation* inside colonies, another concept strongly related to Object-Orientation.

Another major difference lies in the nature of the published events in classic pub/sub systems and the nature of service requests in Arigatoni. Indeed, in classic pub/sub systems, subscriptions are constraints on the set of all possible events. In contrast, in Arigatoni, service requests are also expressed as constraints. This latter point will be explained in more details in the next section.

3 Resource Discovery

Let $\mathcal{R}$ be the set of all possible resources (maybe infinite). GCUs provide resources by registering services to the system. A service S is a constraint on the set of resources. $match(S) \in \mathcal{R}$ is the set of resources that satisfy S. A GCU X that register S announces that it can provide the set of resources $match(S)$. A GCU Y that issues a service request for service S' is looking for a resource that satisfies constraint S', i.e., a resource in $match(S')$. If $match(S) \cap match(S') \neq \emptyset$, then there exists a resource that satisfies both S and S', and X can provide a resource to Y. We say that S and S' *overlap* iff $match(S) \cap match(S') \neq \emptyset$. For example, $S =$ "$[type = CPU][Time < 10s]$" and $S' =$ "$[type = CPU][Time > 5s]$" overlap, since any resource with attribute "$Time$" between $5s$ and $10s$ matches.

The principle of Resource Discovery in Arigatoni is as follows. When a GCU sends a request for a set of services $S_1 \cdots S_n$, it builds a "ServiceRequest" message containing the set of services and sends it to its leader GBU. The message is then recursively processed by the GBUs in the system so as to find "some" Individuals able to serve the services included in the request. The main basic principle of the protocol is that every GBU that receives a request always searches its own Colony first to find the potential Individuals able to serve the services included in the request. If no Individuals are found, then the request is delegated to its leader GBU, and the process proceeds recursively. In addition, if the GBUs maintain some information about the services provided by their children, then they can transform a received request into sub-requests, so as to only ask a given child for the services that it (or its colony) provides.

Eventually, the process leads to some GCUs[2] receiving a request. When one such GCU receives a request for some services, it chooses the services that it accepts to serve and the ones that it refuses to serve. It then sends a "ServiceResponse" message containing the list of accepted services and the list of rejected services, and sends it to its leader GBU. The response messages are then propagated recursively in the system, following the reverse path. The Resource Discovery protocol is formally described in pseudo-code and explained in more details in the following section.

3.1 Resource Discovery in the Arigatoni GIP Protocol

Resource Discovery in the Arigatoni model is the core of the GIP protocol; it is described in pseudo-code in Algorithm 1 and explained as follows. We only focused on the case of GBUs. The Resource Discovery algorithm in the case of GCUs is similar and has been voluntarily omitted (see [2] for details). Indeed, the involvement of GCUs in the process of Resource Discovery is limited to directly replying to request messages. As stated before, Arigatoni only concerns with the discovery of resources. The real resource exchange is done out of the Arigatoni model itself, *e.g.* in a P2P fashion. Let $\mathsf{GBU_N}$ receive a message from a neighbor.

Algorithm 1 The Resource Discovery Routine in the Arigatoni GIP Protocol

```
1:  case Message is
    SREQ :
2:      ReturnPath{Message.Id} ← Message.Sender
3:      SendList ← SelectPeers(Message.Services, search_mode)
4:      for each (P, Serv(P)) ∈ SendList do
5:          Send ServiceRequest(Serv(P)) to P
6:      end for
7:      for each S∈Message.Services such that ∄(P, Serv(P))∈SendList, S∈Serv(P) do
8:          Append S to RejectList
9:      end for
10:     Send ServiceResponse({},RejectList) to ReturnPath[Id]
11: SRESP :
12:     for each S∈Message.AcceptedServices do
13:         if (S was not already accepted ) ∨ (EXHAUSTIVE_REPLY is set) then
14:             Append S to AcceptList
15:         end if
16:     end for
17:     SendList ← SelectPeers(Message.RejectedServices, intra_Colony_mode)
18:     for each (P, Serv(P))∈SendList do
19:         Send ServiceRequest(Serv(P)) to P
20:     end for
21:     for each S∈Message.RejectedServices such that ∄(P, S(P))∈SendList, S∈Serv(P) do
22:         Append S to RejectList
23:     end for
24:     Send ServiceResponse(AcceptList,RejectList) to ReturnPath[Id]
25: end case
```

[2]The model can be easily extended so that GBUs are also able to directly reply to requests, *i.e.*, to act as Global Computer. Indeed, we can simply consider that each GBU has, *de facto*, a *virtual* GCU directly attached to it.

Case of Service Request (SREQ). We first consider the case of request messages. A request message received by $\mathsf{GBU_N}$ means that N is asked to find some Individuals to provide the services included in the request. For that purpose, N first maps the "Id" of the request included in the message to the sender of the message (line 2), so as to allow reply messages to follow the reverse path of the request.

Line 3: Various intra Colony search modes. The leader N then calls function "SelectPeers", taking as input the list of services included in the request message, $Message.Services$ (line 3). Then SelectPeers returns a list of pairs $\{(\mathsf{P}, Serv(\mathsf{P}))\}$, called *SendList*, where the first element P of a pair is the Id of a neighbor, and the second element $Serv(\mathsf{P})$ is a list of services, subset of $Message.Services$, that contains the list of services to ask to neighbor P. The *search_mode* determines the way function SelectPeers determines the *SendList*. The *search_mode* depends itself on whether P maintains some information about the services provided by its Colony, *i.e.* a routing table. Currently, the following search mode are allowed: *broadcast* and *selective*, where the latter is itself sub-divided into tree sub-modes: *exhaustive*, *greedy random*, and *greedy ordered*. If P does not maintain a routing table, then it has no other choice than to ask all its children for all the services included in the message, *i.e.*, to *broadcast* the request message. We will refer to this search mode as the *broadcast* mode. Now if P maintains a routing table that indicates which child leads to a potential Individual able to serve a given service, then P can *selectively* send *customized* requests to its children. More formally, P only asks a child for a service that *overlaps* a service that it advertised, *i.e.* there exists a resource that satisfies both the service requested and the service advertised.We will refer to this mode as the *selective* mode. Consequently, P can choose "some" children and send them a request for the services that overlap the ones that they advertised. The selective search mode can then be refined as follows. Consider a particular service S included in the request message.

- In the *exhaustive* mode, P sends a request for service S to all the children that can serve it (*i.e.*, that contain potential Individuals in their Colony).
- In the *greedy random* mode, P sends a request for S to only one child that can serve the request, chosen uniformly at random.
- In the *greedy ordered* mode, P sends the request to only one child, chosen according to some predefined or *ad hoc* criteria (*e.g.*, depending on network factors, or according to the quantity of services that were accepted by each child, *à la* tit-for-tat).

In addition, we can refine even more the greedy modes, by introducing a parameter n, that defines the number of children to whom the request is sent. We could then define the *n-greedy random* or the *n-greedy ordered* modes. It is important to mention that the *SendList* variable can contain N's leader, lets call it L. That is, it may contain a pair $(\mathsf{L}, Serv(\mathsf{L}))$. As explained in the previous section, when considering a particular service $S \in Serv(\mathsf{L})$, this only happens when *no* child advertised some services that overlap S, *i.e.*, there are no potential Individuals able to serve service S in N's Colony. GBU N then *delegates* service S to its leader GBU. In addition, obviously enough, note that to prevent routing loops, the sender of the request message is never considered as a potential service provider.

Lines $4-6$: Forwarding service request messages. Consequently, for each pair $(\mathsf{P}, Serv(\mathsf{P}))$ in the *SendList*, N sends to neighbor P a service request message for services $Serv(\mathsf{P})$ (lines $4-6$).

Lines $7-9$: Services rejection. Finally, each service S included in the request message, and such that no potential Individual was found amongst N's neighbors, is reported as rejected by N, to the original issuer of the request message (lines $7-10$). Note that since N may only maintain information about its own Colony (no information is maintained about its leader, other than its id), this may only happen if N is the root of the topology or if the request message originated from N's leader.

Case of Service Response (SRESP). We now consider the case of reply messages. As previously explained, the process of propagating SREQ messages eventually leads to a certain number of GCUs receiving a request. Each such GCU sends a reply message to its leader, with the list of accepted and the list of rejected services, along with its Id. Consequently, a given GBU_N that participated in the propagation of the SREQ message eventually receives a certain number of SRESP messages from each of its children that was sent an instance of the (maybe transformed) SREQ message. Consider now an SRESP message sent to GBU_N by a neighbor Q.

Lines $12-16$ and 24: Reporting accepted services. For each accepted service S, there are two different possibilities: either Q is the first child to accept to serve the service, or the service was already accepted by some child other than Q. In the first case, N sends the reply back to the original sender or the request, reporting that service S has been accepted (lines 14 and 29). Otherwise, some neighbor other than Q already accepted to serve service S (*i.e.*, an Individual in its Colony). Then, if the EXHAUSTIVE_REPLY parameter flag is set (either in the GBU or included in the original request message), N also reports the reply back. Consequently, in the EXHAUSTIVE_REPLY mode, every GCU that accepted to serve a given service will be reported back to the GCU that issued the request. Otherwise, for each service asked in the request, only one single servant GCU will be reported. Furthermore, it is easy to add more flexibility by including a threshold $T_r > 1$ on the number of replies. For example each GBU would report back T_r replies for the same service(s).

Lines 17: Finding other Individuals for rejected services. We now consider the case of rejected services. Call it S. This means that in Q's Colony, no potential Individuals serving service S could be found, or no Individuals accepted to serve it. Then, N has to find other neighbors that might contain Individuals for service S. Consequently, N calls again function SelectPeers, with the list of rejected services as input (line 17). The function works as previously explained, except that it does not consider the peers (including Q) that were already sent a particular service. Also, logically enough, the services that were previously accepted are ignored. Finally, the original sender of the request is not considered (*i.e.*, *ReturnPathId*). Note that in the case where the *exhaustive* search mode is used, then the list *SendList* returned by function SelectPeers may only contain a single pair $(\mathsf{L}, Serv(\mathsf{L}))$ (L is N's leader). Indeed, in the *exhaustive* search mode, all possible children in N's Colony have already been asked for all the services included in the request message, that they can

serve. Hence, rejected services are directly delegated to the leader L, if possible (*i.e.* if the latter was not the original sender of the request). As a result, the variable *SendList* contains a list of pairs $(\mathsf{P}, Serv(\mathsf{P}))$, where neighbor P is an Individual that can potentially serve the services in $Serv(\mathsf{P})$, *and* has not been sent a request for any of them yet.

Lines $18-20$: Forwarding request messages for rejected services. Consequently, for each pair $(\mathsf{P}, Serv(\mathsf{P}))$ included in *SendList*, N sends to neighbor P a service request message for services $Serv(\mathsf{P})$ (lines $18-20$).

Lines $21-23$ and 24: Service rejection. Finally, each service S included in the list of rejected services, and such that no additional potential Individual could be found amongst N's neighbors, is reported as rejected by N, to the original issuer of the request message (lines $21-24$).

3.2 Example

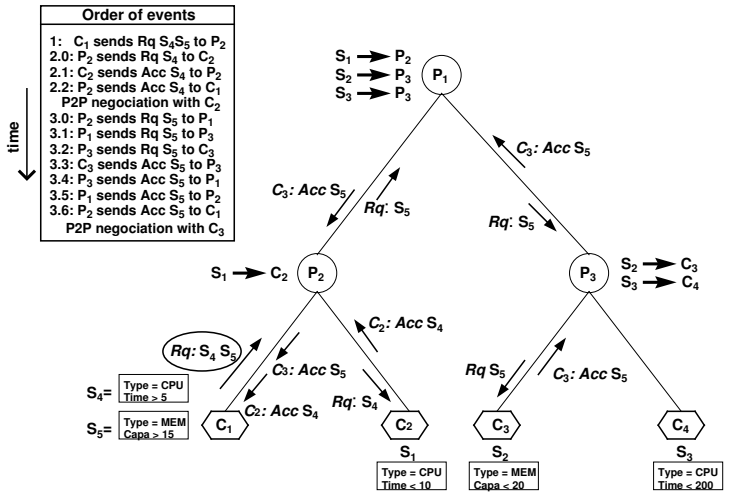

Figure 1: A simple Resource Discovery scenario. GCUs are represented in diamond shaped boxes, and GBUs in circles. The routing table of a GBU is represented next to it. The services registered by a GCU are shown below them. GCU C_1 has issued a service request for services S_4 and S_5. The paths followed subsequently by the different request and reply messages are highlighted by the arrows. The order in which the different messages are issued is shown in the table at the left hand side. Actions $2.*$ and $3.*$ occur in parallel with each others.

Consider the example illustrated in Figure 1. Three GBUs are represented, namely $\mathsf{P}_1 \cdots \mathsf{P}_3$, and 4 GCUs, namely $\mathsf{C}_1 \cdots \mathsf{C}_4$. GCUs C_1 and C_2 (resp. C_3 and C_4) have P_2 (resp.

P_3) as their leader, while P_1 is the leader of GBUs P_2 and P_3. GCUs C_2, C_3 and C_4 have registered services S_1, S_2 and S_3, respectively, and the routing tables of the upstream GBUs have been updated accordingly. In the example, resources are expressed as conjunctions of attribute/value pairs, and services are conjunctions of constraints on those attributes. We suppose that the *search mode* is set to *selective*, and we consider the scenario where GCU C_1 issues a service request for services S_4 and S_5, to its leader P_2. Since S_4 and S_1 overlap (any resource with $5 < Time < 10$ satisfies both S_1 and S_4), GBU P_2 forwards a service request for service S_1 to GCU C_2. Note that given that S_5 and S_1 do not overlap, S_5 is not included in the request. Since P_2 does not find any GCU potentially able to serve S_5 (*i.e.*, no services in its routing table overlap with S_5), it delegates it to its leader GBU P_1. When C_2 accepts to serve S_4, it sends a reply message with its Id and the accepted service S_4, back to GBU P_2, which, in turn, forwards it back to C_1. Then C_1 can directly negotiate the resource with C_2. When P_1 receives the service request for S_5, it forwards it to P_3 (since S_2 and S_5 overlap), which in turn forwards it to GCU C_3. When C_3 accepts to serve S_5, the same process then repeats as for GCU C_2. Eventually, C_1 receives a reply message with the Id of GCU C_3 and the accepted service, namely S_5. We have an illustration of the asynchronous communication (C_1 received the reply messages independently of each others) and the encapsulation of resources in Arigatoni (GBU P_2 only searched for service S_4 in its own Colony, *i.e.* GCU C_2).

3.3 Discussions

As said in the Introduction, in this paper, we mainly focused on the Resource Discovery mechanism used in Arigatoni. Total decoupling between GCUs in space (GCUs do not know each others), time (GCUs do not participate in the interaction at the same time), and synchronization (GCUs can issue service requests and do something else, or may be doing something else when being asked for services) is a major feature of Arigatoni. Another important property is the encapsulation of resources in colonies. Those properties play a major role in the scalability of Resource Discovery in Arigatoni.

As stated in Subsection 2.2, the subscription mechanisms of classical tree-based pub/sub systems [6, 7, 5, 4] can be used for the maintenance and update of consistent routing tables. Furthermore, as for the reliability of subscription advertisement, we can adapt the reliability mechanisms described in [7] to allow Arigatoni to be fault-tolerant or to adapt to dynamic topology changes.

The reliability of the Resource Discovery mechanism itself, although desirable, is of lesser importance, given the fact that service provision is not guaranteed at all in the Arigatoni model. In other words, when a GCU issues a service request, it is possible that no Individual is found for some of the services included in the request. This happens, for example, if those services were not declared by any GCUs in the system, or if all the GCUs that declared themselves as potential Individual refuse to serve them. However, at the cost of memory and bandwidth requirements, it is still possible to implement reliable Resource Discovery by using a reliable transmission protocol (TCP), an acknowledgment scheme in combination

with a retransmission buffer, and persistent data storage. This enhancement will be studied in a future work.

3.4 Load Balancing and Scalability

As defined above, GBUs are organized as a dynamic tree structure. Each GBU is a node of the tree, leader of its own SubColony and root of a subtree corresponding to the GBUs of its Colony. It is then natural to address scalability issues that arise from that tree structure. In the remaining of this section, we show that, under reasonable assumptions, the Arigatoni model is scalable. However, a complete performance evaluation is out of the scope of this paper and will rather be studied in a future work.

The most serious scalability issue in Arigatoni comes from the fact that high-level GBUs may receive large numbers of request messages, and may have a large number of children GBUs to forward request messages to. Although it is possible to limit the number of direct children that a GBU manages (during the registration phase, a GBU may refuse to be the leader of another, joining GBU), the main reason that Arigatoni is scalable comes from the encapsulation of resources in a GBU's Colony, *i.e.* a GBU always looks for Individuals in its own Colony first before delegating the request to its leader.

We call $T(t)$ the tree at time t, $d_T(t)$ its depth, *i.e.* the number of levels, and $l(i)$ the number of nodes in level i. We then call $\tau(i,j)$ the j^{th} node of level i of T ($0 \leq i \leq d_T$ and $0 \leq j \leq l(i)$). Remark that $T(i,j)$ will represent the subtree of T whose root is $\tau(i,j)$. Each $\tau(i,j)$ is the leader of a local Colony $c(i,j)$ and of a Colony $C(i,j) = \bigcup_{\tau(u,v)\in T(i,j)} c(u,v)$. Finally, we call $\gamma(i,j)$ the size of Colony $c(i,j)$, and $\Gamma(i,j)$ the size of $C(i,j)$, *i.e.* $\Gamma(i,j) = \sum_{\tau(u,v)\in T(i,j)} \gamma(u,v)$. In the following, let $c(i,j,k)$ with $k \in [0,\cdots,\gamma(i,j)-1]$ be the k^{th} GCU of $c(i,j)$. Figure 1(b) illustrates some of the notations that we have just introduced; the size of the Colony $C(1,0)$, which leader is $\tau_{1,0}$, is the sum of the sizes of all the local colonies in $T_{1,0}$, *i.e.* $c_{1,0}$, and $c_{2,0}$, and $c_{2,1}$: $\Gamma(1,0) = \gamma(1,0) + \gamma(2,1) + \gamma(2,0)$.

A typical scenario is the following: a service S is requested by $c(i,j,k)$ to its GBU $\tau(i,j)$ and processed thanks to the Arigatoni protocol. Let $X(u,v,w)$ be random variable associated to the fact that S can be served by $c(u,v,w)$, with probability $P(X(u,v,w))$. We assume that all $X(u,v,w)$ are independent, and we call $Q(X(u,v,w)) = 1 - P(X(u,v,w))$ the reverse probability. Hence, the probability that S *cannot* be served within Colony $c(i,j)$ is equal to:

$$\widehat{Q}(\widehat{X}(i,j)) = \prod_{k=0}^{\gamma(i,j)-1} Q(X(i,j,k))$$

It is very important to remark that the whole tree structure is dynamic and that all the entities are variable. For example, $\gamma(i,j)$ is not fixed. Hence, new members can join a Colony or old ones can leave.

In the remaining, we shall assume that we are in a steady state in the sense that, during the time a request is processed, the structure of the tree will not change. On one hand this

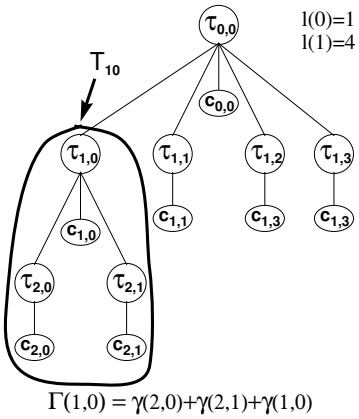

Figure 2: A simple topology with 7 GBUs and their local colonies. The size of the Colony $C(1,0)$, which leader is $\tau_{1,0}$, is the sum of the sizes of all the local colonies in $T_{1,0}$: $c_{1,0}$, $c_{2,0}$, and $c_{2,1}$: $\Gamma(1,0) = \gamma(1,0) + \gamma(2,1) + \gamma(2,0)$.

is reasonable since the communication time on the Internet is very low with respect to the protocol stack to log/delog GBU or GCU. This is not a completely satisfactory assumption since Arigatoni is mainly designed to handle large sets of colonies which evolve very rapidly. Nevertheless, we do believe that the gain of simplicity is much larger than the loss of accuracy. From this, we deduce that the probability that S cannot be served by $C(i,j)$ (the Colony whose leader GBU is $\gamma(i,j)$) is:

$$Q(X(i,j)) = \prod_{\forall(u,v),\tau(u,v)\in T(i,j)} \widehat{Q}(\widehat{X}(u,v)) = \prod_{\forall(u,v),\tau(u,v)\in T(i,j)} \prod_{w=0}^{\gamma(u,v)-1} Q(X(u,v,w))$$

Assume that $\gamma(u,v)$ is fixed during the request processing time. Then:

$$Q(X(i,j)) = \prod_{\Gamma(i,j)} Q(X(u,v,w))$$

Hence, $Q(X(i,j))$ is the probability that a request sent by $c(i,j,k)$ cannot be served by the Colony $C(i,j)$ of the GBU associated to $\tau(i,j)$. Counterwise, the probability that the request can be served is:

$$P(X(i,j)) = 1 - \prod_{\Gamma(i,j)} (1 - P(X(u,v,w))) \quad (1)$$

If $P(X(u,v,w))$ is constant equal to ρ, we get: $P(X(i,j)) = 1-(1-\rho)^{\Gamma(i,j)}$, and if ρ is small, then $P(X(i,j))$ can be approximated by $\Gamma(i,j).\rho$.

Let $P_h(X(i,j))$ be the probability that the request issued by $c(i,j,k)$ can be served by the predecessor (or ancestor) of $\tau(i,j)$ at level $i-h$, but cannot be served at a lower level. Let $\tau(i-h,j')$ be the corresponding node, and decompose $T(i-h,j')$ into $T_L \cup T_h$ where T_L is the maximal subtree of $T(i-h,j')$ containing $T(i,j)$. Hence, we have:

$$P_h(X(i,j)) = Q(T_L).(1-Q(T_h)) = \left(\prod_{\Gamma_L}(1-P(X(u,v,w)))\right).\left(1-\prod_{\Gamma_h}(1-P(X(u,v,w)))\right)$$

Remark that:

$$P_h(X(i,j)) \leq \prod_{\Gamma_h}(1-P(X(u,v,w))) \tag{2}$$

From (1), we deduce that the probability that a request issued by a member of a Colony is served by the Colony increases with the size of the Colony. From (2), we deduce that the probability for a request to be served at a lower level decreases along the level. From these 2 properties, we finally deduce the scalability of the GIP protocol.

4 Protocol Evaluation

To assess the effectiveness and the scalability of our Resource Discovery mechanisms, we have conducted simulations using large numbers of units and service requests.

Simulation Setup. We have generated a network topology using the transit-stub model of the Georgia Tech Internetwork Topology Models package [20], on top of which we added the Arigatoni Overlay Network. The resulting network topology, shown in Figure 3 contains 103 GBUs. GBU_2 (highlighted with a square in Figure 3) was chosen as the root of the topology.

GCUs were not directly simulated in the network topology. Instead, to simulate the population of GCUs, we added a GCU *agent* to each GBU in the system. The GCU agent of a GBU represents the local Colony of GCUs that are attached to that GBU as their leader.

We considered a finite set of resources $R_1 \cdots R_r$ of variable 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 GCU 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 scalability of Resource Discovery in our system.

To simulate GCU load, we then randomly added each service with probability ρ at each GCU agent, and had it registered via the registration service of Arigatoni. The routing tables

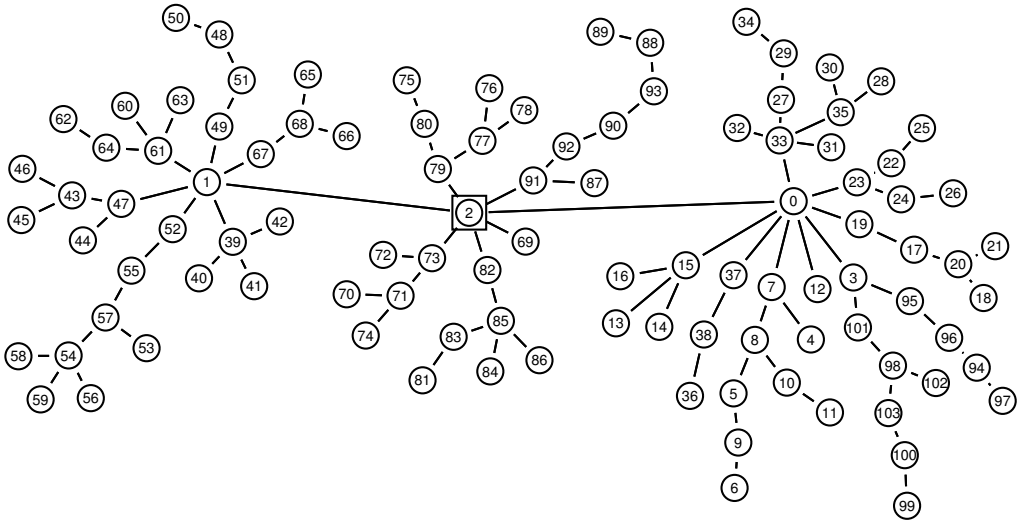

Figure 3: Simulated network topology with 103 GBUs

of the GBUs were updated starting at the initial GBU and ending at the root of the topology, GBU_2. In other words, it is as if each GBU has a probability ρ of having a GCU which registered service S_i, for any S_i. Thus, the parameter ρ can be seen as either the global availability of services, or as the density of population of GCUs (since the more the number of GCUs, the more likely it is that a given service is provided).

We then issued n service requests at GCU agents chosen uniformly at random. Each request contained one service[3], also chosen uniformly at random. Each service request was then handled by the Resource Discovery mechanism of Arigatoni described in Section 3. We used a service acceptation probability of $\alpha = 75\%$, which corresponds to the probability that a GCU 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.

The Resource Discovery algorithm 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. Upon completion of the n requests, we measured for each GBU its load as the number of requests (messages) that it received. We then computed the average load as the average value over the population of GBUs in the system. We also computed the maximum load as the maximum value of the load over all the GBU s in the system. Similarly, we computed the average and maximum load fractions as the average and max

[3]Service requests with k services can be seen as k service requests with one service.

Parameter	Description	Value
K	Number of GBUs	103
r	Size of services pool	128
ρ	Service availability	0.1% to 7%
α	Service acceptation probability	75%
n	Number of service requests issued	100 to 50000

Table 1: Parameters of the experiments

loads divided by the number of requests. The average load represents the average load of a GBU due to the completion of the n requests. The average load fraction represents the fraction of requests that a GBU served, in average. The maximum fraction represents the maximum fraction of the requests that a GBU served. Note that since a GBU receives at most one request message corresponding to a given service request, the average load fraction can be seen as the fraction of GBUs in the system involved in a service request, in average.

Finally, we computed the average service acceptation ratio as follows. For each GCU agent, 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 GCU agent. We then computed the average acceptation ratio as the average value over the number of GCU agents (that issued at least one service request).

We repeated the experiments for different values of ρ and n. Results are illustrated in Figure 4. Figure 4(a) and (c) were obtained with a fixed value of n of 50000 service requests.

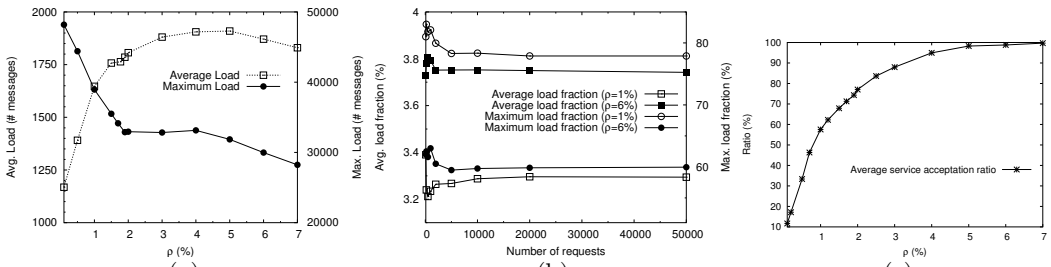

Figure 4: (a) Average and maximum load w.r.t. service availability ρ. (b) Average and maximum load fraction w.r.t. the number of requests issued. (c) Average service acceptation ratio w.r.t. service availability ρ.

Results and Interpretations. Figure 4(a) shows the evolution of the average and maximum load when varying the service availability ρ. The maximum load was obtained for GBU_2 or GBU_0, that are both very low-depth GBUs in the tree topology. It appears that the maximum load decreases with the service availability, while the average load increases. In

other words, the load is more evenly distributed amongst the GBUs in the system. This is due to the strategy of our Resource Discovery mechanism which consists in always searching for Individuals in its own Colony first before delegating to its leader. Indeed, as the service availability increases, GBUs have a higher chance to find Individuals in their own Colony. Hence, GBUs of high-depth in the topology participate more in the process of Resource Discovery, and GBUs of low-depth participate less. In other words, the Resource Discovery mechanism used in Arigatoni does not overload low-depth GBUs in the tree topology.

We observe for values of $2\% \leq \rho \leq 4\%$, a *plateau* in the curve of the maximum load, followed by a decreasing phase, but with a much lower slope than before ($\rho < 2\%$). This is due to the fact that for $\rho < 2\%$, GBU_2 has the maximum load in the system. For $\rho > 2\%$, however, GBU_0 takes over. This transition can be explained by the fact that for higher values of ρ, less messages are delegated to GBU_2. At some point ($\rho \sim 2\%$), the load of GBU_2 becomes less important than that of GBU_0, due to the high number of colonies that the latter manages. The constantness observed in the curve around that value is probably due to the fact that a transition phase is necessary for GBU_0 to be sensitive again to the increase of ρ. The following decreasing period with a lower slope corresponds to the fact that GBU_0 is less sensitive to an increase of ρ (indeed, GBU_0 is mostly concerned with the availability of services in its own colonies).

Finally, we observe that the average load stabilizes, which shows that the system scales to large number of GCUs (since as previously mentioned, the service availability ρ can be assimilated to the number of GCUs in the system).

Figure 4(b) shows the average and maximum load fractions w.r.t. the number of service requests. It appears clearly that Arigatoni scales to large numbers of requests. In fact, the average number of requests received by a GBU increases linearly with the total number of requests, at a rate of $\sim 3.5\%$. In other words, in average, a GBU only receives $\sim 3.5\%$ of the total number of requests. Equivalently, only 3.5% of the overall population of GBUs in the system participate in the process of discovering a particular resource, in average. Figure 4(b) also shows that low level GBUs in the topology are not particularly overloaded (the most overloaded GBU manages 60% of the overall load for $\rho = 6\%$). Finally, it corroborates the assertion that higher values of ρ favor the maximum load over the average load, *i.e.*, load balancing gets more effective.

Figure 4(c) shows that, unsurprisingly, the average service acceptation ratio increases exponentially with the availability of services. This shows that Arigatoni is efficient in searching Individuals for requested services. Indeed, a service availability of 4% enables the system to achieve an acceptation rate of 90%. In other words, the more the number of GCUs in the system, the more chances to find an Individual for a service request.

5 Conclusion

In this paper, we presented the Arigatoni lightweight communication model. We exposed in details the mechanisms that allow Arigatoni to offer dynamic and generic resource discovery. The main achievements are the complete decoupling between the different units in the

system, and the encapsulation of resources in local colonies, which enable Arigatoni to be potentially scalable to very large and heterogeneous populations. We are currently 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). We are also working on the implementation of a real prototype and the subsequent deployment on the PlanetLab experimental platform, and/or on GRID5000, the experimental 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.

Acknowledgment. The authors would like to thank Philippe Nain for its invaluable comments and interactions on the Arigatoni performance model. This work is supported by Aeolus FP6-2004-IST-FET Proactive.

References

[1] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R.E. Strom, and D.C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of ICDCS*, May 1999.

[2] D. Benza, M. Cosnard, L. Liquori, and M. Vesin. Arigatoni: Overlaying Internet via Low Level Network Protocols. Technical Report RR 5805, INRIA, November 2006. `http://www.inria.fr/rrrt/rr-5805.html`.

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

[4] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3), 2001.

[5] R. Chand. *Large scale diffusion of information in Publish/Subscribe systems.* PhD thesis, University of Nice-Sophia Antipolis and Institut Eurecom, 2005.

[6] R. Chand and P. Felber. A scalable protocol for content-based routing in overlay networks. In *Proceedings of NCA*, Cambridge, MA, April 2003.

[7] R. Chand and P.A Felber. XNet: A Reliable Content-Based Publish/Subscribe System. In *SRDS 2004, 23rd Symposium on Reliable Distributed Systems*, Florianopolis, Brazil, October 2004.

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

[9] M. Cosnard and L. Liquori. Virtual Organizations in Arigatoni. Manuscript, 2006.

[10] P. Th. Eugster, P. Felber, R. Guerraoui, and A.M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

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

[12] Dennis Heimbigner. Adapting publish/subscribe middleware to achieve gnutella-like functionality. In *SAC '01: Proceedings of the 2001 ACM symposium on Applied computing*, pages 176–181, 2001.

[13] 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.

[14] 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. full version in `http://www.cs.uchicago.edu/files/tr_authentic/TR-2002-06.pdf`.

[15] 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.

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

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

[18] V. Sassone. Global Computing II: A New FET Program for FP6. Talk, Bruxelles, 4/6/04. `http://www.cogs.susx.ac.uk/users/vs/research/paps/gc2InfoDayPres.pdf`.

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

[20] E.W. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proceedings of INFOCOM 1996*, San Francisco, March 1996.

A GRID Scenario for Seismic Monitoring, from [2]

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/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.

Aftermath, the processed data should be analyzed using a *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.

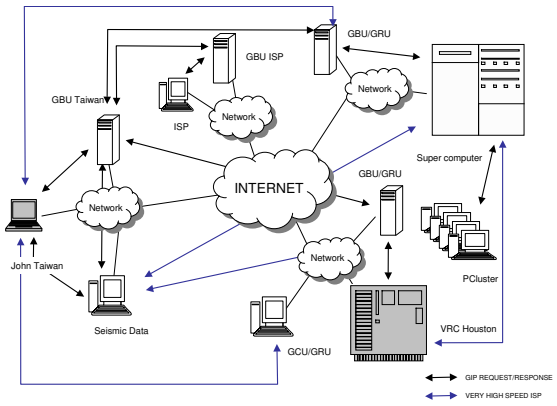

Figure 5: A GRID Scenario for Seismic Monitoring

Hence he would like the following scenario to happen:

- 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;
- 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 processed;

- John will ask also 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;
- John will ask also to the Global SuperComputer unit the task of collecting all intermediate results so calculating the final result of the computation (*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 5 (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). All GBU's are trusted[4], making *de facto* in common all resources of their colonies.

[4] As a simpler approximation *à la* Globus, all GBU s share the same PKI.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399