Logical Networks: Towards Foundations for Programmable Overlay Networks and Overlay Computing Systems

Luigi Liquori and Michel Cosnard

INRIA, France
[Luigi.Liquori,Michel.Cosnard]@inria.fr

Abstract. We propose and discuss foundations for *programmable overlay networks* and *overlay computing systems*. Such overlays are built over a large number of distributed *computational individuals*, virtually organized in *colonies*, and ruled by a leader (*broker*) who is elected or imposed by system administrators. Every individual asks the broker to log in the colony by declaring the resources that can be offered (with variable guarantees). Once logged in, an individual can ask the broker for other resources. Colonies can recursively be considered as *evolved individuals* who can log in an outermost colony governed by another (super)-broker. Communications and routing intra-colonies goes through a broker-2-broker PKI-based negotiation. Every broker routes intra- and inter- *service requests* by filtering its *resource routing table*, and then by forwarding the request first inside its colony, and second outside, via the proper super-broker (thus applying an *endogenous-first-estrogen-last* strategy). Theoretically, queries are formulæ in first-order logic equipped with a small program used to *orchestrate* and *synchronize* atomic formulæ. When the client individual receives notification of all (or part of) the requested resources, then the real resource exchange is performed directly by the server(s) individuals, without any further mediation of the broker, in a pure peer-to-peer fashion. The proposed overlay promotes an *intermittent* participation in the colony, since peers can appear, disappear, and organize themselves dynamically. This implies that the routing process may lead to *failures*, because some individuals have quit, or are temporarily unavailable, or they were logged out *manu militari* by the broker due to their poor performance or greediness. We design, validate through simulation, and implement these foundations in a programmable overlay computer system, called Arigatoni.

"Computer is moving on the edge of the Network..."
[Jan Bosch, Nokia Labs, Keynote ARCS, LNCS 4415, 2007]

1 Introduction

The explosive growth of the Internet gives rise to the possibility of designing large *overlay networks* and *virtual organizations* consisting of Internet-connected

computers units, able to provide a rich functionality of services that makes use of aggregated computational power, storage, information resources, etc. We would like to start this paper with the standard definition of *Computer System* (we emphasize some text using underline).

Definition 1 (Computer System).
A computer system is composed by a computer hardware and a computer software.

- *A Computer Hardware is the physical part of a computer, including the digital circuitry, as distinguished from the computer software that executes within the hardware. The hardware of a computer is infrequently changed, in comparison with software and data.*
- *A Computer Software is composed by three parts, namely, system software, program software, and application software.*
  - *The System Software helps run the computer hardware and computer system. Examples are operating systems (OS), device drivers, diagnostic tools, servers, windowing systems...*
  - *The Program Software usually provides tools to assist a programmer in writing computer programs and software using different programming languages. Examples are text editors, compilers, interpreters, linkers, debuggers for general purpose languages...*
  - *The Application Software allows end users to accomplish one or more specific (non computer related) tasks industrial automation, business software, educational software, medical software, databases, computer games...*

Starting from the previous basic skeleton definition, we elaborate our vision of what an *Overlay Computer System* is. The reader can focus on the tiny but crucial differences between the above and below definitions.

Definition 2 (Overlay Computer System).
An overlay computer system is composed by an overlay computer hardware and an overlay computer software.

- *An Overlay Computer Hardware is the physical part of an overlay computer, including the digital circuitry, as distinguished from the overlay computer software that executes within the hardware. The hardware of an overlay computer changes frequently and it is distributed in space and in time. Hardware is organized in a network of collaborative computing individuals connected via* IP *or ad-hoc networks; hardware must be negotiated before being used.*
- *An Overlay Computer Software is composed by three parts, namely, overlay system software, overlay program software, and overlay application software.*
  - *The Overlay System Software helps run the overlay computer hardware and overlay computer system. Examples are network middlewares playing as a distributed operating systems(dOS), resource discovery protocols, virtual intermittent protocols, security protocols, reputation protocols...*
  - *The Overlay Program Software usually provides tools to assist a programmer in writing overlay computer programs and software using different overlay programming languages. Examples are compilers, interpreters, linkers, debuggers for workflow-, coordination-, and query-languages.*

- *The Overlay Application Software allows end users to accomplish one or more specific (non-computer related) tasks industrial automation, business software, educational software, medical software, databases, and computer games... Those class of applications deals with computational power (*Grid*), file and storage retrieval (*P2P*), web services (*Web2.0*), band-services (*VoIP*), computation migrations...*

The Arigatoni overlay network computer, designed and developed since 2006 at INRIA, is a structured multi-layer overlay network which provides resource discovery with variable guarantees in a virtual organization where peers can appear, disappear, and organize themselves dynamically. Arigatoni is mainly concerned with how and where resources are declared and discovered in the overlay, allowing agent computers to make secure use of agent aggregated computational power, storage, information resources etc. We anticipate, in a nutshell, the key functional units of Arigatoni (discussed in details later on).

- An *Agent Computer (*AC*)* is the basic computational entity of the overlay: it is typically a device, like a PDA, a laptop, a PC, or smaller devices, connected through IP or other *ad hoc* communication protocols in different fashions (wired, wireless, etc.).
- An *Agent Broker (*AB*)* is devoted to (un)subscribing ACs, to receiving service queries from clients, to contacting and negotiating with potential servers, to authenticating clients and servers, and to routing requests. An AB is the leader of the colony of the ACs and of the sub-colonies that is manages. Intra-colony communication is initiated through the leader AB, while inter-colonies communication is initiated through a chain of (PKI-based) AB-2-AB message exchanges. The rationale ensuring scalability is that every request is handled first inside its colony, and then forwarded through the proper super-leader (thus applying an *endogenous-first-estrogen-last* strategy). In both cases, when a client AC receives an acknowledgment of a service request from the direct leader AB, then the AC is served directly by the server(s), *i.e.* without a further mediation of the AB, in a pure P2P fashion. Logically, an AC can be seen as a *collapsed colony*, or a *broker managing itself*.
- An *Agent Router (*AR*)* is the basic unit close to ACs and ABs that is devoted to sending and receiving packets of the resource discovery and the virtual intermittent protocols (see below) and to forwarding the "payload" to the units which are connected with this router. The connection AB-AR-AC is ensured via a suitable API.

The total decoupling between peers in *space* (peers do not know other peers' locations), *time* (peers do not participate in the interaction at the same time), *synchronization* (peers can issue service requests and do something else, or may be doing something else when being asked for services), and *encapsulation* (peers do not know each other) are key features of Arigatoni's scalability.

Summarizing, the main challenges in Arigatoni lie in the management of an overlay network with a dynamic topology, the routing of queries, and the discovery of resources in the overlay. In particular, resource discovery is a non-trivial

problem for large distributed systems featuring a discontinuous amount of resources offered by agent computers and their intermittent participation in the overlay. For more technical details on the Arigatoni overlay network, the interested reader can have a look on [CLC07b,CCL06,CLC07a,CCL08].

Therefore, the main contributions of this paper are:

- to provide adequate notions and definitions of a programmable overlay network computer;
- on the basis of these definitions, to propose a precise architecture of a programmable network computer;
- to provide insight of the architecture by putting emphasis on technical problems, security, social, implementations, and related issues;
- to summarize and collect previous efforts by the authors on Arigatoni into one reference paper easy to read.

1.1 Virtual Organizations

Computational units 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 agent computers, or sub-colonies. Every colony has *exactly* one leader and at least one individual (the leader itself), and a colony contains only individuals.

Agent computers communicate by first registering to the colony and then by asking and offering services. The leader agent broker analyzes service requests/responses, coming from its own colony or arriving from a surrounding colony, and routes requests/responses to other individuals. Individuals get in touch with each other without any further intervention from the system, in a P2P fashion. Peers' coordination is achieved by a simple program written in an orchestration/business language *à la* BPEL [IBM], or JOpera [Pau].

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 strategy, reminiscent of the Roman *do ut des*, is nowadays called, in Game Theory, Rapoport's *tit-for-tat* strategy [Rap63] of cooperation based on reciprocity. Tit-for-tat 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 stability (*i.e.* balanced uploads and downloads) policy of the Bittorrent P2P protocol [Bit].

Once an agent computer has issued a request for some services, the system finds some agent computers (or, recursively, some sub-colonies) that can offer the resources needed, and communicates their identities to the (client) agent 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 (agent computer or sub-colony) to login/logout in/from a colony. This essentially means

that the process of routing request/responses may lead to failure, because some individuals logged out or because they are temporarily unavailable (recall that individuals are not slaves). This may also lead to temporary 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.

Trees vs. graphs: a conflict without a cause. In the first versions of Arigatoni, the network topology was tree- or forest-based. But since AC are not slaves, multiple registrations are in principle possible and unavoidable. This weaves the network topology to a *dynamic graph*. As an immediate consequence, Arigatoni's protocols deal with multiple registrations of the same individual in different colonies, with the natural consequence of resource overbooking, routing table update loops, and resource discovery loops (when a resource request comes back to the individual that generates the request itself), see [LC07].

As an example of resource overbooking, suppose an AC registers to two colonies, by declaring and offering the same resource S twice, *i.e.* once for each colony. This phenomenon is well known in the telecommunications industry, such as in the "frame-relay" world. For the record, overbooking in telecommunications means that a telephone company has sold access to too many customers which basically flood the telephone company lines, resulting in an inability for some customers to use what they purchased. Other examples of overbooking can be found in the domain of transportation and hotel reservations.

1.2 User application independence, parametricity, universality

Dealing only with resource discovery has one important advantage: the complete generality and independence of any offered and requested resource. Thus, Arigatoni can fit with various scenarios in the agent 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 agent computer, the latter being useful in case of catastrophic scenarios, like fire, terrorist attack, earthquake, etc., in the vein of agent programming languages *à la* Obliq [Car95] or Telescript [Whi94]. We could envisage at least the following scenarios to be a tight fit for our model (list not exhaustive):

- Ask for computational power (*i.e.* the Grid);
- Ask for memory space (*i.e.* distributed storage);
- Ask for bandwidth (*i.e.* VoIP);
- Ask for a distributed file retrieving (*i.e.* standard P2P applications);
- Ask for a (possibly) distributed web service (*i.e.* query *à la* Google or any service available via web-oriented protocols);
- Orchestration of a distributed execution of an algorithm;
- Ask for a computation migration (*i.e.* transfer one partial run in another agent computer, saving the partial results;

– Ask for a *human computer interaction* (the human playing the role of an individual);
– ...

Thus, Arigatoni is *parametric* or *universal* in the sense of universal Turing machine, or *generic* as the von Neumann computer architecture. In one sentence: "Arigatoni is the first fully programmable overlay network computer".

2 Functional units and protocols in Arigatoni

2.1 Functional units

The Agent Computer (AC). This unit can be, *e.g.*, a cheap computer device composed by a small RAM-ROM-HD memory capacity, a modest CPU, a ≤ 40 keystrokes keyboard (or touchscreen), a tiny screen (≤ 4 inch), an IP or *ad hoc* connection (via DHCP, BLUETOOTH, WIFI, WIMAX...), an USB port, and very few programs installed inside (one simple editor, one or two compilers, a `mail` client, a mini browser...)[1]. Of course an AC can be a supercomputer, or an high performance PC-cluster, a large database server, an high performance visualizer (*e.g.* connected to a virtual reality center), or any particular resource provider, even a *smart-dust*. The operating system (if any) installed in the AC is not important. The computer should be able to work in *local mode* for all the tasks that it could do locally, or in *global mode*, by first registering itself to one or many colonies of the overlay, and then by asking and serving global requests via the colony leaders. In a nutshell, the tasks of an AC are:

– Discover the address of one or many ABs, playing as colony leaders, upon its arrival in a "connected area";
– Register on one or many ABs, so entering *de facto* the Arigatoni's virtual organization;
– Ask and offer some services to others ACs, via the leaders ABs;
– Connect directly with others AC in a P2P fashion, and offer/receive some services. Note that an AC can also be a resource provider. This symmetry is one of the key features of Arigatoni. For security reasons, we assume that all AC come with their proper PKI certificate.

The Agent Broker (AB). This unit can be, *e.g.*, a computer device made by an high speed CPU, an IP or *ad hoc* connection (via DHCP, BLUETOOTH, WIFI, WIMAX...), an high speed hard-disk with a *resource routing table* to route queries, and an efficient program to match and filter the routing table. The computer should be able to work in *global mode*, by first registering itself in the overlay and then receiving, filtering and dispatching global requests through the network. The tasks of an AB are:

[1] Our favorite device actually is the Internet terminal Nokia N810 [Nok].

- Discover the address of another *super*-AB, representing the *super-leader* of the *super-colony*, where the AB colony is embedded. We assume that every AB comes with its proper PKI certificate. The policy to accept or refuse the registration of an individual with a different PKI is left open to the level of security requested by the colony;
- Register/unregister the proper colony on the *leader* AB which manages the super-colony;
- Register/unregister clients and servants AC in its colony, and update the internal resource routing table, accordingly;
- Receive the request of service of the client AC;
- Discover the resources that satisfy an AC request in its local colony, according to its resource routing table;
- Delegate the request to an AB leader of the direct super-colony in case the resource cannot be satisfied in its proper colony. Prior to this, it must register itself (and byproduct its colony) to another super-colony;
- Perform a combination of the above last two actions;
- Deal with all PKI intra- and inter-colony policies;
- Notify, after a fixed *timeout period*, or when all individuals failed to satisfy the delegated request, to the AC client the *denial of service* requested by the AC client;
- Send all the information necessary to make the AC client able to communicate with the AC servants. This notification is encoded using the resource discovery protocol. (Finally, the AC client will directly talk with the ACs servants).

*The Agent Router (*AR*).* This unit implements all the low-level overlay network routines, providing access to the underlay network. In a nutshell, an AR is a shared library dynamically linked with an AC or an AB. The AR is devoted to the following tasks:

- Upon the initial start-up of an AC (resp. AB) it helps to register the unit with one or many AB that it knows or discovers;
- Checks the well-formedness and forwards packets of the two Arigatoni's protocols across the overlay toward their destinations;

2.2 The Resource Discovery Protocol (RDP)

Kind of resource discovery. The are mostly three mechanisms of resource discovery in Arigatoni, namely:

- The process of an AB to find and negotiate resources to serve an AC request in its own colony;
- The process of an AC (resp. AB) to discover an AB, upon physical/logical insertion in a colony;

The first discovery is processed by the resource discovery protocol, while the second is processed out of the Arigatoni overlay, using well known network technologies like DHCP [AD97], DNS [GVE00], BLUETOOTH, WIFI, WIMAX...

The current RDP protocol version allows the request for *multiple services* and *service conjunctions*. Adding service conjunctions allows an AC to offer several services at the same time. Multiple services requests can be also asked to an AB; each service is processed sequentially and independently of others. As an example of multiple instances, an AC may ask for three CPUs, *or* one chunk of 10GB of HD, *or* one gcc compiler. As an example of a service conjunction, an AC may ask for another AC offering *at the same time* one CPUs, *and* one chunk of 1GB of RAM, *and* one chunk of 10GB of HD, *and* one gcc compiler. If a request succeeds, then, using a simple orchestration language, the AC client can use all resources offered by the servers ACs.

The RDP protocol proceeds as follows: suppose an AC X registers – using the intermittent protocol VIP presented below – to an AB and declares its availability to offer a service S, while another AC Y, already registered, issues a request for a service S'. Then, the AB looks in its *routing table* and *filters* S' against S. If there exists a solution to this filter equation, then X can provide a resource to Y. For example, the resource $\mathsf{S} = [\mathsf{CPU}{=}\mathsf{Intel}, \mathsf{Time}{\leq}\mathsf{10sec}]$ filters against $\mathsf{S}' = [\mathsf{CPU}{=}\mathsf{Intel}, \mathsf{Time}{\geq}\mathsf{5sec}]$, with attribute values Intel and Time between 5 and 10 seconds.

2.3 Inside routing tables for resource discovery

Each AB maintains a *routing table* $\mathcal{T}$ locating the *services* that are registered in its colony. The table is updated according to the *dynamic registration and unregistration* of ACs in the overlay; thus, each AB maintains a partition of the data space. When an AC asks for a resource (service request), then the query is *filtered* against the routing tables of the ABs where the query is arrived and the AC is registered; in case of a *filter-failure*, the ABs forward the query to their direct super-ABs. Any answer of the query must follow the reverse path.

Thus, resource look-up overhead reduces when a query is satisfied in the current colony. Most structured overlays guarantee look-up operations that are logarithmic in the number of nodes. To improve routing performance, caching and replication of data and search paths can be adopted. Replication also improves load balancing, fault tolerance, and the durability of data items.

Every AC registers in the colony with a *tuple* of (services,instances), like $\mathsf{SREG}{:}[(\mathsf{S}_i, n_i)]^{i=1\ldots h}$, and asks for a another tuple of (service,instances), like $\mathsf{SREQ}{:}[(\mathsf{S}_j, n_j)]^{j=1\ldots k}$. Each service is looked-up sequentially and independently of others, by wrapping a unitary resource discovery inside a for-loop:

$$\textbf{for each } j = 1 \ldots k \textbf{ do } -\text{find service } \mathsf{S_j} - \textbf{ end foreach}$$

An atomic service request may also have the shape $\mathsf{SREQ}{:}[((\bigwedge_{i=1\ldots n} \mathsf{S}_i), m)]$, *i.e.* the system is no longer asked to find m occurrences of a single service, but rather m occurrences of a conjunction of n services. That is, the system has to look for m distinct ACs, each AC being able to provide all the services in $\bigwedge_{i=1\ldots n} \mathsf{S}_i$.

For a given resource S, one entry in the routing table has the form $\mathcal{T}[\mathsf{S}] = [(\mathsf{P}_j, m_j)]^{j=1\ldots k}$, where $(\mathsf{P}_j)^{j=1\ldots k}$ are the addresses of the *direct children in the*

AB*'s colony*, and $(m_j)^{j=1...k}$ are the instances of S available at P_j. Intuitively and for an atomic service request SREQ:[(S, n)], the most economic resource discovery routing steps performed by an AB are:

1. Look in the table $\mathcal{T}$ for all *distinct* q ACs able to provide S in the local AB's colony;
2. If $n \leq q$, then search n resources first inside the current colony (and, recursively, in sub-colonies), and finally delegate to the AB's super-leaders all the denied resources.
3. If $n \geq q$, then search q resources inside the colony (and, recursively, in sub-colonies), and finally delegate all the $n-q$ remaining instances to the AB's super-leader.

Pragmatically speaking this strategy, reminiscent of the object-oriented "method-lookup algorithm", pushes always first queries on the leafs of the overlay in order to avoid, if possible, routing bottlenecks.

An AC receiving a service request first chooses the services that it accepts or denies to serve; then, it generates a SRESP message containing the lists of accepted or rejected services, and finally sends it to its AB. The response messages are then propagated back in the overlay, following the reverse path.

2.4 The Virtual Intermittent Protocol (VIP)

Peers' participation in Arigatoni's colonies is managed by the *Virtual Intermittent Protocol (*VIP*)*; the protocol deals with the *dynamic topology* of the overlay, by allowing individuals to login/logout to/from a colony (using the SREG message). Due to this high node churn, the routing process may lead to *failures*, because some individuals have logged out, or because they are temporarily unavailable, or because they have logged out *manu militari* by the broker for their poor performance or greediness. In the VIP protocol, there are two ways an individual can register to an AB (sensible to its physical position in the network topology), the latter being not enforced in Arigatoni:

1. Registration of an individual to an AB belonging to the same *current administrative domain*;
2. Registration, via *tunneling*, of an individual to another AB belonging to a *different administrative domain*.

If both registrations apply, then the individual is working *de facto* both in local mode *in the current administrative domain*, and in global mode *in another administrative domain*. Symmetrically, an individual can unregister according to the following simple rules *"d'étiquette"*:

– Unregistration of an individual is allowed only when there are no pending services demanded or requested to the leader AB of the colony: individual must always wait for an answer of the AB or for a direct connection of the AC requesting or offering the promised service, or wait for an internal timeout (the time-frame must be negotiated with the AB);

– (As a corollary of the above) an AB cannot unregister from its own colony, *i.e.* it cannot discharge itself. However, for fault tolerance purposes, an AB can be faulty. In that case, the ACs unregister one after the other and the colony disappear;
– Once an AC has been disconnected from a colony, it can physically migrate in another colony belonging to any other administrative domain;
– Selfish nodes in P2P networks, called "free riders", that only utilize other peers' resources without providing any contribution in return, can be fired by a leader; if the leader of a colony finds that an individual ratio of fairness is too small ($\leq \epsilon$, for a given ϵ), it can arbitrarily decide to fire that individual without notice. Here, the VIP protocol also checks that the individual has no pending services to offer, or that the timeout of some promised services has expired, the latter case means that the free rider promised some services but finally did not provide any service at all (not trustfulness).

In both cases of node (un)registration, a *service update* SUPD message will be flooded in the brokers' network in order to keep resource tables as much updated as possible; thus, high node churn leads to message overhead in the overlay.

2.5 Inside routing tables for intermittent participation

As said before, routing tables denoting the set of resources are stored in AB's. An individual (AC or AB representing a sub-colony) registers to a colony with a tuple of (services,instances), like in SREG:$[(\mathsf{S}_i, n_i)]^{i=1\ldots h}$. If a broker AB accepts an individual in its colony, then it sends a service update, written SUPD:$[(\mathsf{S}_i, n_i)]^{i=1\ldots h}$, to its direct super-broker AB$'$ in order to communicate the availability of the new resources in its colony, by an update of the routing table $\mathcal{T}'$ of AB$'$. This message is then propagated from broker to broker until the root (if any) of the multi-layer overlay is reached. This means a high node churn forces routing tables to be *faulty* until all service updates are properly propagated. As such, service registration in an overlay network computer is an activity that must be taken seriously into account [CLC07b].

The first Arigatoni network topology was tree-based. In [LC07], the authors make a significant step by moving from a tree-based network topology to a more general graph-based one. As an immediate consequence of this move, the Arigatoni VIP protocol must be reconsidered in order to take into accounts routing loops when updating routing tables.

3 Social model, security, trust and implementation issues

3.1 The social model underneath Arigatoni

The Arigatoni overlay network computer defines mechanisms for devices to inter-operate by offering services, in a way that is reminiscent to Rapoport's *tit-for-tat* strategy of co-operation based on reciprocity. This way to understand common behavior of virtual organizations has some theoretical basis on Game Theory

[Rap63]. Classical results from game theory are based on the assumption that a shared amount of resources is available, and then users have an incentive to collaborate. The very first design of Arigatoni forced each AC to register to only one AB, but the architecture can be smoothly scaled up to a more general topology where each AC may simultaneously be registered to several AB, and where a *colony* is just one possible *social scheme*.

This means that Arigatoni fits with motivations and cooperation behavior of different communities. It tries to be *policy neutral*, leaving policy choices for each node at the implementation or configuration level, or at the community or organization level. Policy domains can overlap (one node can define itself as belonging "much" to colony *foo* and "a little bit" to colony *bar*). This denotes a decentralized non-exclusive policy model.

One question can arise: who is Arigatoni designed for? We believe the overlay is flexible enough to serve a mix of different "social structures" and "end-users":

- Independent end-user connecting through his ISP or migrating from hot-spot to hot-spot;
- Cooperative communities of disseminated individuals;
- More regulated or hierarchical communities (maybe a better picture of a corporate network);
- Cooperative or competitive resource providers and resource brokers.

The Arigatoni overlay network computer is suitable to support various extended trust models. Moreover, reputation score could be expanded to a multi-dimensional value, for example adding a score for the *quality of the service* offered by an individual. Moreover, Arigatoni encourages cooperation and enables gratuitous resource offering. But it may also suit for business extensions, *e.g.*:

- An individual can sell resource usage, creating a resource business;
- An AB can sell a resource discovery service, creating a brokering business (*"I point you to the best resources, more quickly than anyone else"*).

The Arigatoni overlay network computer is suitable of a number of service extensions: among others, *e.g.*:

- How to create and call third party services for on-line payment of services;
- How to exchange digital cash for payment of services;
- How to negotiate service conditions between client and servant, including price and quality of service.

The one-to-many nature of the RDP protocol service request (SREQ) are of particular interest in this case.

Another possible Arigatoni extension may define how to join a third party auction server. Candidate servants for a SREQ would contact the auction server and make their bid. The trusted auction server chooses the elected candidate and service conditions based on auction terms. The individual client would then contact the auction server and get this information. Those extensions may take advantage of the RDP optional fields [BCLV06], for example, to transmit location and parameter information to call a third party system.

3.2 Trust, security, and implementation issues

In order to work securely, the Arigatoni overlay network computer needs to be able to offer the following guarantees to its components:

- The communication between two individuals must be secured;
- The role played by a node (*i.e.* client AC, servant AC or AB) must be certified by a third party trusted by the nodes which have to communicate with this particular node. A way to implement those constraints is to use PKI certificates. A *Certification Authority* delivers certificates, and couples of private and public keys for ACs and ABs which attest of their distinctive roles. The whole mechanisms involved by a PKI is out of the scope of this paper, but good use of PKIs and an implementation compliant with RFC2743 [Lin00] can provide all the necessary security, namely the trustfulness on the identity of the peers, and the trustfulness of all the transmitted data, *i.e.* secrecy, authenticity, and integrity.
- In addition to PKIs, a more "liquid" trust model could be built, based on *reputation mechanisms* [WV03]. Reputation represents the amount of trust an individual in the overlay has in another individual based on its partial view. In a nutshell:
  - Each individual maintains a reputation score for each individual it knows;
  - Each individual maintains a reputation score for each resource it serves;
  - Exchanges between individuals update dynamically each other's scores;
  - Conflict between two or many individuals are resolved by the brokers leaders of the colonies to which individuals belong;
  - The computation of the reputation score (a trust metrics) and the way individuals exchange scores is left free to each single implementation.

A last word on implementation issues of the Arigatoni overlay network computer: it is well known that two technical barriers are commonly used to block transmission over IP network in overlays, namely:

- *Firewalls* to drop UDP flows (usually considered as suspects);
- NAT techniques to mask to the outside world the real IP addresses of inside hosts; a NAT equipment changes the IP source address when a packet *goes to* outside, and it changes the IP destination address when a packet *comes from* outside.

The usage of these mechanisms is very frequent on the Internet, and barriers exist to prevent connections between *inside* and *outside* nodes in the Arigatoni overlay. RFC3489 [RWHM03] can be used to overcome such obstacles.

4 Related work, applications, and conclusions

4.1 Discussion on overlay topologies

Many technologies, algorithms, and protocols have been proposed recently for resource discovery . Some of them focus on Grid or P2P oriented applications, but

none of those targets the full generality as the Arigatoni overlay network computer does. Indeed, Arigatoni deals with generic resource discovery for building an overlay network of ABs and ACs, structured in a virtual organization of variable topology, with clear distinct roles between leader ABs and individuals.

In an overlay network, any message is routed through the full overlay; as such, the topology adopted in the overlay strongly affects routing protocols and their complexity. The overlay is built on top of the physical one, and, thus, two neighbor nodes in the overlay network may be many links apart in the physical network. The first Arigatoni network topology was a dynamic *hierarchical n-layer tree*, but a recent work raise Arigatoni to a graph topology [LC07].

To implement resource look-up, structured overlays map (key of) data item to nodes (our ABs). Hence, the mapping is usually done through hashing the key space of the data item to the `id` in the node space. In the literature, *e.g.* [AEO06], there are essentially the following types of overlays: structured (tree, ring, or grid), unstructured (graphs), hybrid overlays (a combination of the two above), and multi-layer (or n-layer) overlays.

Arigatoni falls mostly in the category of multi-layer. In a nutshell, in a n-layer overlay network, the responsibility assigned to individuals differs (think of the different roles between ABs and ACs), since super-peers (ABs) serve as a server for a subset of all peers. Ordinary peers (ACs) submit queries to their super-peers and receive results from it. Super-peers are also connected to each others; they route messages over the overlay network, submit, delegate, and answer queries on behalf of their sub-peers. This structure is replicated *recursively*, creating a *n-layer topology*, where peers become super-peers with decreasing responsibilities. The possibility of having a graph of super-peers complicates routing, registration protocols and resource table update.

Typical issues in n-layer overlays are the size of each colony (load balancing of the colony), and the internal coherence of the resources offered and requested by each colony (homogeneity of the colony). Typical bottlenecks of n-layers are reliability, service availability (related to few points of failure), and load balancing. Classical solutions to cope with these problems are adding redundancy at the broker-layer. Historically, the most related tree topologies are BATON [JV05] and P-GRID [Abe01], whereas the closest n-layer topologies are the one of CANON [GKGM04] and CORAL [FM03].

- (BATON) is a balanced binary tree that features a left and a right routing table, both contained in each node (denoted by a single logical `id`). Nodes may join or leave the network at any time, provided the tree remains balanced. The node receiving a join can forward the join towards a node which has less children or which is a leaf node. This implies that an AC can become an AB. Leaving the network is constrained to not breaking the balanced tree unless finding a substitute. As such, load balancing can be costly.
- (P-GRID) is a distributed dynamic binary search tree, such that the search space is partitioned between peers. The salient feature of P-GRID is the separation of concerns between `id` and its position in the network. All peers maintain a partial routing table of the search space, that is *negotiated* be-

teen the closest peers. Multiple peers can be responsible for the same path, resulting in non uniqueness of routing and robustness under peer failure.
- (CANON) is a multi-layer overlay where routing is based on a hierarchical *distributed hash tables* (DHT). As in Arigatoni, the search space is partitioned into *domains*; in contrast, routing inside a domain is DHT-based, and topology is static.
- (CORAL) is another hierarchical DHT. The search space is partitioned into three *clusters*, based on latency; a regional cluster, a continental cluster and a planet-wide cluster. It also comes with algorithm for self-organizing, merging and splitting clusters, to ensure acceptable diameters.

4.2 Discussion on closest technologies

The GLOBUS toolkit [Glo], is an open-source set of technology, protocols and middleware, used for building Grid applications (sharing computing power, distributed databases, etc.). The toolkit includes stand-alone software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. The analogies with the Arigatoni overlay network computer are in the *Community Scheduler Framework* component and the *Web Service Grid Resource Allocation and Management*, concerning the resource discovery, and the *Teleoperations Control Protocol* concerning the way units cooperate (in analogy with the RDP protocol and with orchestration languages). Hovewer, GLOBUS does not target the full generality of the Arigatoni overlay network computer, that, thanks to its generic resource discovery, is suitable for pervasive, ubiquitous overlay computations in addition to pure Grid-oriented applications.

Promoted by Sun, the JXTA [JXT] technology is a set of open peer-to-peer protocols enabling 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 Arigatoni is the design of protocols for generic resource discovery and intermittent participation, while the main concern of the JXTA technology is to offer some tools to implement a P2P model. In addition, Arigatoni focuses on the evolution/devolution of colonies, while JXTA technology allows peers to communicate using an already existing overlay network of peers. Further, Arigatoni's aim is the dynamicity of the overlay network, while JXTA's is the freedom of connectivity between peers. Finally, JXTA-peers come with their proper JXTA-`id` (logical JXTA peers addressing), while Arigatoni relies on the more conventional IP addresses.

Publish/subscribe (pub/sub) [EFGK03] is a communication paradigm for asynchronous dissemination of information. Consumers subscribe to the system (typically called the *Notification Service*) to specify the type of information that they are interested in. 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 data and the interests declared by the consumers. Many pub/sub systems have been developed recently, such as XNET [CF04],

SIENA [CRW01] or GRYPHON [BCM+99]. Banavar *et al.*, in [Hei01], propose to adapt the SIENA publish/subscribe system to achieve GNUTELLA-like resource discovery. Their work resembles ours in the sense that Arigatoni is also inspired by the pub/sub paradigm. However, resource discovery in pub/sub 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 look-up.

Worthy also to notice the OSGi technology [OSG], a component integration platform with a service-oriented architecture and life cycle capabilities that enable dynamic delivery of services. These capabilities greatly increase the value of a wide range of computers and devices that use the Java platform. The OSGi specifications provide a platform for an universal middleware.

4.3 Challenges

We envision a long term meta-application and a medium-term specific-application.

Challenge 1: from large-scale computing machines to large-scale overlay network computing machines

This challenge is inspired by the seminal talk by John von Neumann, given in May 1946, "Principles of Large-Scale Computing Machines", reprinted in [vN88]. At that time, "large-scale" meant the ENIAC computer, *i.e.*, 17,468 vacuum tubes, 7,200 crystal diodes, 1,500 relays, 70,000 resistors, 10,000 capacitors, 5 million joint, 30 short tons, 2.4m x 0.9m x 30m, stored in a 167 m^2 room, and 150 kW to operate. Today, thanks to the Moore's law and to the Internet, "large scale" means "planetary scale", *i.e.* the computer hardware is distributed in space and in time and must be negotiated before being used. The authors think that the main inspirations of our Arigatoni overlay network computer are still contained in that historical paper.

As such, we plan to design and implement a *pervasive, programmable, overlay network computer*, *i.e.* a colony of communicating computer individuals that exchange resources and services with various guarantees, execute sequential or parallel algorithms on one or more computer individuals, or perform tasks written in a workflow&dataflow language. An *overlay program* will be a combination of an overlay network connectivity dealing with virtual organizations and a computation of an algorithm resulting of the *summa* of all algorithms running on different computer individuals, and the coordination of all computer individuals using an *ad hoc* language. The metalanguage used to program the overlay network computer is often called (terminology often overlaps), *workflow- dataflow- orchestration- composition- metaprogramming- language*. We could better call such metalanguage a *distributed assembler*, since there is a strong similarity with machine code. As examples, the pseudo machine code instruction *à la* Backus [Bac54] `move R0 R1` can be "refreshed" as

```
move dataR0 from ipR0:portR0 to ipR1:portR1
```

(where of course latency is an non-trivial issue), and the pseudo machine code instruction `op R0 R1 R2` can be recasted as

```
op on ipR0 with
      ipR0:portR0:dataR0 and
      ipR1:portR1:dataR1 and stockin
      ipR2:portR2:dataR2
```

Challenge 2: developing a vehicular infrastructure

We plan to develop algorithmic methods and adapt Arigatoni protocols for building an *ad hoc* vehicular network infrastructure, called Ariwheels [Ari]. That network must enable efficient and transparent access to the resources of on-board and roadside nodes. Commercial services and access to public information will be available to vehicles transiting in specific areas where such information is broadcast by roadside wireless gateways or by other vehicles. Data retrieved can be stored on the on-board vehicle computer; then, they can be used and rebroadcast at a later time without the need of persistent connectivity. We envision that these new features will offer innovative functions and services, such as:

- Distribution, from infrastructure to vehicles (I2V), and among vehicles (V2V), of safety and/or traffic-related information;
- Collection, from vehicles to infrastructures (V2I), of datas useful to perform traffic management operations;
- Information exchange between private vehicles and public transportation systems (buses, vehicles, road side equipments, etc.) to support and, thus, foster inter-modality in urban areas;
- Distribution of real-time information to enable dynamic navigation services.

4.4 Conclusions and future work

The design of our programmable overlay network computer is far to be complete. We are working on a more complete mathematical study of our system, based on more elaborate statistical and stochastic models and realistic assumptions [NCL07], as well as the possibility to include hierarchical DHT in addition to the routing tables. We have already implemented an efficient simulator to validate our design choice [Log]. We are currently working on the implementation of a real client to be deployed on a real size experiments and platforms, like, *e.g.* PLANETLAB, and GRID5000 [Gri]. We hope that Arigatoni could represent a step toward a natural integration of different scenarios under the common paradigm of Overlay and Pervasive Computing (see the *Grand UK Challenges* [Cha], or the new INRIA strategic plan [INR]).

Acknowledgment. The authors would warmly like to thank Didier Benza and Marc Vesin on all issues related to trust, security and social networks, and Philippe Nain for its invaluable comments and interactions on the Arigatoni performance model. This work is supported by Aeolus IST-015964.

References

[Abe01] K. Aberer. P-Grid: A Self-Organizing Access Structure for P2P Information Systems . In *Proc.of CoopIS*, number 2172 in LNCS, pages 179–194. Springer, 2001.

[AD97] S. Alexander and R. Droms. RFC2132, DHCP Options and BOOTP Vendor Extensions. Technical report, IETF, 1997.

[AEO06] AEOLUS. Deliverable D2.1.1: Resource Discovery: State of the Art Survey and Algorithmic Solutions, 2006.

[Ari] Ariwheels. Arigatoni on wheels. `http://www-sop.inria.fr/mascotte/Luigi.Liquori/ARIGATONI/Ariwheels.htm`.

[Bac54] J. W. Backus. The IBM 701 Speedcoding System. *J. ACM*, 1(1):4–6, 1954.

[BCLV06] D. Benza, M. Cosnard, L. Liquori, and M. Vesin. Arigatoni: Overlaying Internet via Low Level Network Protocols. In *JVA, John Vincent Atanasoff International Symposium on Modern Computing*, pages 82–91. IEEE, 2006.

[BCM+99] 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 *Proc. of ICDCS*, 1999.

[Bit] BitTorrent, Inc. `http://www.bittorrent.com/`.

[Car95] L. Cardelli. A language with distributed scope. *Computing Systems*, 8(1):27–59, 1995.

[CCL06] R. Chand, M. Cosnard, and L. Liquori. Resource Discovery in the Arigatoni Overlay Network. In *I2CS, International Workshop on Innovative Internet Community Systems*, LNCS. Springer Verlag, 2006.

[CCL08] R. Chand, M. Cosnard, and L. Liquori. Powerful resource discovery for Arigatoni overlay network. *Future Generation Computer Systems*, 24(1):31–38, 2008.

[CF04] R. Chand and P. Felber. XNet: A Reliable Content-Based Publish/Subscribe System. In *Proc. of SRDS: Symposium on Reliable Distributed Systems*, 2004.

[Cha] Grand UK Challenge. Global Computing and Pervasive Computing. `http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/`.

[CLC07a] R. Chand, L. Liquori, and M. Cosnard. Improving Resource Discovery in the Arigatoni Overlay Network. In *ARCS, International Conference on Architecture of Computing Systems*, LNCS, pages 98–111. Springer Verlag, 2007.

[CLC07b] M. Cosnard, L. Liquori, and R. Chand. Virtual Organizations in Arigatoni. *DCM, International Workshop on Developpment in Computational Models*, 171(3), 2007.

[CRW01] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM TOCS*, 19(3), 2001.

[EFGK03] P. Th. Eugster, P. Felber, R. Guerraoui, and A. M. Kermarrec. The Many Faces of Publish/Subscribe. *Computing Survey*, 35(2):114–131, 2003.

[FM03] M. J. Freedman and D. Mazières. Sloppy Hashing and Self-Organizing Clusters. In *Proc. of IPTPS*, number 2735 in LNCS, pages 45–55. Springer, 2003.

[GKGM04] P. Ganesan, P. Krishna, and H. Garcia-Molina. Canon in G-major: Designing DHTS with Hierarchical Structure. In *Proc. of ICDCS*, pages 263–272. IEEE, 2004.

[Glo] Globus Alliance. `http://www.globus.org/`.

[Gri] Grid 5000 Consortium. http://www.grid5000.org.

[GVE00] A. Gulbrandsen, P. Vixie, and L. Esibov. RFC2782, A DNS RR for specifying the location of services (DNS SRV). Technical report, IETF, 2000.

[Hei01] D. Heimbigner. Adapting publish/subscribe middleware to achieve gnutella-like functionality. In *Proc. of SAC*, pages 176–181, 2001.

[IBM] IBM. Business Process Execution Language. http://www.ibm.com/developerworks/library/specification/ws-bpel/.

[INR] INRIA. Strategic Plan 2008-2012. To appear.

[JV05] H.V. Jagadish and B.C.Ooiand Q.H. Vu. BATON: A Balanced Tree Structure for Peer-to-Peer Networks. In *Proc. of VLDB*, pages 661–672. ACM, 2005.

[JXT] JXTA Community. http://www.jxta.org/.

[LC07] L. Liquori and M. Cosnard. Weaving Arigatoni with a Graph Topology. In *ADVCOMP, International Conference on Advanced Engineering Computing and Applications in Sciences*. IEEE Computer Society Press, 2007.

[Lin00] J. Linn. RFC 2743, Generic Security Service Application Program Interface Version 2, Update 1. Technical report, IETF, 2000.

[Log] LogNet. Arigamulator. http://http://www-sop.inria.fr/mascotte/Luigi.Liquori/ARIGATONI/index.html.

[NCL07] P. Nain, C. Casetti, and L. Liquori. A Stochastic Model of an Arigatoni Overlay Computer . Research report, Politecnico di Torino, 2007.

[Nok] Nokia. N810 Internet Terminal.

[OSG] OSGi Alliance. Open Services Gateway Initiative. http://www.osgi.org/.

[Pau] C. Pautasso. JOpera: Process Support for more than Web Services. http://www.jopera.org/.

[Rap63] A. Rapoport. Mathematical models of social interaction. In *Handbook of Mathematical Psychology*, volume II, pages 493–579. John Wiley and Sons, 1963.

[RWHM03] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. RFC3489, STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). Technical report, IETF, 2003.

[vN88] J. von Neumann. The Principles of Large-Scale Computing Machines. *IEEE Ann. Hist. Comput.*, 10(4):243–256, 1988.

[Whi94] J.E. White. Telescript Technology: the Foundation for the Electronic Marketplace. White Paper. General Magic, Inc., 1994.

[WV03] Y. Wang and J. Vassileva. Trust and Reputation Model in Peer-to-Peer Networks. In *Proc. of Peer-to-Peer Computing*. IEEE Computer Society, 2003.