



# Logical Networks: Self-organizing Overlay Networks and Overlay Computing Systems

Luigi Liquori, Didier Parigot, Bernard P. Serpette

## ► To cite this version:

Luigi Liquori, Didier Parigot, Bernard P. Serpette. Logical Networks: Self-organizing Overlay Networks and Overlay Computing Systems: [EPI Proposal V2.0]. 2010. hal-01147443

**HAL Id: hal-01147443**

**<https://inria.hal.science/hal-01147443>**

Submitted on 30 Apr 2015

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Logical Networks:

## Self-organizing Overlay Networks and Overlay Computing Systems

EPI Proposal

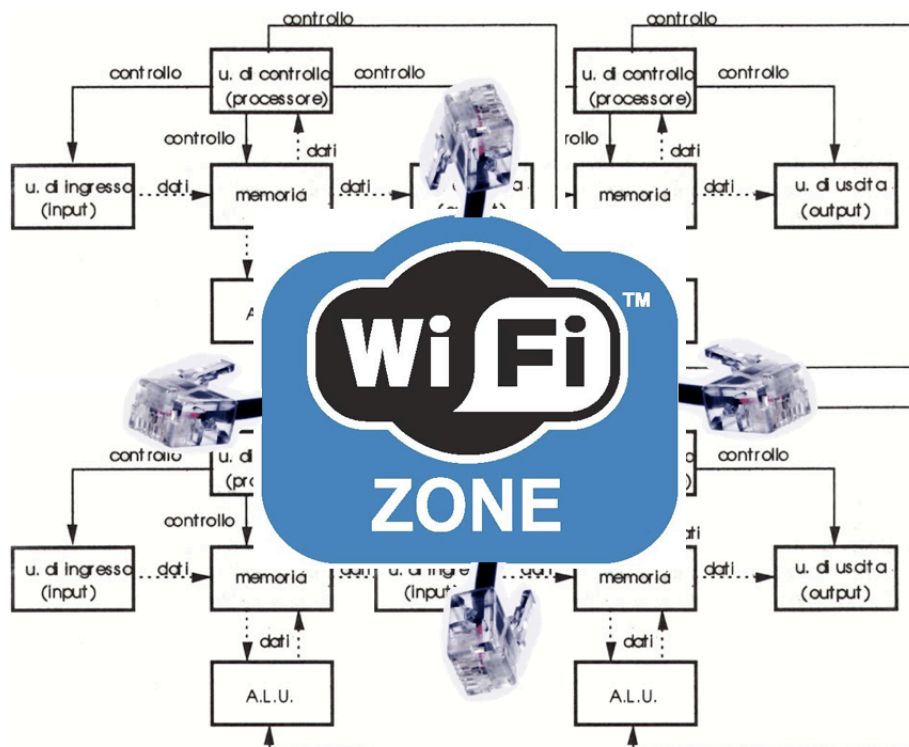
Version 2.0

Theme : Networks and Telecommunications

[LogNet is an Inria team created on January the 1<sup>st</sup>, 2008]

Get the latest version on

<http://www-sop.inria.fr/lognet/v2/>



# Contents

<b>1</b>	<b>Team on March 15, 2010</b>	<b>4</b>
<b>2</b>	<b>Capsule</b>	<b>5</b>
2.1	Slogan and logo . . . . .	5
2.2	One equation fits all and keywords . . . . .	6
2.3	How to read this proposal . . . . .	6
<b>3</b>	<b>Vertical view</b>	<b>6</b>
3.1	Panorama . . . . .	6
3.2	General definitions . . . . .	8
3.3	Virtual organization . . . . .	9
3.4	Execution model . . . . .	9
<b>4</b>	<b>Horizontal view</b>	<b>9</b>
4.1	Panorama . . . . .	9
4.2	Arigatoni overlay network . . . . .	10
4.2.1	Arigatoni units . . . . .	10
4.2.2	Virtual organizations in Arigatoni . . . . .	12
4.2.3	Resource discovery protocol (RDP) . . . . .	12
4.2.4	Virtual Intermittent Protocol (VIP) . . . . .	13
4.2.5	iNeu: libraries for network computing . . . . .	14
4.3	Babelchord, a DHT's tower . . . . .	14
4.4	Synapse, interconnecting heterogeneous overlay networks . . . . .	15
4.5	Cross-layer overlay design for geo-sensible applications . . . . .	17
<b>5</b>	<b>Diagonal view</b>	<b>17</b>
5.1	Panorama . . . . .	17
5.2	Trees <i>versus</i> graphs: a conflict without a cause . . . . .	17
5.3	Fault tolerance . . . . .	18
5.4	Parametricity and universality . . . . .	18
5.5	Social networking . . . . .	19
5.6	Choice of development platform . . . . .	19
5.7	Quality metrics for an overlay computer . . . . .	19
5.8	Trust and security . . . . .	20
5.9	New models of computations . . . . .	21
<b>6</b>	<b>Topics and time line</b>	<b>22</b>
6.1	Panorama . . . . .	22
6.2	Topic view . . . . .	22
6.2.1	Vertical issues . . . . .	22
6.2.2	Horizontal issues . . . . .	22
6.2.3	Diagonal issues . . . . .	23
6.3	Time view . . . . .	23
6.3.1	Short-term . . . . .	23
6.3.2	Medium-term . . . . .	24
6.3.3	Long-term . . . . .	24

<b>7</b>	<b>Potential application domains</b>	<b>24</b>
7.1	Panorama . . . . .	24
7.2	P2P social networks . . . . .	25
7.3	Overlay computer for mobile <i>ad hoc</i> networks . . . . .	25
7.4	OverStic: the mesh overlay network in Sophia Antipolis . . . . .	27
7.5	Reducing the <i>Digital Divide</i> . . . . .	28
7.6	GRID applications : scenario for seismic monitoring . . . . .	29
7.7	Interconnection of heterogeneous overlay networks . . . . .	30
7.8	Toward an overlay network of things (RFID) . . . . .	31
<b>8</b>	<b>Software</b>	<b>32</b>
8.1	Panorama . . . . .	32
8.2	Prototype software . . . . .	32
8.2.1	Arigatoni simulator . . . . .	32
8.2.2	Ariwheels . . . . .	32
8.2.3	BabelChord . . . . .	36
8.2.4	Synapse . . . . .	37
8.2.5	Open-Synapse Client . . . . .	38
8.2.6	myTransport Gui . . . . .	39
8.2.7	CarPal: a P2P car pooling service . . . . .	39
8.2.8	Husky interpreter . . . . .	40
8.3	Potential software . . . . .	41
8.3.1	myMed (in french), see <a href="http://www-sop.inria.fr/mymed">http://www-sop.inria.fr/mymed</a> . . . . .	41
<b>9</b>	<b>Contracts</b>	<b>43</b>
9.1	INTERREG Alcotra: myMed, 2010-2013 . . . . .	43
9.2	COLOR: JMED, 2010 . . . . .	43
9.3	FP6 FET Global Computing: IST AEOLUS, 2006-2010 . . . . .	43
9.4	JET TEMPUS DEUKS, 2007-2009 . . . . .	44
<b>10</b>	<b>Collaborations</b>	<b>44</b>
<b>11</b>	<b>Self assessment</b>	<b>44</b>
11.1	Trivia . . . . .	45
11.2	Conclusions . . . . .	45

# 1 Team on March 15, 2010

## Research Scientist

Luigi Liquori [Team Leader, Research associate, CR INRIA, HdR]

## External Collaborators

Claudio Casetti [ Assistant professor, Politecnico di Torino, Italy ]

Carla-Fabiana Chiasserini [ Associate professor, Politecnico di Torino, Italy ]

Michel Cosnard [ CEO INRIA, Full professor UNSA]

## Technical Staff

Laurent Vanni [ Software Engineer myMed, from January 1st 2010 to January 31th 2013 ]

X.Y [ Coordination Engineer myMed, (under recruiting) till January 31th 2013 ]

**Forecast 100%**

X.Y [ Software Engineer myMed, from September 1st 2010 to January 31th 2013 ]

## PhD Students

Vincenzo Ciancaglini [ MENRT grant, since October 1st 2009, defense planned in 2012 ]

Petar Maksimovic [ TEMPUS-BASILEUS grant, since December 1st 2008, defense planned in 2011 ]

**Forecast 100%**

X.Y [myMed grant, Ph.D. from September 1st 2010 to August 31th 2013 ]

X.Y [myMed grant, Ph.D. from September 1st 2010 to August 31th 2013 ]

## Others

Fofack Nicaise [ Master Ubinet, from March 1st 2010 to August 31th 2010 ]

Ali Makke [ Master Ubinet, from March 1st 2010 to August 31th 2010 ]

Thao Nguyen [ Master Ubinet & MC3 I3S, from March 1st 2010 to August 31th 2010 ]

Nicolas Goles [Trainee, Universidad Tecnica Federico Santa Maria, Chile, from January 1st 2010 to March 31th 2010 ]

Rossella Fortuna [ Ph.D. Politecnico di Bari, Italy from March 15th to September 15st 2010 ]

Marco Servetto [ Ph.D. Università di Genova, Italy, from Juin 1st 2010 to December 31th 2010 ]

X.Y [Trainee, IUT Valrose, from April 5th 2010 to Juin 11th 2010 ]

**Forecast 100%**

Nicolas Goles [Master, Universidad Tecnica Federico Santa Maria, Chile, from January 1st 2011 to Juin 31th 2011 ]

## Administrative Assistant

Nathalie Bellesso [ INRIA ]

## Potential forecast, with other ressources

## Research Scientist

Vincenzo Mancuso [CR2 INRIA candidate, Competitive selection researchers 2010, from September 1st 2010 ]

## Technical Staff

X.Y [ Senior Software Engineer ADT myMed@INRIA, from September 1st 2010 to August 31th 2013 ]

X.Y [ Junior Software Engineer ADT myMed@INRIA, from September 1st 2010 to August 31th 2013 ]

## 2 Capsule

**Acronym** : LogNet

**Domain** : Networks, Systems and Services, Distributed Computing

**Theme** : Networks and Telecommunications

**Center** : INRIA Sophia Antipolis - Méditerranée

**Abstract** We propose foundations for *generic overlay networks* and *overlay computing systems*. Such overlays are built over a large number of distributed *computational agents*, virtually organized in *colonies* or *virtual organizations*, and ruled by a leader (*broker*) who is elected democratically (*vox populi, vox dei*) or imposed by system administrators (*primus inter pares*). Every agent asks the broker to log in the colony by declaring the resources that can be offered (with variable guarantees). Once logged in, an agent can ask the broker for other resources. Colonies can recursively be considered as *evolved agents* who can log in an outermost colony governed by another super-leader. 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 forwarding the request first inside its colony, and second outside, via the proper super-leader (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 (atomic services). When the client agent receives notification of all (or part of) the requested resources, then the real resource exchange is performed directly by the server(s) agents, 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 agents have quit or are temporarily unavailable, or they were logged out *manu militari* by the broker due to their poor performance or greediness. We aim to design, validate through simulation, and implement these foundations in an overlay network computer system. (From [LC07b])

### 2.1 Slogan and logo

Internet Computer or Computer Networks? This seems the current dilemma (see FET's calls in FP6 and FP7 programs). As usual the true is probably in the middle. What is surely true is that we are constantly moving toward a new type of computational models and devices where the network play an important role as the "bus" play a crucial rule in von Neumann architecture: in other word we are moving from physical, proprietary, personal, more or less expensive, programmable, Turing Complete computers versus to ethereal, virtual, public, impersonal, ubiquitous, cloud, inexpensive, programmable, Turing Complete *overlay computers*. The above statement is well resumed by the slogan below.

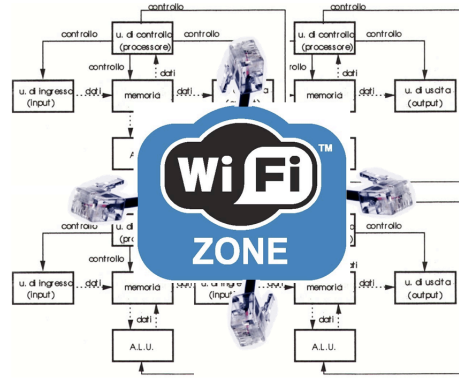
*"Computer is moving on the edge of the Network..."*

[Jan Bosch, Nokia Labs, Keynote ARCS, LNCS 4415, 2007]

The official team logo we have invented mix a wifi area's logo with four Internet cables (RJ42) and four copies of a von Neumann model from the original textbook (in italian) of the team leader. This logo pictorially suggests that probably current computational models (based on the Von Neumann model) are reaching their limits, and other models are actually combining computing with communicating, so leading to an interesting marriage of two, usually unrelated, disciplines. In the functional jargon we could say

$$\text{COMM } B =_{\beta} \text{SYM } A$$

where, for the non acolytes of lambda-calculus,  $=_{\beta}$  means convertible or equivalent.



## 2.2 One equation fits all and keywords

Our golden equation is

$$\begin{array}{rcl}
 \text{Register into the Overlay Network} & + & \\
 \text{Discover in the Overlay Network} & + & \\
 \text{Execute through the Overlay Network} & = & \\
 \hline
 \text{Overlay Computer} & & 
 \end{array}$$

and the most important keywords are

Internet of the Future and of Things, Overlay Computer, {Logical, Overlay, Social, Peer-to-peer, Self\*}-Networks, Cloud, Service Oriented, Grid, Network, Brain-inspired-models of computation, Publish-Subscribe message paradigm. Formal Methods, Logic and Type theory, {Object-Oriented, Functional}-programming.

## 2.3 How to read this proposal

Inspired by the seminal paper of Abiteboul “On views and XML” (Proceedings of Symposium on Principles of Database Systems, 1999) the content of our proposal can be viewed by different “angles” (called views by Abiteboul), or “filters” in term-rewriting jargon.

**(Vertical view)** is about the main research axes investigated by the team;

**(Horizontal view)** is about the particular topic each member is more involved;

**(Diagonal view)** is about some topics that we will encounter and study during our scientific pilgrimage with less priority;

**(Topic view)** is about a “dry” list of research item;

**(Time view)** is the same dry list organized by priority;

**(Positioning view)** is about some related work and teams involved in part of our research topics.

Management and Referees, can choose and browse the views that they feel more comfortable with. “Criss-crossing” views could be also useful.

## 3 Vertical view

### 3.1 Panorama

**Keywords.** Overlay networks, virtual organizations, social networks, resource discovery, query routing

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 our statement with the standard definition of *Computer System* and then scale up to the our definition of *Overlay Computer* (we emphasize some text using underline). A plætora of overlay computer's definitions can be found in FP6 project and presentations (see *e.g.* Aeolus, Mobius, Sensoria).

### 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 runs 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 the LogNet's 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 agents 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 middleware playing as a distributed operating system (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 classes of applications deal with computational power (Grid), file and storage retrieval (P2P), web services (Web2.0), band-services (VoIP), computation migrations...*



Therefore, LogNet’s general objectives can be resumed as follows:

- to provide adequate notions and definitions of an *generic overlay computer*: logic, communications, implementations, applications, hardware;
- on the basis of the above definitions, to propose a precise architecture of an overlay computer with related execution model and implement it;
- on the basis of the above definitions, to implement useful applications suitable to help the logical and software assembling of an overlay computer and experiment it at large scale;
- putting our *savoir faire* in logics, type theory, formal systems, object-oriented, functional programming to the service of telecommunications and the, so called *Internet of the future and of things*.

### 3.2 General definitions

An overlay network is a logical network which is built on top of another physical network. Overlay networks can be constructed in order to permit routing messages to destinations not specified by an IP address. In what follows, we briefly describe the main entities underneath our vision of an overlay network and an overlay computer.

**Agents.** An agent in the overlay 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). Agents in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, through many physical links, in the underlying network. For example, many peer-to-peer networks are overlay networks because they run on top of the Internet.

**Colonies and colony’s leaders.** Agents in the overlay are regrouped in *Colonies*. A colony is a simple virtual organization composed by exactly one *leader*, offering some broker-like services, and some set of *agents*. The leader, being also an agent, can be an agent of a colony different of the one it manages. Thus, agents are simple computers (think it as an *amoeba*), or sub-colonies (think it as a *protozoa*). Every colony has *exactly* one leader and at least one agent (the leader itself). Logically an agent can be seen as a *collapsed colony*, or a *leader managing itself*. The leader is the only one who knows all agents of its colony. One of the tasks of the leader is to manage (un)subscriptions to its colony.

**Resource discovery.** By adhering a colony, an agent can expose resources it has and/or ask for resources it needs. Another task of a leader is to manage the resources available in its colony. Thus, when an agent of the overlay needs a specific resource, it makes a request to its leader. A leader is devoted to contacting and negotiating with potential servers, to authenticating clients and servers, and to route requests. 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-exogenous-last* strategy).

**Orchestration.** When an agent receives an acknowledgment of a service request from the direct leader, then the agent is served directly by the server(s) agents, *i.e.* without a further mediation of the leader, in a pure P2P fashion. Thus, the “main” program will be run on the agent computer machine that launched the service request and received the resources availability: it will orchestrate and coordinate data and program resources executed on others agent computers.

**Underlay-aware operation.** Overlay protocols have to run on top of different physical layers, *i.e.*, different underlay technologies, and traverse hardware and software barriers like gateways and firewalls, respectively. Overlay routing can smartly take into account the heterogeneity of the physical layer and the presence of barriers in two ways: *(i)* performing underlay-aware routing, *i.e.*, taking explicitly into consideration the network connectivity of the agents, and *(ii)* delivering to the agent more information than a mere routing answer, *e.g.*, by providing the agents with multiple choices, each coupled to a descriptor of the connectivity/quality of the connectivity.

### 3.3 Virtual organization

A virtual organization is a flat or hierarchical collection of *agents*. An agent in the overlay 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). Agents in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, through many physical links, in the underlying network. For example, many peer-to-peer networks are overlay networks because they run on top of the Internet. Agents may collaborate through flat connections between colonies or hierarchical connection in tree-based colonies. In this latter case a colony is a simple virtual organization composed by exactly one *leader*, offering some broker-like services, and some set of *agents*. The leader, being also an agent, can be an agent of a colony different of the one it manages. Thus, agents are simple computers (think it as an *amoeba*), or sub-colonies (think it as a *protozoa*). Every colony has *exactly* one leader and at least one agent (the leader itself). Logically an agent can be seen as a *collapsed colony*, or a *leader managing itself*. The leader is the only one who knows all agents of its colony. One of the tasks of the leader is to manage (un)subscriptions to its colony.

### 3.4 Execution model

We plan to investigate the theoretical study and the actual implementation of new primitives to better interface programming languages with overlay networks. Once resources (hardware, software...) have been discovered, the agent computer that made the request may wish to use and manipulate it; to do this, the agent computer has written a (distributed) program using a library called *Ivonne*, in honor to the great scientist John von Neumann. Those languages are often called (terminology often overlaps), *coordination- workflow- dataflow- orchestration- composition- metaprogramming- languages*. *Ivonne* will have *ad hoc* primitives to express sequences, iterators, cycles, parallel split, joins, synchronization, exclusive/multi/deferred choice, simple/multi/synchronizing merge, discriminators, pipelining, cancellation, implicit termination, exception handling... [vdAtH05].

The “main” of an *Ivonne* program will be runned on the agent computer machine that launched the service request and received the resources availability: it will orchestrates and coordinates data and program resources executed on others agent computers.

In case of *failure* of a remote service – due to a network problem or simply because of the unreliability or untrustability of the agent that promised the resource – an exception handling mechanism will send a resource discovery query *on the fly* to recover a faulty peer and the actual state of the run represented, in semantic jargon, by the *current continuation*. Resuming, an *overlay program* will be a smooth combination of an overlay network connectivity dealing with virtual organizations and discovery protocols, a computation of an algorithm resulting of the *summa* of all algorithms running on different computer agents, and the coordination of all computer agents, made by an *Ivonne* main program.

## 4 Horizontal view

### 4.1 Panorama

As suggested by our previous definitions, we are mainly concerned by three topics: network organization, resource discovery and orchestration. These topics are studied in an overlay called *Arigatoni* (work started by Luigi Liquori and Michel Cosnard). Indeed *Arigatoni* is build around a notion of virtual organization (colonies of agents). Following our main three topics, network organization, resource discovery and orchestration, *Arigatoni* goes deeper in the network organization and the resource discovery (VIP and RDP). We will make an implementation of the VIP and RDP protocols.

## 4.2 Arigatoni overlay network

The Arigatoni overlay network computer [BCLV06a, CCL06, LC07b, LC07a, CLC07a, CLC07b, NCL07], and [CCL08] developed since 2006 in the Mascotte Project Team by Luigi Liquori and Michel Cosnard, and then in the LogNet team, is a structured multi-layer overlay network which provides resource discovery with variable guarantees in a virtual organization where peers can appear, disappear, and self-organize themselves dynamically. Arigatoni is *universal* in the sense of Turing machines, or *generic* as the von Neumann computer architecture is.

Every agent asks the broker to log in the colony by declaring the resources that it provides (with variable guarantees). Once logged in, an agent can ask the broker for other resources. Colonies can recursively be considered as *evolved agents* who can log in an outermost colony governed by another super-leader. Communications and routing intra-colonies go through a broker-2-broker PKI-based negotiation. Every broker routes intra- and inter- *service requests* by filtering its *resource routing table*, and then forwarding the request first inside its colony, and second outside, via the proper super-leader (thus applying an *endogenous-first-estrogen-last* strategy).

Theoretically, queries are formulæ in first-order logic. When the client agent receives notification of all (or part of) the requested resources, then the real resource exchange is performed directly by the server(s) agents, without any further mediation of the broker, in a pure peer-to-peer fashion. The proposed overlay promotes an *intermittent* participation in the colony. Therefore, the routing process may lead to *failures*, because some agents have quit, or are temporarily unavailable, or they were logged out by the broker due to their poor performance or greediness.

Arigatoni features essentially two protocols: the *resource discovery protocol* dealing with the process of an agent broker to find and negotiate resources to serve an agent request in its own colony, and the *virtual intermittent protocol* dealing with (un)registrations of agents to colonies.

Dealing essentially with resource discovery and peers' intermittence has one important advantage: the complete generality and independence of any offered and requested resource. Arigatoni can fit with various scenarios in the global computing arena, from classical P2P applications (file- or bandwidth-sharing), to new Web2.0 applications, to new V2V and V2I over MANET applications, to more sophisticated Grid and Cloud applications, until possible, futuristic *migration computations*, i.e. transfer of a non-completed local run to another agent, the latter being useful in case of catastrophic scenarios, like fire, terrorist attack, earthquake, etc.

### 4.2.1 Arigatoni units

In what follows, we briefly introduce the logic peers underneath a generic overlay network. 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 agent computers 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 agents 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.

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.

**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  $\leq 40$  keystrokes keyboard (or touchscreen), a tiny screen ( $\leq 4$  inch), a network connection such as the ones using IP through wireless infrastructures (e.g. WIFI, WIMAX, HSP, LTE, ...) or ad hoc connectivity (e.g. BLUETOOTH, Zigbee, RFID, ...).

a USB port, and very few programs installed inside, e.g. one simple editor, one or two compilers, a mail client, a mini browser... Our favorite device actually is the Internet tablet Nokia N810. 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 agent brokers (ABs), playing as colony leaders, upon its arrival of itself in a “connected area”; this can be done using the underlay network and related technologies;
- 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 other ACs 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.

**Agent Broker (AB).** This unit can be, *e.g.*, a computer device made by a high speed CPU, a network connection such as the ones using IP through wireless infrastructures (*e.g.* WIFI, WIMAX, HSP, LTE, ...) or ad hoc connectivity (*e.g.* BLUETOOTH, Zigbee, RFID, ...), a 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:

- 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 AC 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 base (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; it must register itself (and by product 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 ACs 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).

**Agent Router (AR).** This unit implements all the low-level overlay network routines, those which really have access to the IP or to the *ad-hoc* connections. 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.

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.

#### 4.2.2 Virtual organizations in Arigatoni

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 agents. Agent computers 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 [IBM].

Symmetrically, the leader of a colony can arbitrarily unregister an agent 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 service, 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 agent (computer or broker, representing a 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 agents logged out or because they are temporarily unavailable (recall that agents are not slaves). This may also lead to temporary denials of service or, more drastically, to the complete logout of an agent from a given colony in the case where the former does not provide enough services to the latter.

#### 4.2.3 Resource discovery protocol (RDP)

**Kind of discovery.** There are mostly two mechanisms of resource discovery, 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 Arigatoni's resource discovery protocol, while the second is processed out of the Arigatoni overlay, using well-known network protocols, like DHCP, DNS, the service discovery protocol SLP of BLUETOOTH, or Active/Passive Scanning in WIFI.

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 will 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  $S \triangleq [\text{CPU}=\text{Intel}, \text{Time} \leq 10\text{sec}]$  filters against  $S' \triangleq [\text{CPU}=\text{Intel}, \text{Time} \geq 5\text{sec}]$ , with attribute values Intel and Time between 5 and 10 seconds.

**Routing tables in RDP.** In Arigatoni, each AB maintains a *routing table*  $T$  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 lookup overhead reduces when a query is satisfied in the current colony. Most structured overlays guarantee lookup 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.

#### 4.2.4 Virtual Intermittent Protocol (VIP)

There are essentially two ways AC can register to an AB (sensible to its physical position in the network topology), the latter being not enforced by the Arigatoni model (see [CLC07b]):

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

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

- Unregistration of an AC is allowed only when there are no pending services demanded or requested to the leader AB of the colony: agent computers 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 belonging to any administrative domain, it can physically migrate in another colony belonging to any other administrative domain;
- Selfish agents 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 the agent’s ratio of fairness is too small ( $\leq \epsilon$  for a given  $\epsilon$ ), it can arbitrarily decide to fire that agent without notice. Here, the VIP protocol also checks that the agent 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).

**Registration policies in VIP.** VIP registration policies are usually not specified in the protocol itself; thus every agent broker is free to choose its acceptance policy. This induces different self-organization policies and allow to reason on colony’s load-balancing and kind of colonies. Possible politics and are:

- **(mono-thematic)** An agent broker accept an agent in its colony if the latter offer resources  $S$  that the colony already have in quantity  $\geq \epsilon$ , for a given  $\epsilon$ ;
- **(multi-thematic)** An agent broker accept an agent if the latter offer resources that the colony have in quantity  $\leq \epsilon$ , for a given  $\epsilon$ ;

- **(unbalanced)** An agent broker accept an agent always;
- **(pay-per-service)** An agent broker accept only agents that accept to pay some services;
- **(metropolis/village)** An agent broker accept an agent in its colony only if the number of citizens is greater/lesser than  $N$ ;
- **(village)** An agent broker accept an agent in its colony only if the number of citizens is lesser than  $N$ ;
- **(custom)** An agent broker accept an agent following a flexible meta-politics that can mix of the above politics.

#### 4.2.5 iNeu: libraries for network computing

Once resources (hardware, software...) have been discovered, the agent computer that made the request may wish to use and manipulate them; to do this, the agent computer has written a (distributed) program using suitable library/dialect (the final choice still left open by the team) let's call it **lvonne**, in honor to the great scientist John von Neumann. Those languages are often called (terminology often overlaps), *coordination-workflow- dataflow- orchestration- composition- metaprogramming- languages*. **lvonne** will have *ad hoc* primitives to express sequences, iterators, cycles, parallel split, joins, synchronization, exclusive/multi/deferred choice, simple/multi/synchronizing merge, discriminators, pipelining, cancellation, implicit termination, exception handling... [vdAtH05].

The “main” of a Java or C or C++ program using the **lvonne** library will be runned on the agent computer machine that launched the service request and received the resources availability: it will orchestrate and coordinate data and program resources executed on others agent computers.

In case of *failure* of a remote service – due to a network problem or simply because of the unreliability or untrustability of the agent that promised the resource – an exception handling mechanism will send a resource discovery query *on the fly* to recover a faulty peer and the actual state of the run represented, in semantic jargon, by the *current continuation*.

### 4.3 Babelchord, a DHT's tower

A significant part of today's Internet traffic is generated by peer-to-peer (P2P) applications, used originally for file sharing, and more recently for real-time multimedia communications and live media streaming.

Distributed Hash Tables (DHTs) or “structured overlay networks” have gained momentum in the last few years as the breaking technology to implement scalable, robust and efficient Internet applications. DHTs provide a lookup service similar to a hash table: (key, value) pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from names to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows DHTs to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

Chord [SMK<sup>+</sup>01] is one of the simplest protocols addressing key lookup in a distributed hash table: the only operation that Chord supports is that given a key, it routes onto a node which is supposed to host the entry (key,value). Chord adapts efficiently as nodes join and leave the system. Theoretical analysis and simulations showed that the Chord protocol scales up logarithmically with the number of nodes. In Chord, every node can join and leave the system without any peer negotiation, even though this feature can be implemented at the application layer. Chord uses consistent hashing in order to map keys and nodes' addresses, hosting the distributed table, to the same logical address space. All the peers share a common hash function, representing the only way to map physical addresses and keys to a single logical address space. Peers can join the Chord just by sending a message to any node belonging to the Chord overlay. No reputation mechanism is required to accept, reject, or reward peers that are more reliable or more virtuous than others. Merging two Chord rings together is a costly operation because of the induced message complexity and the substantial time the finger tables needs to stabilize. Separated rings having

different hash function for security reasons but wishing to access resources (read only access) of other rings without merging could take advantage of BabelChord. We propose to connect smaller Chord networks in an unstructured way via special nodes playing the role of *neural synapses*.

Schematically, the main BabelChord's features are: **Routing over SW/HW-Barriers**. Namely, the ability to route queries through different, unrelated, DHTs (possibly separated by firewalls) by "crossing floors". A peer "on the border" of a firewall can bridge two overlays (having two different hash functions) that were not meant to communicate with each other unless one wants to merge one floor into the other (operation with a complexity linear in the number of nodes). The possibility to implement strong or weak security requirements makes BabelChord suitable to be employed in Internet applications where software or social barriers are an important issue to deal with.

**Social-based**. Every peer has data structures recording peers and floors which are more "attractive" than others. A "hot" node is a node which is stable (alive) and which is responsible for managing a large number of (keys-values) in all hosted DHTs. A "hot" floor is a floor responsible of a high number of successful lookups. Following a personal "good deal" strategy, a peer can decide to invite an hot node on a given floor it belongs to, or to join a hot floor, or even create from scratch a new floor (and then invite some hot nodes), or accept/decline an invitation to join a hot floor. This social-behavior makes the BabelChord network topology to change dynamically. As observed in other P2P protocols, like Bittorrent, peers with similar characteristics are more willing to group together on a private floor and thus will eventually improve their overall communications quality. Finally, the "good deal" strategy is geared up to be further enhanced with a reputation-system for nodes and floors.

**Neural-inspired**. Since every floor has a proper hash function, a BabelChord network can be thought as a sort of *meta overlay network* or *meta-DHT*, where inter floors connections take place via crossroad nodes, a sort of neural synapses, without sharing a global knowledge of the hash functions and without a time consuming floor merging. The more synapses you have the higher the possibility of having successful routing is.

## 4.4 Synapse, interconnecting heterogeneous overlay networks

We are currently investigating Synapse [LTV<sup>+</sup>10], a scalable protocol for information retrieval over the inter-connection of heterogeneous overlay networks. Applications of top of Synapse see those intra-overlay networks as a unique inter-overlay network.

See <http://www-sop.inria.fr/lognet/synapse>.

Scalability in Synapse is achieved via co-located nodes, *i.e.* nodes that are part of multiple overlay networks at the same time. Co-located nodes, playing the role of *neural synapses* and connected to several overlay networks, give a larger search area and provide alternative routing.

Synapse can either work with "open" overlays adapting their protocol to synapse interconnection requirements, or with "closed" overlays that will not accept any change to their protocol. Built-in primitives to deal with social networking give an incentive for nodes cooperation. Results from simulation and experiments show that Synapse is scalable, with a communication and state overhead scaling similarly as the networks inter-connected. thanks to alternate routing paths, Synapse also gives a practical solution to network partitions. We precisely capture the behavior of traditional metrics of overlay networks within Synapse and present results from simulations as well as some actual experiments of a client prototype on the Grid'5000 platform. The prototype developed implements the Synapse protocol in the particular case of the inter-connection of many Chord overlay networks.

The inter-connection of overlay networks has been recently identified as a promising model to cope with today's Internet issues such as scalability, resource discovery, failure recovery or routing efficiency, in particular in the context of information retrieval. Some recent researches have focused on the design of mechanisms for building bridges between heterogeneous overlay networks for the purpose of improving cooperation between networks that have different routing mechanisms, logical topologies and maintenance policies. However, more comprehensive approaches of such inter-connections for information retrieval and both quantitative and experimental studies of its key metrics, such as satisfaction rate or routing length, are still missing.



Many disparate overlay networks may not only simultaneously co-exist in the Internet but also compete for the same resources on shared nodes and underlying network links. One of the problems of the overlay networking area is how heterogeneous overlay networks may *interact* and *cooperate* with each other. Overlay networks are heterogeneous and basically unable to cooperate each other in an effortless way, without merging, an operation which is very costly since it is not scalable and not suitable in many cases for security reasons. However, in many situations, distinct overlay networks could take advantage of cooperating for many purposes: collective performance enhancement, larger shared information, better resistance to loss of connectivity (network partitions), improved routing performance in terms of delay, throughput and packets loss, by, for instance, cooperative forwarding of flows.

As a basic example, let us consider two distant databases. One node of the first database stores one (*key, value*) pair which is searched by a node of the second one. Without network cooperation those two nodes will never communicate together. As another example, we have an overlay network where a number of nodes got isolated by an overlay network failure, leading to a partition: if some or all of those nodes can be reached via an alternative overlay network, then the partition “could” be recovered via an alternative routing.

In the context of large scale information retrieval, several overlays may want to offer an aggregation of their information/data to their potential common users without losing control of it. Imagine two companies wishing to share or aggregate information contained in their distributed databases, obviously while keeping their proprietary routing and their exclusive right to update it. Finally, in terms of fault-tolerance, cooperation can increase the availability of the system, if one overlay becomes unavailable the global network will only undergo partial failure as other distinct resources will be usable.

We consider the tradeoff of having one *vs.* many overlays as a conflict without a cause: having a single global overlay has many obvious advantages and is the *de facto* most natural solution, but it appears unrealistic in the actual setting. In some optimistic case, different overlays are suitable for collaboration by opening their proprietary protocols in order to build an open standard; in many other pessimistic cases, this opening is simply unrealistic for many different reasons (backward compatibility, security, commercial, practical, etc.). As such, studying protocols to interconnect collaborative (or competitive) overlay networks is an interesting research vein.

The main contribution of this research vein is to introduce, simulate and experiment with *Synapse*, a scalable protocol for information retrieval over the inter-connection of heterogeneous overlay networks. The protocol is based on co-located nodes, also called *synapses*, serving as low-cost natural candidates for inter-overlay bridges. In the simplest case (where overlays to be interconnected are ready to adapt their protocols to the requirements of interconnection), every message received by a co-located node can be forwarded to other overlays the node belongs to. In other words, upon receipt of a search query, in addition to its forwarding to the next hop in the current overlay (according to their routing policy), the node can possibly start a new search, according to some given strategy, in some or all other overlay networks it belongs to. This obviously implies to providing a Time-To-Live value and detection of already processed queries, to avoid infinite loop in the network, as in unstructured peer-to-peer systems.

We also study interconnection policies as the explicit possibility to rely on *social* based strategies to build these bridges between distinct overlays; nodes can invite or can be invited.

In case of concurrent overlay networks, inter-overlay routing becomes harder, as intra-overlays are provided as some black boxes: a *control* overlay-network made of co-located nodes maps one hashed key from one overlay into the original key that, in turn, will be hashed and routed in other overlays in which the co-located node belongs to. This extra structure is unavoidable to route queries along closed overlays and to prevent routing loops.

Our experiments and simulations show that a small number of well-connected synapses is sufficient in order to achieve almost exhaustive searches in a “synapsed” network of structured overlay networks. We believe that Synapse can give an answer to circumventing network partitions; the key points being that:

- several logical links for one node leads to as many alternative physical routes through these overlay, and

- a synapse can retrieve keys from overlays that it doesn't even know simply by forwarding their query to another synapse that, in turn, is better connected.

Those features are achieved in Synapse at the cost of some additional data structures and in an orthogonal way to ordinary techniques of caching and replication. Moreover, being a synapse can allow for the retrieval of extra information from many other overlays even if we are not connected with.

## 4.5 Cross-layer overlay design for geo-sensible applications

Overlay and underlay exist separately, but they are not forced to be uncorrelated. Instead, the overlay operation could exploit the characteristics of the physical underlay, i.e., the underlay technologies and its performance in terms of connectivity, throughput, delay, congestion status at a given point in time, and also geographical localization.

These pieces of information could be smartly leveraged by the overlay agents, especially in the field of mobile overlays, such as the mobile social network case. As an example, the knowledge of network characteristics could drive mobile agents in a social networks selecting the contact agent (e.g., a touristic center) to which the underlay network path has the best performance.

It is also possible to make the underlay reactive to the overlay requirements. For instance, the availability of multiple wireless network interfaces (e.g., 802.11, 802.16, 802.20, GPRS, UMTS, HSPA, LTE) at the mobile agent allows the underlay network to select the communication channel according to the service requested by the agent. Therefore, the overlay might be aware of underlay operations/performance and vice versa. If not, the exchange of data between two agents could be severely affected by the technologies selected on the path in between the agent pair [CCFM08]. Likewise, optimization in the usage of dissemination points in the overlay is only possible if the underlay infrastructure is known [TCFC<sup>+</sup>10].

A study case in which the cross-layer approach helps improving performance is given by social networks with agents that use a vehicular-to-infrastructure architecture. In this case, the social network could coordinate the information about agents' mobility and service coverage. Thereby, the overlay could support highly geo-sensible applications, made available based on the position of agents and the subscribed services.

In order to sustain geo-based services, it is interesting to investigate the potentiality of using, in the underlay, some lightweight base stations (e.g., femtocells [CAG08]) that have the capability to follow the mobility of the agent, efficiently trigger fast handover operations, guarantee high bandwidth, enable new power saving operations. For instance, base stations might be operated as on-demand agents with a wired and a wireless interface, the latter being active only in presence of traffic demand detected in the overlay.

## 5 Diagonal view

### 5.1 Panorama

The diagonal view presents some *transverse* research problems that we could eventually encounter and study during our research pilgrimage.

### 5.2 Trees *versus* graphs: a conflict without a cause

In the first versions of Arigatoni, the network topology was tree- or forest-based. But since agent computers are not slaves, multiple registrations are in principle possible and unavoidable. This weaves the network topology to a *dynamic graph* where nodes do not have a complete knowledge of the topology itself. As an immediate consequence, Arigatoni's protocols deal with multiple registrations of the same agent in different colonies, with the natural consequence of resource overbooking, routing table update loops (when a service update request comes back to the broker that generates the request itself), and resource discovery loops (when a resource service request comes back to the agent that generates the request itself), see [MC82], [LC07b].

As an example of resource overbooking, suppose an agent computer 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 (airlines) and hotel reservations.

The Synapse evolution .....

### 5.3 Fault tolerance

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. Peers’ intermittence lead also to design new routing algorithms and protocols stable to agent churn; this scenario can be modeled using dynamic/random graph theory.

The overlay network model offers some mechanisms to dynamically adapt to *dynamic topology changes* of the overlay network, by allowing an agent (computer or broker, representing a sub-colony) to login/logout in/from a colony. This essentially means that the process of routing requests and responses may lead to failure, because some agents logged out or because they are temporarily unavailable (recall that agents are not slaves). This may also lead to temporary denials of service or, more drastically, to the complete “delogging” of an agent from a given colony in the case where the former does not provide enough services to the latter.

### 5.4 Parametricity and 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 or Telescript. We could envisage at least the following scenarios to be a tight fit for our model:

- 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 (*i.e.* a kind of distributed von Neumann machine);
- Ask for a computation migration (*i.e.* transfer one partial run in another agent computer, saving the partial results, as in a truly mobile ubiquitous computation);
- Ask for a *human computer interaction* (the human playing the role of an agent)...

## 5.5 Social networking

The Arigatoni and the BabelChord overlay networks define 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, recent studies showed that the Arigatoni overlay 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* [BCLV06b].

This means that Arigatoni and BabelChord fits with motivations and cooperation behavior of different communities. It tries to be *policy neutral*, leaving policy choices for each agent at the implementation or configuration level, or at the community or organization level. Policy domains can overlap (one agent can define itself as belonging “much” to colony (or floor) *foo* and “a little bit” to colony (or floor) *bar*). This denotes a decentralized non-exclusive policy model. As such, one question can arise: who is Arigatoni and BabelChord are 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 agents;
- More regulated or hierarchical communities (maybe a better picture of a corporate network);
- Cooperative or competitive resource providers and resource brokers.

## 5.6 Choice of development platform

We have chosen iPhone OS to be one of the main deployment targets for our research in the team for several reasons. First of all, the iPhone OS is a “closed” development platform, in the sense that we can achieve better control over the quality of the applications we build, and actually be certain that they will run in all versions of iPhone/iPod touch, without making any additional cross-platform efforts. We view this as a substantial advantage over similar developments in Android OS or Symbian, mainly due to the different nature of the devices which host these Operating Systems. The problem therein lies in the fact that these devices are made by different companies, such as Motorola, Acer, and Google, resulting in the user experience being quite similar, but not identical across all of them, thus requiring an additional effort in polishing the usability of our service applications for each particular device, in order for the user interfaces and interaction to be coherent.

Secondly, the iPhone has access to Apple’s “App Store”. This provides a vast potential user base, which we believe to be of great importance and potential benefit. Up until now, there are more than 133.000 third-party applications in the App Store, with more than 3 billion total downloads. This sets the iPhone market apart from its competition, and very much places it into a league of its own. To illustrate, the Android market, by December of 2009, had approximately only 20.000 applications.

Finally, one last important reason for our choice would be the robust programming language that is used for writing iPhone applications. Called Objective-C, a superset of the C language, it allows for excellent integration with C and C++, making the iPhone the ideal first candidate on which to develop and test a cross-platform, C++, Peer-to-Peer engine which could later be extended to run in Android OS, Symbian and various other systems which support C++.

## 5.7 Quality metrics for an overlay computer

The Arigatoni and BabelChord overlay networks are suitable to support various extended trust models. Moreover, reputation score could be expanded to a multi-dimensional value, for example adding a score for quality

of the service offered by an agent. However, they encourages cooperation and enables gratuitous resource offering. But it may also suit for business extensions, *e.g.*:

- An agent computer can sell resource usage, creating a resource business;
- An agent broker can sell a resource discovery service, creating a brokering business ( *“I point you to the best resources, more quickly than anyone else”* ).

The Arigatoni and BabelChord overlay network computer are 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 agent would then contact the auction server and get this information. Those extensions may take advantage of the RDP optional fields [BCLV06a], for example to transmit location and parameter information to call a third party system.

## 5.8 Trust and security

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 quality of the service offered by an agent. However, Arigatoni encourages cooperation and enables gratuitous resource offering. But it may also suit for business extensions, *e.g.*:

- An agent computer can sell resource usage, creating a resource business;
- An agent broker 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.

Another possible Arigatoni extension may define how to join a third party auction server. Candidate servants for a request 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 agent would then contact the auction server and get this information. Those extensions may transmit in the resource discovery protocol location and parameter information to call a third party system.

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 agents must be secured;

- The role played by an agent (*i.e.* client AC, servant AC or AB) must be certified by a third party trusted by the agents that communicate with this particular agent. 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 research statement, 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 agent in the overlay has in another agent based on its partial view. In a nutshell:
  - Each agent maintains a reputation score for each agent it knows;
  - Each agent maintains a reputation score for each resource it serves;
  - Exchanges between agents update dynamically each other’s scores;
  - Conflict between two or many agents are resolved by the brokers leaders of the colonies to which agents belong;
  - The computation of the reputation score (a trust metrics) and the way agents 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 they are barriers that can prevent connections between *inside* and *outside* agents in Arigatoni. The implementation of RFC3489 [RWHM03] could be used to overcome such obstacles.

## 5.9 New models of computations

The final vein, more abstract and long term, try to envisage new model of computations that encompasses the von Neumann machine, taking seriously into account networking issues as in Turing machines we take into account the number of tapes or the capability of the head to move and write on both directions.

**From large-scale computing machines to large-scale overlay network machines (John von Neumann was right before all).** This challenge is inspired by the seminal talk by John von Neumann, given in May 1946, “Principles of Large-Scale Computing Machines”, typesetted and 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<sup>2</sup> room, and 150 kW to operate. Today, thanks to the Moore’s law and to the Internet, “large scale” means “worldwide scale”, *i.e.* the computer hardware is distributed in space and in time and must be negotiated before being used. The main inspirations of the programmable overlay network computer research’s vein are still contained in that article.

The term “von Neumann bottleneck” was coined by John Backus in his 1977 ACM Turing award lecture. Bottleneck refers to the fact that, since data and program are stored on the same support (the memory), the throughput (data transfer rate) between the CPU and the memory is very low. In current von Neumann architecture the bottleneck is alleviated by using big cache memories. Since in overlay network computers

the bus can be modeled by an Internet connection, the data transfer is still more critical than on a single processor machine. As such, we should probably look at new computer architectures, such as the Harvard one.

Needless to say that the “icing on the cake” will be to formalize this new distributed computational model and architecture, together with a formal proof of its Turing completeness statement!

## 6 Topics and time line

### 6.1 Panorama

This schematic section gives some hints on the general research topic and categories and a tentative “time line” that are at the heart of our proposal. The purpose of this list is just to fit LogNet into a more general “*semantic context*” with the help of few general problems setting and thematic disciplines.

This should help the Management and Evaluators to better evaluate the originality, the pertinence, the validity, and the positioning of our proposal w.r.t. the INRIA Strategic Plan.

### 6.2 Topic view

#### 6.2.1 Vertical issues

- **Overlay network issues**

- management and self-organization of an overlay network with a dynamic topology
- structured *vs.* unstructured topology: stable *vs.* dynamic virtual organizations
- scalability of an overlay network and its relation with the number of users *vs.* the number of join/leave of users
- resource discovery and query routing in future overlay
- find algorithms and protocols for self-organize colonies (topic-based or load-balancing-based)
- find algorithms and protocols to get an *equilibrium* in self-organizing networks
- self-stabilization, self-organization, self-negotiation
- reputation and trust protocols, fault tolerance
- dynamic graph theory
- distributed algorithms for local cycles/knot detection under a partial graph knowledge

- **Execution issues**

- symbolic design of a Java library for orchestrate negotiated resources : implementation of the library (or the precompiler)
- build a logical software layer in order to make hardware and software resources available in the P2P phase

#### 6.2.2 Horizontal issues

- **Arigatoni and BabelChord issues**

- broker discovery using underlay network techniques
- efficient pattern matching algorithms to filter resource routing tables
- design of a first-order query language to denote resource queries
- relaying content-based routing on a IP, MANET, wireless, or cellular-based routing

- self-organizing policies for load balancing and for minimizing routing path in RDP requests and for fitting the overlay (logical) layer over the physical (transport) layer
- “higher-order” broker discovery in RDP (ask a broker for another broker)
- crossing firewalls during the P2P interaction
- social-based strategies to organize floors and synapses

### 6.2.3 Diagonal issues

- communication, activation, and orchestration protocols
- crossing system administration barriers
- find algorithms and protocols for self-organize colonies (topic-based or load-balancing-based)
- self-stabilization, self-organization, self-negotiation
- security, reputation and trust protocols, fault tolerance
- find new distributed computational models, architectures, and distributed programming paradigm adapted to the new ubiquitous nature of the computation (internet-of-things-inspired)
- find alternative computer architecture that can fit with the distributed nature of an overlay network computer (*e.g.* the Harvard architecture with cache)
- formally prove the Turing completeness of such models and architectures!

## 6.3 Time view

### 6.3.1 Short-term

- structured *vs.* unstructured overlay networks
- resource discovery and query routing in future overlay
- publish subscribe over P2P networks
- management and self-organization of an overlay network with a dynamic topology
- broker discovery using underlay network techniques
- scalability of an overlay network and its relation with the number of users *vs.* the number of join/leave of users
- algorithms and protocols for self-organize colonies (topic-based or load-balancing-based)
- social networks
- Crosslayer-oriented design
- Use of femtocells as overlay agents



### 6.3.2 Medium-term

- efficient pattern matching algorithms to filter resource routing tables
- design of a first-order query language to denote resource queries
- relaying content-based routing on a IP, MANET, wireless, or cellular-based routing
- self-organizing policies for load balancing and for minimizing routing path in RDP requests and for fitting the overlay (logical) layer over the physical (transport) layer
- social-based strategies to organize floors and synapses
- symbolic design of a Java library (or a Java dialect) for orchestrate negotiated resources : implementation of the library (or the precompiler)
- mobile social networks
- underlay-aware interconnection of heterogeneous networks

### 6.3.3 Long-term

- reputation and trust protocols, fault tolerance
- crossing system administration barriers
- find algorithms and protocols for self-organize colonies (topic-based or load-balancing-based)
- find algorithms and protocols to get an *equilibrium* in self-organizing networks
- further elaborate semantics of our communication model of component based on asynchronous messages and one thread per component
- cooperation in the vehicle-to-infrastructure architecture
- find new distributed computational models, architectures, and distributed programming paradigm adapted to the new ubiquitous nature of the computation (internet-of-things-inspired)
- find alternative computer architecture that can fit with the distributed nature of an overlay network computer (*e.g.* the Harvard architecture with cache)
- formally prove the Turing completeness of such models and architectures!

## 7 Potential application domains

### 7.1 Panorama

Because of its generality, overlay networks can target many applications. We would like to list a small list of useful programmable overlay networks case of study that can be considered as “LogNet Grand Challenges” to help potential readers to understand the interest of our research program.

- P2P social networks
- Overlay computer for mobile *ad hoc* networks
- Reduce the digital divide
- Grid applications

- New distributed models of computation
- Interconnection of heterogeneous overlay networks
- Towards an overlay network of things

## 7.2 P2P social networks

See contracts section and potential software.

## 7.3 Overlay computer for mobile *ad hoc* networks

We plan to build an *ad hoc* vehicular network infrastructure using the Arigatoni overlay infrastructure. That network must enable efficient and transparent access to the resources of on-board and roadside agents. In such a scenario, commercial services and access to public information are 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. These new features will offer innovative functions and services, such as:

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

In this scenario, vehicles/pedestrians play the role of agent computers, while Bus-stop stations equipped with IP network, routing tables and WIFI access point play the role of agent brokers; Buses play the role of mobile agent brokers, a sort of proxy of a unique bus-stop agent broker. Proxy load balancing policies are left to the bus headquarter (HQ).

*Arigatoni on wheels* (Ariwheels for short) is an overlay architecture designed for a vehicular network underlay environment. Ariwheels provides efficient, transparent advertising and retrieves resources carried by on-board and roadside nodes. Consider an urban area in which a Mobile Ad hoc Network (MANET) is deployed. Such MANET is populated by both mobile users, *e.g.* pedestrians with hand-held devices, cars equipped with browsing/computational capabilities, public-transportation vehicles and roadside infrastructures such as bus stops. All devices are supposed to have a wireless interface. Depending on their mobility, they may also be equipped with a wired interface. Such is the case of wireless Access Points (APs), which are installed at a bus stop, in order to provide connectivity either to users waiting for a bus or to the bus itself (hence to its passengers). In such settings, devices carried by cars and pedestrians play the role of mobile agent computers; roadside infrastructures (APs) and public transportation vehicles (buses, trams, cabs...) act as agent brokers, although some distinctive behaviors have to be introduced.

An agent broker in Ariwheels is logistically represented by a bus stop, and its colony is composed by mobile agent computers that have registered to it when they were within radio range of the AP installed at the bus stop. However, to take into account the high mobility of the scenario and enhance its performance in terms of load balancing and service response time, we introduce an additional, Ariwheels-specific entity, the *mobile agent broker* (mAB). This unit is a public transport vehicle equipped with a scaled-down broker-like wireless device. Every mobile agent broker is *associated* to (*i.e.*, it has the same identity of) a single agent broker. Such association exists at the overlay level and holds throughout its bus route. Clearly, at the underlay level, connectivity between the mobile agent broker and the associated agent broker may at times be severed.

The main aim of the mobile agent broker is to introduce the novel concept of “*colony-room*”: a small subset of mobile agents computers with a wireless connection to the mobile agent broker (pedestrian users on the bus, or pedestrian/vehicles around the bus or traveling along the same bus direction during a traffic jam...). In addition, thanks to its mobility, the mobile agent broker can collect registrations from mobile agent computers that were too far from the AP of the associated agent broker, and, therefore, might have never had the chance to register to it. The mobile agent broker operates in tight coordination with infrastructure devices, *i.e.*, with agent brokers, and acts, in effect, as a colony-room for one of them.

The mobile agent broker collects (un)registrations, service requests and service offers from the agent computers within the colony-room. When a wireless connection has been established between the mobile agent broker and a roadside AP (not necessarily corresponding to the associated agent broker), the data path to the associated agent broker is again available and an information exchange takes place resulting in the updating of each other’s data. Specifically, the following actions occur. Firstly, the associated agent broker merges the mobile agent broker’s routing table with the one it currently carries. Then, the associated agent broker handles the registration/discovery information and generates the appropriate responses. Finally, depending on the response time, the responses are returned to the mobile agent broker before it leaves the wireless AP coverage, or the next time it connects to an AP: this will normally happen at the next bus stop.

Wireless devices may either be user terminals (laptops, hand-helds, sensors...) or higher-level units providing connection to the entities within the Ariwheels architectures. mABs will be able to provide, via the RDP protocol, identities of ACs in its local area offering (in a P2P fashion) various information specific to the area where the bus stop is located, such as movie listings of local theaters, or lunch menus of nearby restaurants, or traffic jams. Therefore, mobile devices carried by passengers on a bus act as ACs registered to a mAB (the AP on the bus), and may primarily exchange information among themselves. If information cannot be found among ACs in the subcolony, SREQs are relayed to the AB of which the mAB is a subcolony. As a consequence, a mAB holds a subset of routing table entries that can be found on the AB. Mobile users may want to access the wealth of information available through the ABs or the mAB, by first subscribing to the colonies governed by ABs, and then by sending service requests to (mobile) agent brokers.

Figure 1 illustrates the relationships among overlay and underlay entities in Ariwheels. A central coordination entity is located at a headquarter (HQ), in our case corresponding to the local transportation authority building. The coordination entity plays the role of a super-broker and it is provided with a wired connection to each of the 4 roadside AP at bus stops (tagged B1 to B4). Mobile agent brokers (tagged mB1 and mB3) shuttle between bus stops, each carrying a different broker association (tagged B1 and B3), while mobile agents (portable devices or on-board car devices in the figure) are either connected to brokers or mobile agent brokers, depending on their mobility. This figure also shows the simulated city topology, that featured 6 bus stops with APs, each corresponding to an agent broker. Furthermore, 3 buses acting as mobile agent brokers weave their own routes across the topology, among a population of as many as 60 vehicles acting as mobile agent computers. Each bus carries 10 passengers equipped with mobile agent computers capabilities, and it associates to the agent broker with the smallest colony at the time of departure from the bus station.

In the simulator of Figure 2 the color’s semantics is:

- BLUE = Car already logged to a Broker (Bus Stop).
- PINK = Car that has lost WIFI connection with the Broker (Bus Stop).
- GREEN = Car that is looking for a Broker (Bus Stop).

The Ariwheels overlay network is being proposed as a publish & subscribe protocol in the vehicular platform under development in the VICSUM project (2Meur, founded by the Regione Piemonte) led by Politecnico di Torino and involving the Centro Ricerche Fiat (CRF) and the Centro Supercalcolo Piemonte [LBCC08, BCCL08]. (Successful) results will be suitable to be integrated in the new BLUETOOTH system device Blue&Me™ by Fiat&Microsoft. The project will exploit the availability of existing urban infrastructure and public transportation vehicles (pledged by GTT – Gruppo Torinese Trasporti, the Torino’s public bus and metro).

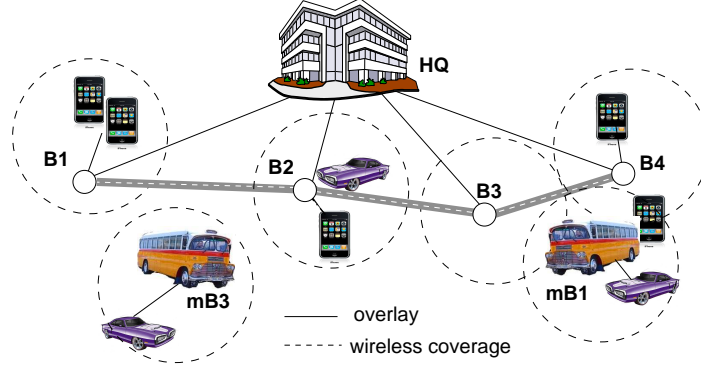


Figure 1: An Ariwheels scenario

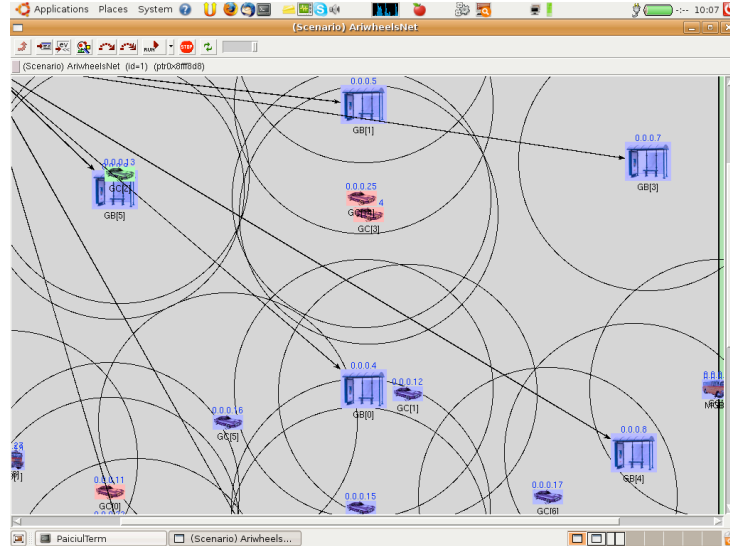


Figure 2: The Ariwheels simulator

#### 7.4 OverStic: the mesh overlay network in Sophia Antipolis

The future Campus STIC, grouping EPU, UNSA, Eurecom, CNRS, and INRIA will be ready in one year. It will be equipped with a WIFI network infrastructure implementing 802.11a/b/g protocols, with potential evolution to 802.11n protocol. The main objectives of such underlay network are to offer IP connection to all Campus “citizens”: the network must guarantee the respect of French laws concerning public network connections (*décret 2006-358 sur l’offre de connexion au public loi 2006-64*). To do this, it would be

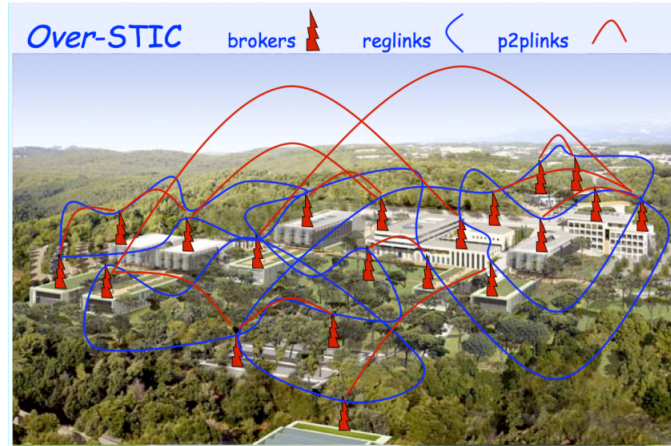


Figure 3: Overstic

suitable that all users get identified using, *e.g.*, using the “pin” code of the student/employee-card. The infrastructure mainly targets Internet access for all. The Campus STIC WIFI underlay network could be an unique opportunity to have a real testbed into which put our programmable overlay at work. *Arigatoni* and *Ariwheels* could represent the overlay network infrastructure to offer *much more* than simply an Internet connection: the LogNet vision can provide a list of interesting high-level semantic (on demand) services, and a plausible way to implement it, see Figure 3.

## 7.5 Reducing the *Digital Divide*

The digital divide is the troubling gap between those who use computers and the Internet and those who do not. The term digital divide had a moving target: first, it meant the ownership of a computer. Later and access to the Internet. Most recently it centers on broadband access. In modern usage, the term also means more than just access to hardware, it also refers to the imbalance that exists amongst groups of society regarding their ability to use information technology.

The digital divide tends to focus on access to hardware, access to the Internet. The writer Lisa J. Servon argued in 2002 that the digital divide “is a symptom of a larger and more complex problem – the problem of persistent poverty and inequality”. The four major components that contribute to digital divide are “socioeconomic status, with income, educational level, and race among other factors associated with technological attainment”.

One significant focus was *school computer access*; in the 1990s, rich schools were much more likely to provide their students with regular computer access. In the late 1990s, rich schools were much more likely to have Internet access. In the context of schools, which has constantly been involved in the discussion of the divide, current formulations of the divide focus more on how (and whether) computers are used by students, and less on whether there are computers or Internet connections.

The USA E-rate program (officially the Schools and Libraries Program of the Universal Service Fund), authorized in 1996 and implemented in 1997, directly addressed the technology gap between rich and poor schools by allocating money from telecommunications taxes to poor schools without technology resources. Though the program faced criticism and controversy in its methods of disbursement, it did provide over 100,000 schools with additional computing resources, and Internet connectivity [Sources: Wikipedia].

Recently, discussions of a digital divide in school access have broadened to include technology related skills and training in addition to basic access to computers and Internet access. An interesting example is that, in the North of Italy, the town of Pordenone, 50,000 citizens, will be equipped with public local WIFI

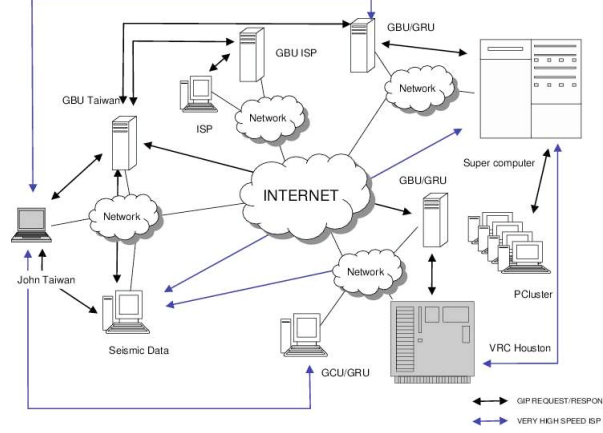


Figure 4: Arigatoni Overlay Network for a Grid Seismic Monitoring Application

LAN (*e.g.* see the declaration of the Major, in Italian, (<http://it.youtube.com/watch?v=zBTnkEnXT1c>)). Our vision could contribute to reduce digital divide in our society and more contextually in the future Campus STIC.

To give an idea of possible usage of the Arigatoni generic overlay network we present two examples; the first one has a Grid-computing flavor while the second is a nice interweaving of the Arigatoni overlay seated on the top of both IP and MANET underlay network. For more information the interested reader can have a look on [BCLV06a] and [LBCC08, BCCL08].

## 7.6 GRID applications : scenario for seismic monitoring

The example below show how overlay networks can be fruitfully employed for Grid computing.

John, chief engineer of the SeismicDataCorp Company, Taiwan, on board of the seismic data collector ship, has to decide on the next data collect campaign. For this he would like to process the 100 TeraBytes of seismic data that have been recorded on the data mass recorder located in the offshore data repository of the company to be processed and then analyzed. He has written the processing program for modeling and visualizing the seismic cube using some *parallel library* like *e.g.* MPI or PVM: his program can be distributed over different machines that will compute a chunk of the whole calculus; however, the amount of computation is so big that a supercomputer and a cluster of PC has to be *rented* by the SeismicDataCorp company. John will ask also for *bandwidth* in order to get rid of any bottleneck related to the big amount of data to be transferred. Then, the processed data should be analyzed using the *Virtual Reality Center*, (VRC) based in Houston, U.S.A. by a specialist team and the resulting recommendations for the next data collect campaign have to be sent to John. As such:

1. John logs on the Arigatoni Overlay Network in a given colony in Taiwan, and sends a quite complicated service request in order for the data to be processed using his own code. Usually the AB leader of the colony will receive and process the request;
2. If the Resource Discovery performed by the AB succeeds, *i.e.* a supercomputer and a cluster and an ISP are found, then the data are transferred at a very high speed and the “*Sinfonia*” begins;
3. John will also ask (in the RDP request) to the AC containing the seismic data to dispatch suitable chunks of data to the supercomputer and the cluster designated by the AB to perform some pieces of computation;
4. John will also ask (in the RDP request) to the supercomputer the task of collecting all intermediate results so calculating the final result of the computation, like a “*Maestro di Orchestra*”;

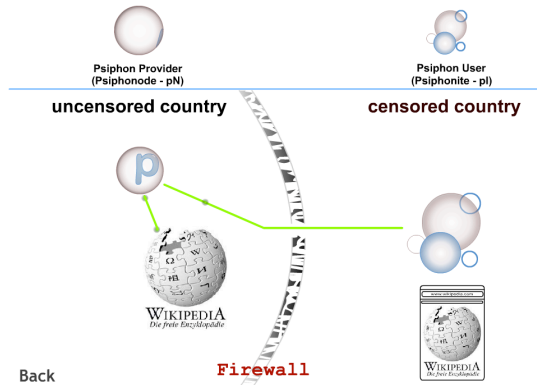


Figure 5: Psiphon

5. The processed data are then sent from the supercomputer, via the high speed ISP, to the Houston center for being visualized and analyzed;
6. Finally, the specialist team's recommendations will be sent to John's laptop.

This scenario is pictorially presented in Figure 4 (we suppose a number of sub-colonies with related leaders AB, all registered as agents to a super-AB; for example the John's AB could be elected as the super-leader). For simplify security issues, all AB's are trusted using the same PKI, making *de facto* in common all resources of their colonies. An animation of the coordination program, written in the visual language JOpera can be downloaded at ([http://www-sop.inria.fr/members/Luigi.Liquori/ARIGATONI/arigatoni\\_animation.wmv](http://www-sop.inria.fr/members/Luigi.Liquori/ARIGATONI/arigatoni_animation.wmv)).

## 7.7 Interconnection of heterogeneous overlay networks

Because of the original features of BabelChord, the following are examples of applications for which BabelChord can provide a good groundwork (in addition, of course, to all genuine Chord-based applications, like cooperative mirroring, time-shared storage, distributed indexes and large-scale combinatorial search).

**Anti Internet censorship applications.** Internet censorship is the control or the suppression of the publishing or accessing of information on the Internet. Many applications and networks have been recently developed in order to bypass the censorship: among the many we recall Psiphon (<http://psiphon.ca>), Tor (<http://www.torproject.org>), and many others. BabelChord can support such applications by taking advantage of intra-floor routing in order to bypass software barriers.

**Fully Distributed social-networks applications.** Social-networks are emerging as one of the Web 2.0 applications. Famous social networks, such as Facebook or LinkedIn are based on a client-server architecture; very often those sites are down for maintenance. BabelChord could represent a scalable and reliable alternative to decentralize key search and data storage.

**Large-scale brain model and simulations.** (Via a distributed, neural-based, network.) As well explained by R.D. DeGroot (Project founded by KNAW, Netherlands), supercomputers exist now with raw computational powers exceeding that of a human brain. Technological and production advances will soon place such computing power within the hands of cognitive and medical neuroscience research groups. For the first time it will be possible to execute brain-scale simulations of cognitive and pharmacological processes over millions



Figure 6: Social Media Landscape

and then billions of neurons - even at the biological model level. BabelChord could help modeling as a meta-overlay network the human brain.

## 7.8 Toward an overlay network of things (RFID)

Since resource discovery is at the hearth of our research program, an interesting application is to study overlay networks where agents can be either objects with related RFID, RFID's card readers, or RFID's databases and RFID-end users; the overlay network of things is built over an underlay network (IP, MANET, ...). The overlay can be structured or unstructured, and the query routing will be strongly affected by this topology choice. To that end, a first challenge is

- to find efficient routing protocols to dispatch queries over the overlay network of things;
- to find filter mechanisms, based on pattern-matching, suitable for “aggregating” non structured query results into structured and semantic ones in order to present the results in an intuitive and organized way;
- to efficiently map the overlay network of things into the underlay (transport) network in order to minimize the average cost of physical hops required for a single “logical” hop;



```

mascotte~/ARIGATONI/CODE - Konsole
Session Edit View Bookmarks Settings Help

./a.out: /usr/local/lib/libstdc++.so.6: no version information available (required by ./a.out)
./a.out: /usr/local/lib/libstdc++.so.6: no version information available (required by ./a.out)
1 topology: 1 node = 3 <- 3 0 <- 3 0 <- 3 123 <- 3 123 <- 3 10000 <- 3 1 <- 3 100 <- 3 100
100%
ALL CORRECT
AVERAGE # PEERS INVOLVED PER REQUEST (GRU+ ONLY) | GRU+ & GCU+ : 19.7818 26.6485
AVERAGE # REQ MESSAGES PER REQUEST (GRU+ ONLY) | GRU+ & GCU+ : 20.6098 22.4714
AVERAGE # MESSAGES (ALL TYPES) PER REQUEST (GRU+ ONLY) | GRU+ & GCU+ : 68.0267 74.8867
MEAN # REQUESTS FOR DEPTH 0: 10000
MEAN # REQUESTS FOR DEPTH 1: 4131
MEAN # REQUESTS FOR DEPTH 2: 2067.71
MEAN # REQUESTS FOR DEPTH 3: 868.589
MEAN # REQUESTS FOR DEPTH 4: 506.055
MEAN # REQUESTS FOR DEPTH 5: 404.25
MEAN # REQUESTS FOR DEPTH 6: 313.842
MEAN # REQUESTS FOR DEPTH 7: 188.873
MEAN # REQUESTS FOR DEPTH 8: 0
MEAN: 951.096 MIN: 0 MAX: 10000 (100) STD DEV: 1625.7 MEDIAN: 43
ACCEPT RATE: 0.919375 REJECT RATE: 0.080625 UNARY ACCEPT RATE: 0.991244 UNARY REJECT RATE: 0.00875567
5

```

Figure 7: The Arigatoni simulator

- to fix a “logic language” on ground terms and *ad hoc* ontologies on well-founded queries suitable to fully describe a semantic query into the overlay network of things;
- to translate a complex semantic query into a sequence of protocol messages routed in the overlay network of things.

To be adopted, an overlay network should address some fundamental network issues like scalability, fault tolerance, resistance to denial of service attack, privacy, security at any level of the supply chain: in other words, participation to the overlay, query routing and result aggregation, must be accessible only to authorized persons and security mechanism must prevent or detect unauthorized persons to know about or perform certain access to or modification to data. Crossing firewalls is an important issue that must be taken into account. Merging different overlays (*e.g.* of different companies) is an issue that could be also worth of interest.

## 8 Software

### 8.1 Panorama

We divide this section into two subsections: Prototype software, *i.e.* software that already we have (at least in a rough form) and potential software, *i.e.* software that potentially we can build.

### 8.2 Prototype software

#### 8.2.1 Arigatoni simulator

We have implemented in C++ (~2.5K lines of code) the Resource Discovery Algorithm and the Virtual Intermittent Protocol of the Arigatoni Overlay Network. The simulator was used to measure the load when we issued  $n$  service requests at Global Computers chosen uniformly at random. Each request contained a certain number of instances of one service, also chosen uniformly at random. Each service request was then handled by the Resource Discovery mechanism of Arigatoni networks.

#### 8.2.2 Ariwheels

Ariwheels is an infomobility solution for urban environments, with access points deployed at both bus stops (forming thus a wired backbone) and inside buses themselves. Such a network is meant to provide connectivity and services to the users of the public transport system, allowing them to exchange services, resources and information through their mobile devices. Ariwheels is both:



the city section accordingly. A vehicle entering the topology is assigned a top speed of  $v$  m/s, that it tries to reach and maintain, as long as traffic conditions and road signs allow it to. When a vehicle reaches its destination, it stops for a random amount of time, uniformly distributed between 0 and 60 s, then it re-enters the city section. In our simulations, we tested two different top speeds  $v$ : 9 m/s (approx. 32 km/s) and 15 m/s (approx. 54 km/s).

Upon entering the topology, a vehicle acting as Mobile Agent owns a set of 12 unitary services (*e.g.*, files, traffic informations, point of interests) randomly chosen from a set of 20 services. A Mobile Agent issues a (SREQ) for a service it is missing and the inter-request time is supposed to be exponentially distributed with parameter  $\lambda = 0.05$  [req./s]. As typical in the publish/subscribe paradigm, where peers are not slaves, upon receiving a SREQ for a service it owns, a Mobile Agent sends back a positive response with a certain probability, which is set to 0.9 in our simulations.

The simulated city topology, shown in Figure 8, features 6 bus stops with APs, each corresponding to a Broker. Furthermore, 3 buses acting as Mobile Brokers weave their own routes across the topology, among a population of as many as 60 vehicles acting as Mobile Agents. Each bus carries 10 passengers equipped with Mobile Agent capabilities, and it associates to the Broker with the smallest colony at the time of departure from the bus station. Brokers apply the unbalanced acceptance policy and filter the routing table against a received query by using the liveliness information only.

#### Network client.

**Scenario.** Ariwheels is designed for the scenario of urban public transportation. In such a scenario, a significant number of users equipped with mobile devices spends significant amounts of time at the bus stops or inside the bus themselves. The basic idea of Ariwheels is to exploit this situation to let the users exchange data or services - more generally: resources - through their mobile devices.

**Infrastructure.** Ariwheels is based on the 802.11 wireless LAN protocols. Therefore, its infrastructure is mostly made of access points, deployed both:

- at bus stops, forming a backbone;
- on the bus themselves, thus with an intermittent connection to the backbone.

The infrastructure also includes the IP network connecting the coverage areas of the access points, and some higher-level coordination facilities. Nodes and software

Network nodes will be quite ubiquitous, including bus stops, buses and passengers, *i.e.* the ones equipped with a suitable mobile device. Most nodes will be mobile (*i.e.* they move) and dynamic (*i.e.* they can suddenly be turned off, or leave the network itself). Owing to the peculiarities of its nodes, the Ariwheels network falls in the category of mesh networks.

All nodes will run *ad hoc* pieces of software. In other words, installing the Ariwheels software makes the difference between a node of the underlay network (802.11, IP) and a node of the overlay Ariwheels network.

**Solution.** Ariwheels have four kinds of functional units, namely agents, brokers, mobile brokers, and proxies.

These units interact according to a protocol based on the publish/subscribe paradigm.

**Agent.** The agent is a software - written in C# for better compatibility - running on the user's device. It will run in user space and unprivileged mode, in order to require no additional configuration or permissions. Using appropriate sensing and probing mechanisms, the agent will look for a Broker. Once found one, it performs:

- the registration, which includes sending a list of the resources the agent has to offer;
- the request of the resources the agent needs.

Registration is performed only once - *i.e.* once every time the agents meets a broker. Resource requests are usually repeated several times, until all the needed resources are found. In addition to this seeking activity, the agent has to provide the services it claimed to be providing to the agents requesting them.

**Broker.** The broker is a program, written in C for better performance, running on a mid- or high-end device. There must be (at least) one broker in each L2 network belonging to the Ariwheels system. The Broker performs four main duties:

1. advertise its presence and the resources available through it;
2. receive, elaborate and acknowledge the registration requests coming from the Agents;
3. receive and (try to) answer the resource request coming from the Agents;
4. manage feedback and reputation.

Internally, the broker is equipped with an embedded database (namely SQLite), which is updated at each received packet. The information contained in the packets, as well as the information represented by the packet itself is combined in a (sort of) routing table. For each resource, it contains ID and IP address of the agent which will be asked to provide it. This agent will be chosen among the ones providing the resources according to a policy (hopefully) balancing availability and fairness. After identifying the agent which will have to provide the resource, the broker pings it, in order to avoid committing the supplying of the resource to an agent which is not active anymore.

**Mobile broker.** Mobile brokers are brokers with an intermittent connection to the rest of the network. A typical example is a bus equipped with a wireless access point, connecting - when possible - to the infrastructure, deployed at some stops. Mobile brokers are associated with a fixed broker. As soon as this broker becomes available (*i.e.* the mobile broker can hear its Hello messages), the mobile broker sends it one or more Dump messages, containing its routing table. The fixed broker replies dumping its own routing table. As a result, both mobile and fixed brokers know which services are available through the other. As long as the connection lasts, the two brokers will use this information to answer the service requests they cannot satisfy using their own routing tables. The priority order is:

1. the broker's own table;
2. the table dumped by mobile/fixed brokers;
3. forwarding the request to the proxy parent (see below), if any.

When the connection with a broker is lost, the routing entries relating to it are flushed.

**Proxy.** Proxies are the way Ariwheels copes with the need to access information outside the colony. The following basic principles hold:

- brokers only store information about their own colony;
- brokers are the only entity storing information.

As a consequence, there is no such thing as a super-broker, *i.e.* a node having global (or higher-level) information about the network. The rationale for this is that the rate at which the network changes is comparable to the time needed to propagate such information. Lacking super-brokers, agents still have the opportunity to consume services provided outside the colony they belong to. The Proxy node will handle the forwarding of SREQ's and SRESP's across colony borders, according to the schema:

1. brokers know in advance their parent proxy, and regularly send it Proxy packets;
2. if a broker is unable to answer a SREQ from one of its agents, it forwards it to its proxy;
3. the proxy forwards the SREQ to all its children brokers;
4. brokers reply to this forwarded SREQ only if they know how to gather the requested service;
5. if a SRESP arrives, the proxy forwards it to the broker having originated it;
6. otherwise, after a timeout, sends to the originating broker an empty SRESP;
7. the broker forwards the response it receives from the broker, either full or empty, to its agent.

**Basic interaction.** The most basic interaction between two agents and a broker foresees the following steps:

1. agent B registers with the broker, declaring to provide (among others) a given resource R;
2. agent A registers;
3. agent A queries the broker for resource R;
4. the broker looks up its routing table, and chooses B to provide R to A;
5. the broker transmits to A the details of B (basically, its IP address);
6. A contacts B, asking it for resource R;
7. B provides resource R to A.

Note that the actual data exchange between A and B happens in a fully peer-to-peer fashion, and does not involve the broker. Additionally, the broker itself does not provide any service: it only knows who could provide it. Advertising

Although not included in the basic interaction, the advertising mechanism - *i.e.* how does the Agent get to know that there is a Broker out there? has an important role. The solution adopted in Ariwheels is an Hello/Probe mechanism:

- the Brokers send, at random intervals, a Hello packet, which includes the list of the resources available through them;
- the Agents which are interested in looking for a broker (either because they know none, or because they want to change their current one) send a Probe packet;
- the Brokers receiving a Probe packet answer with a (unicast) Hello packet.

Advertising mechanisms usually foresee the usage of broadcast traffic. However, since Ariwheels is built upon IP, it can profit by the often neglected feature of multicasting. Hello and Probe packets are sent to distinct, well-known multicast addresses. The underlay network is expected to be configured in such a way that multicast packets are not forwarded outside the L2 network they originate in. As a result, only agents interested in knowing new brokers will join the Hello multicast group. Additionally, such a group will be also joined by brokers interested in knowing other brokers.

See the web page <http://www-sop.inria.fr/members/Luigi.Liquori/ARIGATONI/Ariwheels.htm> and <http://arigtt.altervista.org>.

### 8.2.3 BabelChord

**Simulator.** To better capture its relevance, we have conducted some simulations of the BabelChord approach. The simulator, written in Python, works in two phases. First, a BabelChord topology is created, with the following properties: (i) a fixed network size (the number of nodes)  $N$ , (ii) a fixed number of floors denoted  $F$ , (iii) a fixed global *connectivity*, *i.e.*, the number of floors each node belongs to, denoted  $C$ . As a consequence: (i) The nodes are uniformly dispatched among the floors, *i.e.*, each node belongs to  $C$  floors uniformly chosen among the set of floors. (ii) Each resource provided by nodes is present at  $C$  floors. (iii) The average lookup length within one given floor is  $\log((N \times C)/F)/2$ .

In a second time, the simulator computes the number of hops required to reach one of the node storing one of the key of a particular resource. Results are given for different values of  $N$ ,  $F$ , and  $C$ . Figure 9 shows the number of synapses vs. the lookup success rate. **Note that only 5% of synapses made of 2 (resp. 3, 5, 10) floors connections in the whole node population is enough to achieve more than 50% (resp. 60%, 80%, 95%) of exhaustive lookups in the BabelChord network!**

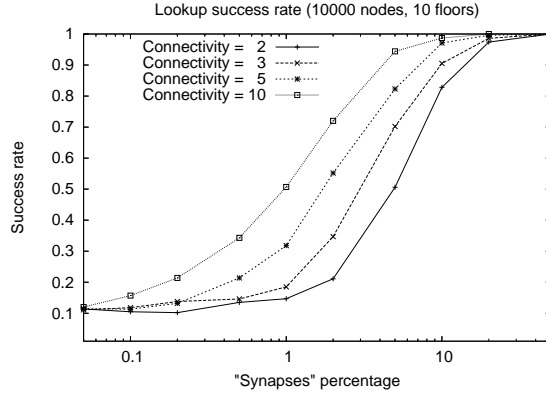


Figure 9: Exhaustiveness,  $N=10000$

**Implementation.** A BabelChord client built on the top of the OpenChord software is currently under development in the team. A rigorous object-oriented methodology will make this client to be fully compatible with OpenChord. Using Inheritance mechanism and exploiting subtyping and late binding, a huge part of the OpenChord code will be recycled.

```

C:\WINDOWS\system32\cmd.exe - console.bat
Listening for transport dt_socket at address: 8100
This program is free software; you can redistribute
it and/or modify it under the terms of the GNU General
Public License as published by the Free Software
Foundation; either version 2 of the License, or (at
your option) any later version.
A copy of the license can be found in the license.txt
file supplied with this software or at:
http://www.gnu.org/copyleft/gpl.html
Welcome to Open BabelChord test environment. (extending OpenChord)
(C) 2004-2006 Distributed and Mobile Systems Group
University of Bamberg
Type 'help' for a list of available commands
Console ready.
oc > create -names peer
Creating new babelchord network.
oc > create -names peer1 peer2 -bootstraps peer
Starting node with name 'peer1' with bootstrap node 'peer'
Starting node with name 'peer2' with bootstrap node 'peer'
oc >

```

Figure 10: BabelChord console

#### 8.2.4 Synapse

##### Simulator

To better capture its relevance, we have conducted some intensive simulations of the Synapse approach [LTV<sup>+</sup>10]. The simulator, written in Python, follows a discrete time approach. First, an initial topology is created, with a specified number of synapses, each having a specified different degree. Then, some discovery queries are sent. At each discrete time step, a message sent at the previous step is received by its destination (next routing step for the sought key). This simulator takes churn into account; at each time step, some events affect the network: some new nodes join the network, some existing nodes leave it. This simulator has been used to have a better and deep understanding of Synapse-like architectures, interconnecting structured overlay networks in a simple ways: routing latency and communication overhead while changing the input parameters (number of nodes, synapses, degree of synapses, level of churn), see <http://www-sop.inria.fr/lognet/synapse/pysynapse/pysynapse.zip>.

##### Implementation

In order to test our Synapse protocol [LTV<sup>+</sup>10] on real platforms, we have initially developed JSynapse, a Java software prototype, which uses the Java RMI standard for communication between nodes, and whose

purpose is to capture the very essence of our Synapse protocol. It is a flexible and ready-to-be-plugged library which can interconnect any type of overlay networks. In particular, JSynapse fully implements a Chord-based inter-overlay network. It was designed to be a lightweight and easy-to-extend software. We also provided some practical classes which help in automating the generation of the inter-overlay network and the testing of specific scenarios. We have experimented with JSynapse on the Grid'5000 platform connecting more than 20 clusters on 9 different sites. Again, Chord was used as the intra-overlay protocol.

We used one cluster located at Sophia Antipolis, France. The **Helios** cluster consists of 56 quad-core AMD Opteron 275 processors linked by a gigabit Ethernet connection. The created Synapse network was first made up of up to 50 processors uniformly distributed among 3 Chord intra-overlays. Then, still on the same cluster, as nodes are quad-core, we deployed up to 3 logical nodes by processor, thus creating a 150 nodes overlay network, nodes being dispatched uniformly over 6 overlays. During the deployment, overlays were progressively bridged by synapses (the degree of which was always 2).

We give a proof of concept and show the viability of the Synapse approach while confirming results obtained by the simulation. We also focus on the metrics affecting the user (satisfaction ratio and time to get a response). Once his request was sent, a user waits only for 1 second before closing the channels opened to receive responses. If no response was received after 1 second, the query is considered as not satisfied, see <http://www-sop.inria.fr/lognet/synapse/jSynapse/index.html>.

### 8.2.5 Open-Synapse Client

Opensynapse is an open source implementation of [LTV<sup>+</sup>10]. It is available for free under the GNU GPL. This implementation is based on Open Chord (v. 1.0.5) - an open source implementation of the Chord distributed hash table implementation by Distributed and Mobile Systems Group Lehrstuhl fuer Praktische Informatik Universitaet Bamberg. <http://www-sop.inria.fr/lognet/synapse/open-synapse/index.html>.

Opensynapse is implemented on top of an arbitrary number of overlay networks. Inter-networking can be built on top of Synapse in a very efficient way. Synapse is based on co-located nodes playing a role that is reminiscent of neural synapses. The current implementation of Opensynapse in this precise case interconnects many Chord overlay networks. The features of Opensynapse are:

- Stores any serializable Java object within a set of distributed hash tables (dhts).
- Facilitates configurable replication of entries in the dht.
- Currently provides two (proprietary) protocols for communication:
  - Local method calls: This protocol can be used to create a dht within one Java Virtual Machine for testing and visualization purposes.
  - Java Sockets: This protocol creates a dht distributed over different nodes (JVMs).

The new client currently can interconnect an arbitrary number of Chord networks. This implementation follows the notation presented in [LTB09], and so, each new Chord network is called a *Floor*. Regarding the **open-chord** implementation, some new classes were implemented, such as **Floor** or **MyFloor**. The rest of the code is changed only to be compatible with the new data structure, that References and Entries are specific for a particular floor. Major changes were made in the main classes **NodeImpl** and **ChordImpl**, as well in the communication part, in form of specific proxy classes: **SocketProxy** with **RequestHandler** and **ThreadProxy** with **ThreadEndpoint**.

Following the idea that every node is potentially a neural synapse, the decision was made not to implement a full object-oriented extension of the classes, but only to change the **open-chord** implementation, because the new classes which should extend the old ones would have almost the same code as the old ones with the only novelties being the calls to these altered structures. So, in this case we do not have a real full object-oriented extension, we just deal with some sort of siblings classes.



Figure 11: myTransport on the Nokia N800 Internet tablet

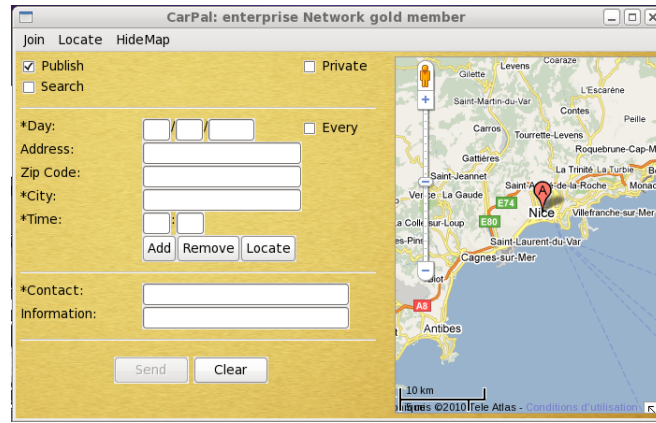


Figure 12: CarPal Gold member tablet

### 8.2.6 myTransport Gui

myTransport is a GUI built on top of the Synapse protocol and network. Its purpose is to be a proof of concept of the future service of infomobility to be available in the myMed social Network, see Figure 11. The GUI is written in Java and it is fully functional in the Nokia N800 internet tablet devices.

### 8.2.7 CarPal: a P2P car pooling service

This year, we put “on the road” the recent Synapse protocol and routing algorithms with CarPal, a proof-of-concept for a mobility sharing application that leverages a Distributed Hash Table to allow a community of people to spontaneously share trip information without the costs of a centralized structure. The peer-to-peer architecture allows moreover the deployment on portable devices and opens new scenarios where trips and sharing requests can be updated in real time. Since Synapse allows to interconnect different overlays/communities, the success rate (number of shared rides) can be boosted up thus increasing the effectiveness of our solution. An iPhone development is in progress. See Figure 12. Unstable. See <http://www-sop.inria.fr/lognet/carpal/>.



```
-bash-3.2$ ./husky1.5

-----
      /\_/\
      (oo)\_____)
      (______)\
      (__)\       )\/\
      ||----w |
      ||     ||

      jgs
The First Ascii-Oriented Programming Language
Based on the Original Screenplay of the
Imperative Rewriting Calculus v 1.1

Kernel Certified by Coq
Powered by OCaml
Copyright INRIA 2009

NoEffect Theory Loaded
Version 1.1alpha
?. = Learn Husky
-----

Husky << █
```

Figure 13: Launching the Husky interpreter

### 8.2.8 Husky interpreter

Husky is a variableless language based on lambda calculus and term rewriting systems. Husky is based on the version 1.1 of *Snake* [LS07a]. It was completely rewritten in CAML by Marthe Bonamy, ENSL (new parser, new syntactic constructions, like, *e.g.*, guards, anti-patterns, anti-expressions, exceptions and parametrized pattern matching). In Husky all the keywords of the language are ASCII-symbols. It could be useful to teach basic algorithms and pattern-matching to children.

## 8.3 Potential software

### 8.3.1 myMed (in french), see <http://www-sop.inria.fr/mymed>



**myMed : un réseau informatique transfrontalier pour l'échange de contenus dans un environnement fixe et mobile<sup>1</sup>.**

#### Objectifs

Le projet myMed est né d'une double constatation: l'existence d'un énorme potentiel de développement des activités économiques de la zone transfrontalière, objet de l'action Alcotra, et le manque criant d'infrastructures techniquement avancées en permettant un développement harmonieux. La proposition myMed est née d'une collaboration existante depuis plus de 15 ans entre l'Institut National de Recherche en Informatique et en Automatique (INRIA) de Sophia Antipolis et l'Ecole Polytechnique de Turin, auxquels viennent s'ajouter deux autres partenaires, l'Université de Turin et l'Université du Piémont Oriental.

Les objectifs de ce projet sont la conception, le prototypage et l'expérimentation d'un réseau social *mobile* et *géo-sensible* dans le but d'améliorer l'échange d'informations et la création de services personnalisés dans la zone transfrontalière Alcotra. Les réseaux sociaux, tels que Facebook, LinkedIn, mySpace, les mécanismes de partage et publication tels que YouTube et Wikipedia, sont des outils extrêmement puissants qui permettent le partage de connaissances ainsi que leur évolution et échange. myMed se propose de conjuguer les fonctionnalités offertes par ces derniers outils aux besoins et particularités de la zone transfrontalière Alcotra pour répondre aux besoins (non limitatifs) tels que :

- La création de services transfrontaliers à forte valeur ajoutée, rencontre entre l'offre et la demande des habitants et sociétés de notre région ;
- Le transfert efficace de technologie entre pôles de compétitivité et universités françaises et italiennes ;
- La diffusion d'une information fiable et *géo-sensible* auprès de groupes très exigeants ;
- La diminution, voire la suppression, dans la mesure du possible, de la fracture géographique due aux difficultés de se mouvoir dans l'inter-région ;
- L'apport d'instruments de soutien à la mobilité et à l'intégration, au travers de mesures de développement et de valorisation du caractère communautaire de la zone transfrontalière.

<sup>1</sup> Projetsoumis dans le cadre du projet Interrégional Alcotra (INTERREG IV A, ALCOTRA), contact Luigi Liguori INRIA Sophia Antipolis Méditerranée [Luigi.Liguori@sophia.inria.fr](mailto:Luigi.Liguori@sophia.inria.fr), +33 6 78 35 80 88).

## Résultats

myMed permettra (i) le transfert de technologies et l'accès à l'innovation aux entreprises utilisatrices de la zone et (ii) mettra à la disposition des populations résidentes toute une gamme de services personnalisés ainsi que des outils de soutien à la mobilité et à l'intégration.








Les services myMed seront accessibles par le biais de tout type de terminal informatique fixe ou mobile : ordinateur personnel, ordinateur portable, PDA, téléphone de dernière génération (3G)... sur lesquels sera installé le logiciel client *myMed client* développé durant le projet.

Le logiciel disposera d'une interface utilisateur graphique intuitive en permettant une rapide adoption auprès d'utilisateurs non avertis. Les services myMed s'appuieront sur une infrastructure informatique à haut degré de disponibilité, de fiabilité et d'évolutivité afin de permettre à des milliers d'utilisateurs de jouir de services de qualité, disponibles en tout temps. La gestion de ces services sera extrêmement simplifiée grâce aux techniques dites de *virtualisation* permettant le découplage entre le matériel et le logiciel avec comme conséquence une véritable résilience aux pannes. Finalement, ces techniques permettant une utilisation optimale des ressources matérielles auront un impact minimal sur notre environnement.

L'accès aux services myMed sera maximisé du fait de la couverture quasi totale offerte par les réseaux cellulaires, et ce, même en zone rurale, auxquels iront s'ajouter des points d'accès réseaux haut débit en zone urbaine (borne d'accès Wi-Fi, réseaux inter-véhiculaires...). Cette accessibilité permettra d'augmenter le nombre de services directs offerts aux utilisateurs, palliant ainsi aux limitations imposées par l'utilisation des seules technologies filaires.

## Exemples de scénario

Les scénarios d'utilisation du système myMed peuvent impacter tous les aspects de notre vie. La liste suivante est proposée afin d'illustrer quelques-uns des services qui seront offerts par myMed :

	<b>myTranslator</b> : Un service de traduction simultanée « non automatique » est offert par les usagers de myMed. L'utilisateur notifie le texte à traduire et les participants au réseau social, qui sont inscrits à ce service, reçoivent une requête de traduction. La confiance entre traducteurs assurera la qualité de la traduction.
	<b>myJob</b> : En utilisant le service myTranslator, les offres et les demandes de travail sont publiées et mises en correspondance dans leurs langues maternelles. [ex. « <i>Esperto in impianti di riscaldamento eco-compatibili cerca lavoro entro 60 km da Cuneo</i> » vs. « PME spécialisée en constructions biocompatibles recherche personnel qualifié »].
	<b>myMe</b> : Le service publie les coordonnées de l'utilisateur à tous ou partie de ses amis, qui pourront choisir de le suivre en temps réel. myMe est particulièrement utile pour les enfants ou les personnes âgées et pendant les excursions en montagne ou en mer. myMe fonctionne aussi en différé [ex. « Ton ami Pierre se trouve à Nice comme toi »].
	<b>myCarShare</b> : Service de covoiturage public/privé. Une solution envisagée – développée en collaboration avec la société Vulog, spécialiste de solutions innovantes de mobilité urbaine – consiste à permettre aux utilisateurs de louer une voiture électrique dans n'importe quel endroit de la ville, et ce, même en « last minute ».
	<b>myJam</b> : Un utilisateur informe le système de la présence d'un embouteillage. Le module GPS fournit les coordonnées au service myJam qui avertira automatiquement tous les utilisateurs ayant souscrits à ce service et se trouvant à proximité de la zone congestionnée. Le même service peut aussi être utilisé dans le cas d'un accident.
	<b>myMenu</b> : Un restaurant annonce aux passants et automobilistes dotés du service myMenu le menu du jour à un prix spécial. Cette information, reçue par l'utilisateur, sera relayée automatiquement et instantanément à tous les utilisateurs du service.
	<b>myLocalProducer</b> : Un service d'approvisionnement en filière courte (« kilomètres zéro ») va permettre aux petits exploitants et aux petites entreprises de proposer et vendre directement aux clients abonnés à ce service leurs produits bio et du terroir.

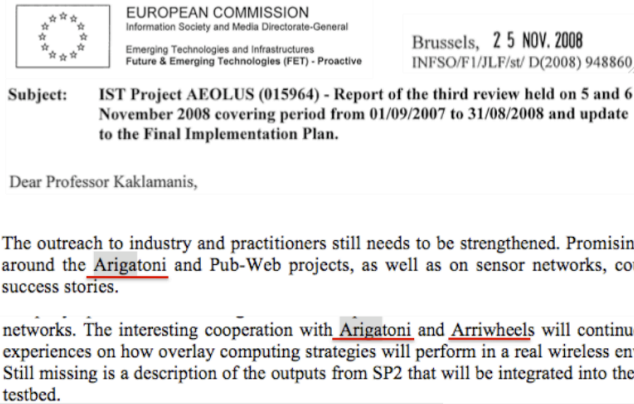


Figure 14: The European Commission point of view.

## 9 Contracts

### 9.1 INTERREG Alcotra: myMed, 2010-2013

The Interreg Alcotra office has founded the three-year project *myMed*: *un réseau informatique transfrontalier pour l'échange de contenus dans un environnement fixe et mobile*. LogNet will head the project; other partners are Vulog PME, GIR Maralpin, Politecnico di Torino, Uni. Torino, Uni. Piemonte Orientale. The total budget 1380Keur (796Keur for l'INRIA) - the external founding is 932Keur (526Keur for l'INRIA). The founders are UE, PACA, CG06, PREF06, and INRIA, see <http://www-sop.inria.fr/mymed>.

### 9.2 COLOR: JMED, 2010

*JMed: Formalizing Multiple Inheritance, Modules and Pluggable Type Systems on the Java Language Platform*. The goal of the Color action, in cooperation with the DISI Programming Languages group in Genova is the formal investigation and the prototype implementation of a Java extension, called “JMed”, featuring novel software composition mechanisms, expressive enough to subsume not only classical inheritance, but also more flexible mechanisms like mixin and trait inheritance. Moreover, such extension should be pursued without modifying the standard Java compilation and execution model, but, rather, by conceiving and formalizing a pre-compiler that processes the desired extensions and maps JMed into plain Java (that is, the implementation is directly driven by a flattening semantics).

### 9.3 FP6 FET Global Computing: IST AEOLUS, 2006-2010

*Algorithmic principles for building efficient overlay computers*, in collaboration with 21 European universities and coordinated by University of Patras, Greece. LogNet participate to the package 2 (Resource management) and to the package 5 (Extending global computing to wireless users). See also LogNet highlights.

Last year, the Arigatoni and the Ariwheels projects have been highlighted in the third year report of the IST Project AEOLUS covering period from 01/09/2007 to 31/08/2008 (Figure 14).

*On the exploitation side, already promising activities around Arigatoni and Pub-Web projects, [...], could lead to success stories ... The interesting cooperation with Arigatoni and Ariwheels will continue and give experiences on how overlay computing strategies will perform in a real wireless environment.*

For the second year, the Arigatoni and the CarPal application has been highlighted at the final AEOLUS review in February 2010 at Munich (the report is forthcoming).

## 9.4 JET TEMPUS DEUKS, 2007-2009

*Doctoral School Towards European Knowledge Society.* Main aim of this Project, in collaboration with 6 European universities, is to promote the current European landscape of doctoral programmes in Serbia. Particularly, the Project will develop and implement a pilot Doctoral Programme according to the European innovative recommendations with comprehensive approach to information technologies, where foundational theories are fully integrated in a pragmatic engineering approach. LogNet is the head of the French chapter.

## 10 Collaborations

We have woven over the last years a dense network of collaborations with national as well as international teams laboratories. To be short, this is our active (i.e. authors of papers or contracts) connection graph over the last two years.

**INRIA** Mascotte, Logical, Myriads.

**France** Department of Mathematics - University of Paris VII, Plume team - ENS Lyon.

**Italy** University of Udine, University of Torino, Politecnico of Torino, Politecnico of Bari, University of Modena, University of Genova, University of Piemonte Orientale, University of Insubria, University of Salerno.

**Europe** Universidad Politecnica de Valencia (Es), University of Sussex (Uk), University of Novi Sad (Sb), Mathematical Institute of the Serbian Academy of Sciences and Arts Belgrade (Sb)

**USA** Worcester Polytechnic Institute.

## 11 Self assessment

- The study of dynamic self-organizing programmable overlay computer is an hot topic that it is emphasized in the new INRIA strategic plan and current FP7.
- Until now, our main achievement was the design of a theory (semantics units, protocols, stochastic models, and preliminary dynamic graph theory) and pragmatics (simulation, implementation, and applications) of the **Arigatoni** and **Synapse** overlay network.
- The publication record w.r.t. the very few number of researchers involved in this activity is a good indicator that we are on the right track.
- We will take advantage of the (world renown) 20 years old experience of the LogNet researchers in the field of Semantics of Programming Languages, to successfully achieve the task of the semantic design and the efficient implementation of the **Ivonne** library.
- The Management should also take into consideration, in case of team creation, the possibility to hire a young promising researcher (CR2) skilled in design protocols for telecommunications.
- From the logistic point of view, we think essential to be “merged in the COM soup”, *i.e.* close to Planete, Maestro and Mascotte project-teams; we would take a great advantage to this proximity and we think we would add some value to those teams in the new field of (programmable) overlay networks. As such, we strongly believe that LogNet should be located in the Lagrange building. We know that actually this seems not possible, but we should be kept in mind by the management.

## 11.1 Trivia

- *Why Arigatoni?* “Arigatou” in Japanese means (informally) “thank-you”, while “Rigatoni” are one of the most commonly used pasta in Southern and Central Italy. Rigatoni, a wide, ridged, tube-shaped pasta, have holes large enough to capture pieces of meat or vegetables in sauces. They have ridges which allow them to hold more sauce. In *freetalian* network-jargon, when you align some rigatoni, then you make an (high-speed) network connection that allows two or more units to communicate in a point-to-point fashion.
- *Do you know that?* The very first discussions and papers about Arigatoni [BCLV06b, BCLV06a] where made in 2006 with the precious help of Didier Benza and Marc Vesin, working with the INRIA Sophia *SEMIR: Service d'Exploitation et de Maintenance Informatiques et Réseau*, Team *RESET: Réseaux et Télécom*. At that time, Michel Cosnard was the DCR of INRIA Sophia and Luigi Liquori just joined the Mascotte Project Team. This experience of mixing researchers with engineers was very fruitful and is often taken, by the INRIA Management, as “the good example” to follow to exploit potential synergies of our institute.
- *Cigarettes and coffee*<sup>1</sup>. . . This document is issue of one year of hard work, doped by strong *italian-café* discussions of the LogNet team. The scientific statement above is the outcome.

## 11.2 Conclusions

The LogNet research statement envisions a new type of computational models and devices: from physical, proprietary, personal, more or less expensive, programmable, Turing Complete computers to ethereal, virtual, public, impersonal, ubiquitous, inexpensive, programmable, Turing Complete overlay network computers. This looks a bit like closed-*vs.*-open social model *querelle*, but we can also pay to be logged inside a rich colony. We conjecture, in the next decade, that at least 2 categories of overlay network computer systems can emerge:

- Not-free, closed overlay networks, made by semantically powerful overlay computers offering services in change of money, and
- Free, open overlay networks, made by semantically powerful programmable overlay network computers offering services for free.

Finally, we think our vision of a programmable overlay network computer be a new exciting research vein that we would be proud to pursue at INRIA Sophia Antipolis Méditerranée. □

## References

- [BCCL08] D. Borsetti, C. Casetti, C.-F. Chiasserini, and L. Liquori. Content Discovery in Heterogeneous Mobile Networks. In E. Hossain, editor, *Heterogeneous Wireless Access Networks: Architectures and Protocols*, pages 419–441. Springer-Verlag, 2008.
- [BCLV06a] 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.
- [BCLV06b] D. Benza, M. Cosnard, L. Liquori, and M. Vesin. Arigatoni: Overlaying Internet via Low Level Network Protocols. Technical Report 5805, INRIA, 2006.

---

<sup>1</sup>Smoking is dangerous to your health.

- [Bit] BitTorrent, Inc. <http://www.bittorrent.com/>.
- [CAG08] V. Chandrasekhar, J. Andrews, and A. Gatherer. Femtocell networks: a survey. *IEEE Communications Magazine*, 46(9):59–67, 2008.
- [CCFM08] C. Casetti, C.F. Chiasserini, R. Fracchia, and M. Meo. Autonomic Interface Selection for Mobile Wireless Users. *IEEE Transactions on vehicular technology*, 56:3666–3678, 2008.
- [CCL06] R. Chand, M. Cosnard, and L. Liquori. Resource discovery in the Arigatoni overlay network. In *I2CS, International Workshop on Innovative Internet Community Systems*, Lecture Notes in Computer Science. Springer Verlag, 2006.
- [CCL08] R. Chand, M. Cosnard, and L. Liquori. Powerful resource discovery for Arigatoni overlay network. *Future Generation Computer Systems*, 1(21):31–38, 2008.
- [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*, volume 4415 of *Lecture Notes in Computer Science*, 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.
- [FHFB07] M. Fiore, J. Härri, F. Filali, and C. Bonnet. Vehicular mobility simulation for vanets. In *Annual Simulation Symposium*, pages 301–309, 2007.
- [HLLS08] F. Honsell, M. Lenisa, L. Liquori, and I. Scagnetto. A conditional logical framework. In *LPAR, Logic for Programming, Artificial Intelligence, and Reasoning*, Lecture Notes in Computer Science, pages 143–157. Springer-Verlag, 2008.
- [IBM] IBM. Business Process Execution Language. <http://www.ibm.com/developerworks/library/specification/ws-bpel/>.
- [LBCC08] L. Liquori, D. Borsetti, C. Casetti, and C. Chiasserini. An Overlay Architecture for Vehicular Networks. In *IFIP Networking, International Conference on Networking*, volume 4982 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 2008.
- [LC07a] L. Liquori and M. Cosnard. Logical Networks: Towards Foundations for Programmable Overlay Networks and Overlay Computing Systems. In *TGC, Trustworthy Global Computing*, volume 4912 of *Lecture Notes in Computer Science*, pages 90–107. Springer-Verlag, 2007.
- [LC07b] 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.
- [LS07a] L. Liquori and B. Serpette. iRho: An Imperative Rewriting-calculus. *MSCS, Mathematical Structures in Computer Science*, 18(3), 2007.
- [LS07b] L. Liquori and A. Spiwack. FeatherTrait: A Modest Extension of Featherweight Java. *TOPLAS, ACM Transaction on Programming Languages and Systems*, 30(2), 2007.
- [LS08] L. Liquori and A. Spiwack. Extending FeatherTrait Java with Interfaces. *TCS, Theoretical Computer Science*, 398(1-3):243–260, 2008. Calculi, types and applications: Essays in honour of M. Coppo, M. Dezani-Ciancaglini and S. Ronchi Della Rocca.

- [LTB09] L. Liquori, C. Tedeschi, and F. Bongiovanni. Babelchord: a social tower of dht-based overlay networks. In *ISCC*, pages 307–312, 2009.
- [LTV<sup>+</sup>10] L. Liquori, C. Tedeschi, L. Vanni, F. Bongiovanni, V. Ciancaglini, and B. Marinkovic. Synapse: A Scalable Protocol for Interconnecting Heterogeneous Overlay Networks. In *IFIP Networking, International Conference on Networking*, Lecture Notes in Computer Science. Springer-Verlag, 2010. To appear.
- [MC82] J. Misra and K. M. Chandy. A Distributed Graph Algorithm: Knot Detection. *ACM TOPLAS*, 4(4):678–686, 1982.
- [NCL07] P. Nain, C. Casetti, and L. Liquori. A Stochastic Model of an Arigatoni Overlay Computer. Research Report to be given, Politecnico di Torino, 2007.
- [Rap63] A. Rapoport. *Mathematical models of social interaction*, 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.
- [SMK<sup>+</sup>01] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup service for Internet Applications. In *ACM SIGCOMM*, pages 149–160, 2001.
- [TCFC<sup>+</sup>10] O. Trullols-Cruces, M. Fiore, C. Casetti, C.F. Chiasserini, and J.M. Barcelo-Ordinas. Planning roadside infrastructure for information dissemination in intelligent transportation systems. *Elsevier Computer Communications*, 33(4):432–442, 2010.
- [vdAtH05] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: yet another workflow language. *Information System*, 30(4):245–275, 2005.
- [vN88] J. von Neumann. The Principles of Large-Scale Computing Machines. *IEEE Ann. Hist. Comput.*, 10(4):243–256, 1988.
- [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.