



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Team LogNet

*Logical Networks: Self-organizing Overlay
Networks and Generic Overlay Computing
Systems*

Sophia Antipolis - Méditerranée

Theme : Distributed Systems and Services

Activity
R *eport*

2009

Table of contents

1. Team	1
2. Overall Objectives	1
2.1. LogNet's Motto and Logo	1
2.2. Overall objectives	1
2.3. Highlights of the year	2
3. Scientific Foundations	3
3.1. Lognet's general context	3
3.2. General definitions	4
3.3. Background: Arigatoni overlay network computer	5
3.3.1. Arigatoni units	5
3.3.2. Virtual organizations	7
3.3.3. Resource discovery protocol (RDP)	7
3.3.4. Virtual Intermittent Protocol (VIP)	8
3.3.5. Two simple examples	9
3.4. General research directions	10
3.4.1. On Virtual organizations	10
3.4.2. On Resource discovery	10
3.4.3. Execution model	12
4. Application Domains	13
4.1. Panorama	13
4.2. Potential applications	14
5. Software	15
5.1. Ariwheels	15
5.2. Arigatoni simulator	16
5.3. BabelChord simulator	16
5.4. Synapse simulator	17
5.5. Synapse client	17
5.6. Open Synapse client	17
5.7. Husky interpreter	18
5.8. myTransport Gui	19
5.9. "Perinet" site of the GIR Maralpin	19
6. New Results	19
6.1. Babelchord, a DHT's tower	19
6.2. Synapse, interconnecting heterogeneous overlay networks	21
6.3. DHT formal specification	22
6.4. Resource discovery and Self-stabilizing in Tree-bases P2P systems	23
6.5. Intersection and Union Types à la Church	23
6.6. Dealing with uncertain knowledge in Logical Frameworks	24
7. Other Grants and Activities	24
7.1.1. Interreg Alcotra: myMed, 2010-2012	24
7.1.2. FP6 FET Global Computing: IST AEOLUS, 2005-2009	24
7.1.3. FP6 TEMPUS DEUKS, 2007-2009	25
8. Dissemination	25
8.1. Participation in committees and referees	25
8.2. Teaching and Meeting organizations	25
8.3. Spare presentations	25
8.4. Visitors	26
9. Bibliography	26

LogNet is an INRIA team.

1. Team

Research Scientist

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

External Collaborator

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

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

Michel Cosnard [CEO INRIA, HdR]

Technical Staff

Laurent Vanni [from September 1st 2009 to December 31th 2009]

PhD Student

Francesco Bongiovanni [INRIA-PACA grant, from October 1st 2008 to October 1st 2009]

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]

Bojan Marinkovic [INRIA-TEMPUS grant, MISANU, Serbia, from February 2009 to May 2009]

Post-Doctoral Fellow

Cédric Tedeschi [INRIA grant, from October 1st 2008 to September 1st 2009]

Administrative Assistant

Nathalie Bellesso [INRIA]

Other

Marthe Bonamy [ENSL, from June 1st 2009 to July 15th 2009]

Luc Marongiu [IUT, from April 10th 2009 to June 12th 2009]

Alexis Paoleschi [IUT, from April 10th 2009 to June 12th 2009]

2. Overall Objectives

2.1. LogNet's Motto and Logo

Our Motto is “*Computer is moving on the edge of the Network...*” by Jan Bosch, Nokia Labs, [LNCS 4415, 2007] and our logo is in Figure 1.

2.2. Overall objectives

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*, 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 into 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 into 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 firstly inside its colony, and secondly 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 of (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,

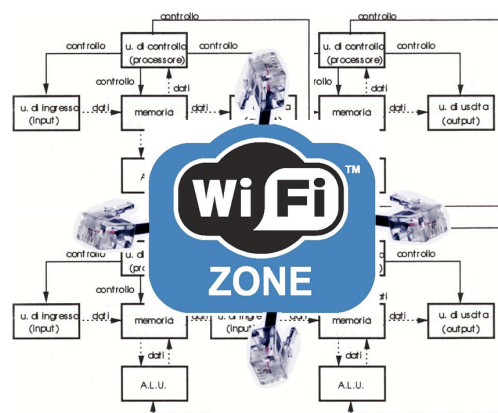
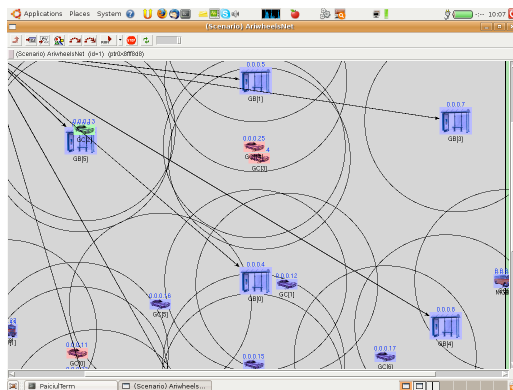


Figure 1. Our logo

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 a generic overlay network computer system.

2.3. Highlights of the year



EUROPEAN COMMISSION
Information Society and Media Directorate-General
Emerging Technologies and Infrastructures
Future & Emerging Technologies (FET) - Proactive

Brussels, 25 NOV. 2008
INFO/F1/JLF/su D(2008) 948860

Subject: IST Project AEOLUS (015964) - Report of the third review held on 5 and 6 November 2008 covering period from 01/09/2007 to 31/08/2008 and update to the Final Implementation Plan.

Dear Professor Kaklamani,

The outreach to industry and practitioners still needs to be strengthened. Promising activities around the Arigatoni and Pub-Web projects, as well as on sensor networks, could lead to success stories.

The interesting cooperation with Arigatoni and Arriwheels will continue and give experiences on how overlay computing strategies will perform in a real wireless environment. Still missing is a description of the outputs from SP2 that will be integrated into the AEOLUS testbed.

Figure 2. The Ariwheels simulator and the European Commission point of view

- 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 [7], [2]. The project ended successfully with a demo in front of Piemonte authorities and members of the press, see http://www.lastampa.it/_web/cmstp/tmplrubriche/tecnologia/grubrica.asp?ID_blog=30&ID_articolo=6876&ID_sezione=38&sezione=News. The *Arigatoni* and the *Ariwheels* projects have been highlighted in the third year report of the IST Project AEOLUS, covering



Figure 3. The myMed social network

the period from 01/09/2007 to 31/08/2008

- Another outcome of the *Arigatoni* research conducted in the team is the founding by the Interreg Alcotra office of 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 funding is 932Keur (526Keur for l'INRIA). The founders are UE, PACA, CG06, PREF06, and INRIA, see <http://www-sop.inria.fr/mymed>.

3. Scientific Foundations

3.1. Lognet's general context

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 which make use of aggregated computational power, storage, information resources, etc. We would like to start our first activity report with the standard definition of a *Computer System*.

Definition 1 (Computer System)

A computer system consists of computer hardware and computer software.

- Computer Hardware is the physical part of a computer, including the digital circuitry, as distinguished from the computer software that is executed within the hardware. The hardware of a computer is infrequently changed, in comparison with software and data.
- Computer Software consists of three parts, namely: system software, program software, and application software.
 - System Software helps run the computer hardware and the computer system. Examples are operating systems (OS), device drivers, diagnostic tools, servers, windowing systems...
 - Program Software usually provides tools to assist the programmer in writing computer programs and software using different programming languages. Examples are text editors, compilers, interpreters, linkers, debuggers for general purpose languages...
 - Application Software allows end-users to accomplish one or more specific (non computer-related) tasks, pertaining to fields such as 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 Network Computer System* is. The reader can focus on the tiny, yet crucial differences.

Definition 2 (Overlay Computer System)

An overlay computer system consists of overlay computer hardware and overlay computer software.

- Overlay Computer Hardware is the physical part of an overlay computer, including the digital circuitry, as distinguished from overlay computer software that is executed 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.
- Overlay Computer Software consists of three parts, namely: overlay system software, overlay program software, and overlay application software.
 - Overlay System Software helps run the overlay computer hardware and the overlay computer system. Examples are network middleware playing as a distributed operating system (dOS), resource discovery protocols, virtual intermittent protocols, security protocols, reputation protocols...
 - 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.
 - Overlay Application Software allows end-users to accomplish one or more specific (non-computer related) tasks, pertaining to fields such as industrial automation, business software, educational software, medical software, databases, and computer games...These 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 objectives can be summarized as follows:

- to provide adequate notions and definitions of a generic overlay network computer; from a desktop distributed calculator to a programmable distributed overlay computer;
- on the basis of these definitions, to propose a precise architecture of a generic overlay network computer and implement it;
- on the basis of these definitions, to implement an overlay software factory suitable to help the logical and software assembling of an overlay network computer.

3.2. General definitions

An overlay network is a computer network which is built on top of another 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 a virtual organization.

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 fashion (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 leaders. Agents in the overlay are regrouped in *Colonies*. A colony is a simple virtual organization consists of 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 he manages. Thus, agents are simple computers (think of them as *amoebas*), or sub-colonies (think of them as *protozoas*). 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 of the agents in its colony. One of the tasks of the leader is to manage (un)subscriptions to its colony.

Resource discovery. By adhering to a colony, an agent can expose resources he has and/or ask for resources it requires. 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, he makes a request to its leader. A leader is devoted to contacting and negotiating with potential servers, to authenticating clients and servers, and to routing requests. The rationale ensuring scalability is that every request is handled firstly 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 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.

3.3. Background: Arigatoni overlay network computer

As suggested by our previous definitions, we are mainly concerned by three topics: network organization, resource discovery and orchestration. These topics are studied in a complementary way by *Arigatoni* (work started by Luigi Liquori and Michel Cosnard). In this section we will describe the current status of *Arigatoni*.

The *Arigatoni* overlay network computer, [1], [3], [9], [8], [5], [6], [4] 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 into 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 into an outermost colony, which is 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 firstly inside its colony, and secondly outside, via the proper super-leader (thus applying an *endogenous-first-exogenous-last* strategy).

Theoretically, queries are formulæ in first-order logic. When the client agent receives notification of all of (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' churn 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* 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, such as fire, a terrorist attack, an earthquake, etc.

3.3.1. Arigatoni units

In what follows, we briefly introduce the logic units 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 consisting of a small *RAM-ROM-HD* memory capacity, a modest *CPU*, a ≤ 40 keystrokes keyboard (or touchscreen), a tiny screen (≤ 4 inch), an *IP* or *ad hoc* connection (via *DHCP*, *BLUETOOTH*, *WIFI*, *WIMAX*...), 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 terminal *Nokia N810*. Of course, a *AC* can be a supercomputer, or an high performance *PC*-cluster, a large database server, a 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 within the *AC* is not important. The computer should be able to work in *local mode* for all of 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 in a "connected area"; this can be done using the underlay network and related technologies;
- Register on one or many *ABs*, thus *de facto* entering 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 *ACs* come with their proper *PKI* certificate.

Agent Broker (AB). This unit can be, *e.g.*, a computer device made up of a high speed *CPU*, an *IP* or *ad hoc* connection (via *DHCP*, *BLUETOOTH*, *WIFI*, *WIMAX*...), 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 a *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 with 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 for servicing 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) with another super-colony;
- Perform a combination of the last two actions mentioned above;
- 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, the *AC* client of 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 *AC*s 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*s that it knows or discovers;
- Checks the well-formedness and forwards packets of the two *Arigatoni*'s protocols across the overlay toward their destinations.

3.3.2. Virtual organizations

Agent computers communicate by first registering with 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*, or *JOpera*.

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 [22] 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.

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 to/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.

3.3.3. 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 service requests can be also asked of an *AB*; each service is processed sequentially and independently of the 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 *AC*s.

The *RDP* protocol proceeds as follows: suppose an *AC* X registers – using the intermittent protocol *VIP* presented below – with 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 has 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.

3.3.4. Virtual Intermittent Protocol (VIP)

There are essentially two ways in which an *AC* can register to an *AB* (sensible to its physical position in the network topology), the latter not being enforced by the *Arigatoni* model (see [6]):

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 of or requested from 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 disappears;
- Once an *AC* has been disconnected from a colony belonging to any administrative domain, it can physically migrate into 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 ϵ), he 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 meaning that the free rider promised some services but finally did not provide any service at all (untrustworthiness).

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 allows for reasoning on the colony’s load-balancing and kind of colonies. Possible politics and are:

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

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

3.3.5. Two simple examples

To give an idea of the 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 [1], [7], [2].

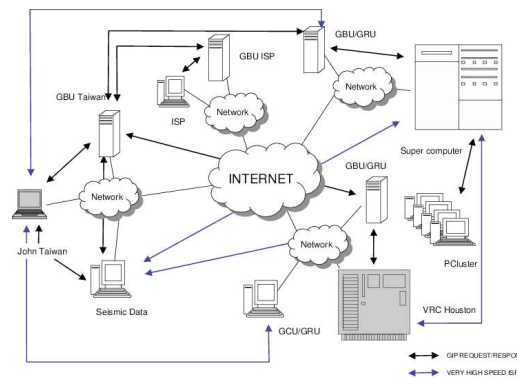


Figure 4. *Arigatoni* Overlay Network for a Grid Seismic Monitoring Application

GRID: scenario for seismic monitoring. John, chief engineer of the SeismicDataCorp Company, Taiwan, on board of the seismic data collector ship, has to decide on the next data collecting campaign. For this he would like to process and analyze 100 TeraBytes of seismic data that have been recorded on the data mass recorder located in the offshore data repository of the company. 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 processing; however, the amount of computation is so big that a supercomputer and a cluster of PCs have to be *rented* by the SeismicDataCorp company. John will ask also for *bandwidth* in order to get rid of any bottlenecks 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. With this in mind:

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

This scenario is pictorially presented in Figure 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 all resources of their colonies *de facto* common. 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.

3.4. General research directions

Following our main three topics, network organization, resource discovery and orchestration, for middle and long term research, we envisage the following studies.

3.4.1. On Virtual organizations

- **Trees vs. graphs: a conflict without a cause.** In the first versions of *Arigatoni*, the network topology was tree- or forest-based. But since agents are not slaves, multiple registrations are in principle possible and unavoidable. This weaves the network topology into a *dynamic graph* [20], where nodes do not have a complete knowledge of the topology itself. As an immediate consequence, our protocols must 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 [9].

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, as in the “frame-relay” world. For the record, overbooking in telecommunications means that a telephone company has sold access to too many customers who 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.

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 the design of new routing algorithms and protocols stable to agent churn; this scenario can be modeled using dynamic graph theory.

- **Fault tolerance.** The virtual organization 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.

3.4.2. On Resource discovery

- **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, such as fire, a terrorist attack, an 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:

- Request for computational power (*i.e.* the *Grid*);
 - Request for memory space (*i.e.* distributed storage);
 - Request for bandwidth (*i.e.* *VoIP*);
 - Request for a distributed file retrieving (*i.e.* standard *P2P* applications);
 - Request 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)*;
 - *Request 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)*;
 - *Request for a human computer interaction (the human playing the role of an agent)...*
- **Social model underneath an overlay network computer.** The *Arigatoni* overlay network computer defines mechanisms for devices to inter-operate, by offering services, in a way that is reminiscent to Rapoport's *tit-for-tat* strategy of co-operation based on reciprocity. This way to understand common behavior of virtual organizations has some theoretical basis on Game Theory. 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* [19].
- This means that *Arigatoni* 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 himself as belonging “much” to the colony *foo* and “a little bit” to the colony *bar*). This denotes a decentralized non-exclusive policy model. As such, one question can arise: who is *Arigatoni* designed for? We believe the overlay is flexible enough to serve a mix of different “social structures” and “end-users”:
- Independent end-user connecting through his *ISP* or migrating from hot-spot to hot-spot;
 - Cooperative communities of disseminated agents;
 - More regulated or hierarchical communities (maybe a better view of a corporate network);
 - Cooperative or competitive resource providers and resource brokers.
- **Quality metrics underneath an overlay network computer.** The *Arigatoni* overlay network computer is suitable to support various extended trust models. Moreover, the reputation score could be expanded to a multi-dimensional value, for example, by 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 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:

- 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 servants, including the 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 [1], for example to transmit location and parameter information to call a third party system.

3.4.3. Execution model

- **Programming an overlay network computer.** 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 in a new language (*à la BPEL, LINDA, YAWL, JOpera...*), let's call it *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... [24].

The “main” of an *Ivonne* 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.

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*.

We also envisage to design a *run-time* distributed virtual machine, built on top of a virtual or hardware machine, in order to scale-up from local to distributed computations and to fit with the distributed nature of an overlay network computer. Communication between agent computers will be performed through a *logic bus*, using Web technologies, like *SOAP* or *AJAX* protocols, or a combination of *Java*-based *JNI+RMI*-protocols, or *.NET, XPCOM, D-BUS, OLE* bus protocols, or even by enriching the *Arigatoni* protocol suite with an *ad hoc* control-flow and data-flow protocol, and permitting to use it directly inside *Ivonne*.

The *Ivonne* language can be both interpreted and compiled. In the latter case we envisage the design of an intermediate low-level distributed assembler language in which *Ivonne* could be compiled. The intermediate machine code will recast the assembler pseudo code

```
move R0 R1
```

à la Backus [18] in

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

where, of course, latency is a non-trivial issue, or the assembler pseudo code

```
op R0 R1 R2 in
```

```
op on ipR0 with ipR0:portR0:dataR0 and ipR1:portR1:dataR1 and
```

```
stockin ipR2:portR2:dataR2.
```

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* program.

- + **Trust and security.** 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 to their distinctive roles. The whole mechanisms involved by a *PKI* are out of the scope of this research statement, but good use of *PKIs* and an implementation compliant with *RFC2743* 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. 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 he knows;
 - Each agent maintains a reputation score for each resource he serves;
 - Exchanges between agents update each other’s scores dynamically;
 - Conflicts between two or many agents are resolved by the broker leaders of the colonies to which the 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:

- *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* could be used to overcome such obstacles.

4. Application Domains

4.1. Panorama

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

- New distributed models of computation
- Overlay networks over mobile *ad hoc* networks
- Reduce the digital divide

4.2. Potential applications

From large-scale computing machines to large-scale overlay network machines (John von Neumann was right after 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 [25]. 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 joints, 30 short tons, 2.4m x 0.9m x 30m, stored in a 167 m² 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!

Developing a pedestrian/vehicular infrastructure based on an overlay network computer. 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*). See, for more details, the *Arigatoni*’s sub-project *Ariwheels*.

Programming services for the new mesh overlay network in the Campus STIC of 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 an underlay network are to offer *IP* connection to all of the 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 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 a unique opportunity to have a real testbed into which we could put our programmable overlay to the test. *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.

Reducing the Digital Divide [Sources Wikipedia]. 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: at first, it meant the ownership of a computer. Later, it meant 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 the digital divide are “socioeconomic status, with income, educational level, and race among other factors associated with technological attainment”.

One area of 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 have 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. Although 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.

Recently, discussions regarding the 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 inhabitants, will be equipped with public local *WIFI LAN* (e.g. see the declaration of the Major, in Italian, <http://it.youtube.com/watch?v=zBTnkEnXTlc>). Our vision could contribute to reducing the digital divide in our society, and, more contextually, in the future Campus STIC.

5. Software

5.1. Ariwheels

Participants: Luigi Liquori [contact for the *Ariwheels* simulator], Claudio Casetti [Politecnico di Torino, Italy], Diego Borsetti [Politecnico di Torino, Italy], Carla-Fabiana Chiasserini [Politecnico di Torino, Italy], Diego Malandrino [Politecnico di Torino, Italy, contact for the *Ariwheels* client].

Ariwheels is an infomobility solution for urban environments, with access points deployed at both bus stops (forming thus a wired backbone) and inside the 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:

- a protocol, based on *Arigatoni* and the publish/subscribe paradigm;
- a set of applications, implementing the protocol on the different types of nodes;
- a simulator, written in OMNET++ and recently ported to the ns2 simulator.

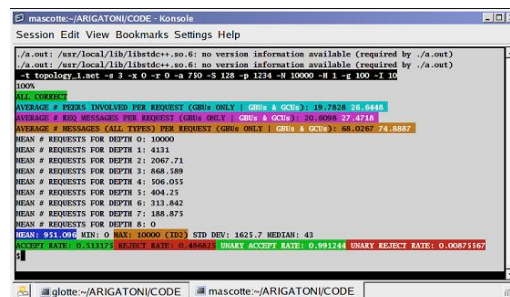
We implemented *Ariwheels* within the Omnet++ simulator, coding the overlay part and exploiting the existing wireless underlay network modules. In the underlay, we used IEEE 802.11 at the MAC layer and the DYMO routing protocol (an AODV-like reactive routing protocol). We tested the performance of *Ariwheels* in a vehicular environment. We used a realistic mobility model generated by the simulator VanetMobiSim, whose output (mobility traces) was fed to the Omnet++ simulator. Vehicles travel in a 1 km² city section over a set of urban roads, which include several road intersections regulated by traffic lights or stop signs. In particular, we adopted the IDM-IM microscopic car-following model [21], which allows us to reproduce real-world traffic dynamics as queues of vehicles decelerating and/or coming to a full stop near crowded intersections.

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. *Ariwheels* is based on the 802.11 wireless LAN protocols. Therefore, its infrastructure is mostly made of access points, deployed at bus stops, forming a backbone; and on the buses themselves, thus with an intermittent connection to the backbone. *Ariwheels* have four kinds of functional units, namely agents, brokers, mobile brokers, and proxies. The agent is a software 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 and the request of the resources the agent needs. The broker is a program, 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: advertise its presence and the resources available through it; receive, elaborate and acknowledge the registration requests coming from the Agents; receive and (try to) answer the resource request coming from the Agents; manage feedback and reputation. 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. 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.

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

5.2. Arigatoni simulator

Participants: Luigi Liquori [contact], Raphael Chand [Université de Geneva, Switzerland].



```

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)
./a.out topology: 1.net = 3 -x 0 +r 0 -a 750 -s 128 -p 1234 -N 10000 -M 1 -g 100 -I 10

100%
ALL COMPLETE
AVERAGE # PEERS INVOLVED PER REQUEST (GRU ONLY || GRU & GCU): 19.7828 16.6648
AVERAGE # MSGS PER REQUEST (GRU ONLY || GRU & GCU): 26.9098 22.6218
AVERAGE # MESSAGES (ALL TYPES) PER REQUEST (GRU ONLY || GRU & GCU): 66.0267 74.8887
MEAN # REQUESTS FOR DEPTH 0: 10000
MEAN # REQUESTS FOR DEPTH 1: 4121
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: 188.875
MEAN # REQUESTS FOR DEPTH 8: 0
MEAN: 951.094 MIN: 0 MAX: 10000 (TID) STD DEV: 1625.7 MEDIAN: 43
REQUEST RATE: 0.999275 REJECT RATE: 0.000725 UNARY ACCEPT RATE: 0.991244 UNARY REJECT RATE: 0.008756
  
```

Figure 5. The 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.

5.3. BabelChord simulator

Participant: Cédric Tedeschi [contact].

We have conducted some simulations of the BabelChord protocol [14]. 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 $\log((N \times C)/F)/2$.

The second time, the simulator computes the number of hops required to reach one of the nodes storing one of the keys of a particular resource. Results are given for different values of N , F , and C . Simulations show 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.

5.4. Synapse simulator

Participant: Cédric Tedeschi [contact].

To better capture its relevance, we have conducted some intensive simulations of the Synapse approach [17]. 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 see <http://www-sop.inria.fr/lognet/synapse/pysynapse/pysynapse.zip>.

5.5. Synapse client

Participants: Laurent Vanni [contact], Luigi Liquori, Cédric Tedeschi, Vincenzo Ciancaglini.

In order to test our Synapse protocol [17] 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>.

5.6. Open Synapse client

Participant: Bojan Marinkovic [contact].

Husky is a variableless language based on lambda calculus and term rewriting systems. Husky is based on the version 1.1 of *Snake* [10]. 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 childrens.

5.8. myTransport Gui

Participants: Laurent Vanni [contact], Vincenzo Ciancaglini.

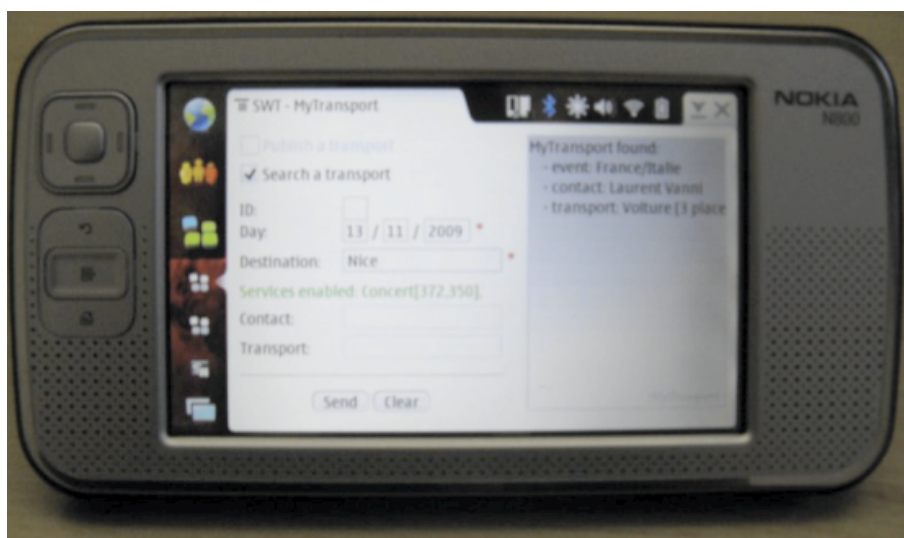


Figure 7. myTransport on the Nokia N800 Internet tablet

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 7. The GUI is written in Java and it is fully functional in the Nokia N800 internet tablet devices.

5.9. “Perinet” site of the GIR Maralpin

Participants: Luc Marongiu [contact], Alexis Paoleschi.

We design completely, using the Content Management Systems JOOMLA, the “Perinet” site of the GIR Maralpin *Groupe Interdisciplinaire de Réflexion sur les traversées sud alpines et l’aménagement du territoire Maralpin*. Hundreds of pages of this site are currently visited.

6. New Results

6.1. Babelchord, a DHT’s tower

Participants: Luigi Liquori, Cédric Tedeschi, Francesco Bongiovanni.

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. The 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 [23] 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 route 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 use a unique hash function, which is 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 distributed finger tables need to stabilize. Both of the rings need to know their relative hash functions and have to decide which ring will absorb the other, the latter point being critical because of the politics and security reliance. We propose to connect smaller Chord networks in an unstructured way via special nodes playing the role of *neural synapses*.

Schematically, the main features of Babelchord 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 for employment 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 for a high number of successful lookups. Following a personal “good deal” strategy, a peer can decide to invite a 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 of as a sort of *meta overlay network* or *meta-DHT*, where inter-floor 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.

Because of the aforementioned original features, the following are examples of applications for which Babelchord can provide 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 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 and then billions of neurons - even at the biological model level. Babelchord could help modeling as a meta-overlay network for the human brain.

6.2. Synapse, interconnecting heterogeneous overlay networks

Participants: Luigi Liquori, Cédric Tedeschi, Laurent Vanni, Francesco Bongiovanni, Vincenzo Ciancaglini, Bojan Marinkovic.

We investigate Synapse [17], 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.

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 interconnected. 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 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, than 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: (i) several logical links for one node leads to as many alternative physical routes through these overlay, and (ii) 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.

6.3. DHT formal specification

Participants: Bernard Serpette, Cédric Tedeschi.

We have begun a formal specification associated to a *PiNet* implementation, of a *DHT* (Distributed Hash Table) oriented network. The specification is done via a relation on network states (small step semantics). A network state is the set of all individuals network nodes states. A node state consists of a, possibly sorted, set of neighbors, *i.e.* the nodes known by a specific node, a local memory and a queue of messages received by the node to be executed. The semantics describe the behavior of incoming messages in each nodes. The specificity of the formalized *DHT* are:

(i) as in Chord, the path found between any two nodes of the network has a length which is in $\mathcal{O}(\text{Log}(n))$, where n is the number of nodes of the network; (ii) the graph induced by the neighbors is symmetric, *i.e.* if a node a can communicate with a node b , then b can communicate with a . This fact is generally ensured by the transport layer (*TCP*, *UDP*...) and thus the associated improvement (path lengths are divided by two) comes at a low price; (iii) the dynamic nature of such network, *i.e.* the fact that nodes can join and leave the network, requires to readjust the neighbors of some nodes. This readjustment is called stabilization. In contrast to Chord, where stabilization is done via a periodic background process which is hard to tune, our stabilization is done during the routing of messages and thus 1) does not make any administrative charge when the network is sleeping 2) uses only few words in already existing packets and so has a very limited impact to the average network latency.

Following a formal general specification of such networks, we have developed a java simulator, coupled to a real implementation intended to catch the exact precise behavior of several topologies used in P2P systems.

6.4. Resource discovery and Self-stabilizing in Tree-bases P2P systems

Participants: Cédric Tedeschi, Eddy Caron [EPI GRAAL INRIA], Frédéric Desprez [EPI GRAAL INRIA], Franck Petit [EPI GRAAL INRIA].

The Distributed Lexicographic Placement (DLP)-Table is a *P2P* approach for the service discovery within large scale grids. It relies on a prefix tree structured overlay network. It provides load balancing, efficient mapping of nodes of the tree onto processors of the network and fault-tolerance mechanisms, formally proved to be self-stabilizing, *i.e.* converging to a correct topology in a finite time starting from an arbitrary topology and memory state. It has been initially developed within the INRIA GRAAL project team.

In collaboration with Eddy Caron, Frédéric Desprez and Franck Petit of GRAAL, we have written a chapter to appear in the future book entitled “Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications” and published by IGI Global [15]. This chapter gives a more *popularizing* view of the system and its features.

A journal paper about the stabilizing part of this architecture has been accepted for publication in Parallel Processing Letters [13].

6.5. Intersection and Union Types à la Church

Participants: Luigi Liquori, Dan Dougherty.

We studied an explicitly typed lambda calculus “à la Church” based on the union and intersection types discipline; this system is the counterpart of the standard type assignment calculus “à la Curry.” Our typed calculus enjoys Subject Reduction and confluence, and typed terms are strongly normalizing when the universal type is omitted. Moreover, both type checking and type reconstruction are decidable. In contrast to other typed calculi, a system with union types will fail to be “coherent” in the sense of Tannen, Coquand, Gunter, and Scedrov: different proofs of the same typing judgement will not necessarily have the same meaning. In response, we introduce a decidable notion of equality on type-assignment derivations inspired by the equational theory of bicartesian-closed categories [16].

We address the problem of designing a λ -calculus *à la* Church corresponding to Curry-style type assignment to an untyped λ -calculus with intersection and union types. In particular, we define a typed language such that its relationship with the intersection-union type assignment system fulfills the following *desiderata*: (i) typed and type assignment derivations are *isomorphic*, *i.e.*, the application of an *erasing function* on all typed terms and contexts (in a typed derivation judgment) produces a derivable type assignment derivation with the same structure, and every type assignment derivation is obtained from a typed one with the same structure by applying the same erasure; (ii) type checking and type reconstruction are decidable; (iii) reduction on typed terms has the same fundamental nice properties of reduction on terms receiving a type in the type-assignment system, namely confluence, preservation of typing under reduction, and strong normalization of terms typable without the universal type ω .

6.6. Dealing with uncertain knowledge in Logical Frameworks

Participants: Petar Maksimovic, Luigi Liquori.

The main goal of this research vein is the design and prototype implementation of logical frameworks in which it would be possible to smoothly encode various probabilistic logics, and other logics considered in general to be exotic, such as Modal, Program, Linear or Relevance logics. It may also involve the formalization and possible creation of a certified SAT checker for one or more probabilistic logics within the proof assistant Coq.

The theoretical background behind this involves:

- typed lambda calculi, namely the cube of Barendregt,
- the Edinburgh Logical Framework, and
- the proof assistant Coq.

At this point, formalization of one of the probabilistic logics, namely LPP_2 , in the proof assistant Coq, is well under way. In this probabilistic logic, one can make statements of the form "the probability of A is at least s ", where A is a classical formula, and s is a rational number from the interval $[0, 1]$, but with formulas themselves still remaining either true or false. The syntax, the semantics, and a complete axiomatization with appropriate inference rules have been encoded in Coq, and the soundness of the system (the validity of axioms with respect to the semantics and the preservation of validity by the inference rules) has been proven formally. The formal proof of completeness and an analysis of possible formalization of a SAT checker for this logic are in progress, and we expect sufficient progress for appropriate publication in the first half of 2010.

7. Other Grants and Activities

7.1. European Initiatives

7.1.1. Interreg Alcotra: myMed, 2010-2012

Participants: Luigi Liquori, Laurent Vanni, Vincenzo Ciancaglini, Claudio Casetti, Carla-Fabiana Chiasserini.

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>.

7.1.2. FP6 FET Global Computing: IST AEOLUS, 2005-2009

Participants: Luigi Liquori, Laurent Vanni.

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

7.1.3. FP6 TEMPUS DEUKS, 2007-2009

Participants: Luigi Liquori, Petar Maksimovic, Bojan Marinkovic [Math. Institute of Belgrade, Serbia].

DEUKS, *Doctoral School Towards European Knowledge Society*. The 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.

8. Dissemination

8.1. Participation in committees and referees

- Luigi Liquori is member of the *Commission de Spécialistes du jury CR2 INRIA Sophia Antipolis*.
- Luigi Liquori is PC member of the Sixth International Workshop on Hot Topics in Peer-to-Peer Systems HotP2P, 2010.
- Luigi Liquori is PC member of the SEMELS Workshop on Semantic Extensions to Middleware: Enabling Large Scale Knowledge Applications.
- Luigi Liquori is PC member of the 20th Tyrrhenian Workshop on Digital Communications "The Internet of Things".
- Luigi Liquori is PC member of 3rd Conference on Algebra and Coalgebra in Computer Science, CALCO 2009.
- Cédric Tedeschi is PC member of the Eleventh International Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS 2009.
- Cédric Tedeschi is PC member of the International Conference on Computational Science, ICCS 2009.
- Cédric Tedeschi is PC member of the the International Conference on Computational Science, ICCS 2010.

Moreover Luigi Liquori was a referee for the IEEE Symposium on Computers and Communications, the Journal of Logic and Computations, and the International Conference on Computational Science, ICCS 2009.

8.2. Teaching and Meeting organizations

- Luigi Liquori gave a course on Overlay and P2P networks at the DEUKS TEMPUS, Foundations of Information Technologies summer school June 14-27, 2009, Novi Sad, Serbia.
- Luigi Liquori gave a 9 TD course on Peer to peer, Master Ubinet UNSA.
- Luigi Liquori organized the DEUKS TEMPUS third training meeting in Sophia Antipolis.

8.3. Spare presentations

- Luigi Liquori presented the LogNet team to the ENS Lyon's students.
- Luigi Liquori presented the LogNet team to the Marseille Master's students.

- Bojan Marinkovic presented the Babelchord work at the DEUKS third training meeting in Sophia Antipolis.
- Cédric presented the BabelChord approach during a seminar on P2P systems at INRIA Sophia Antipolis.

8.4. Visitors

IN

- Giuseppe Persiano, full professor, U. Salerno, 1 month
- Alberto Trombetta, assistant professor, U. Insubria, 7 dd

OUT

Luigi Liquori visited the following sites:

- Politecnico di Torino, multiple visits,
- Università ti Torino, multiple visits
- University of Novi Sad, Serbia, 15dd
- Mathematical Institute of the Serbian Academy of Sciences and Arts, Belgrade, 1dd
- Scientific Computing Laboratory, Institute of Physics Belgrade, 1dd
- INRIA Lille Europe, 1dd

Cédric Tedeschi visited the following sites:

- INRIA Grenoble Rhône-Alpes, MOAIS Project, 1dd
- INRIA Rennes Bretagne Atlantique, PARIS Project, 1dd
- LIFC (Besancon), CARTOON Project, 1dd

9. Bibliography

Major publications by the team in recent years

- [1] D. BENZA, M. COSNARD, L. LIQUORI, M. VESIN. *Arigatoni: Overlaying Internet via Low Level Network Protocols*, in "JVA, John Vincent Atanasoff International Symposium on Modern Computing", IEEE, 2006, p. 82–91.
- [2] D. BORSETTI, C. CASETTI, C.-F. CHIASSERINI, L. LIQUORI. *Content Discovery in Heterogeneous Mobile Networks*, in "Heterogeneous Wireless Access Networks: Architectures and Protocols", E. HOSSAIN (editor), Springer-Verlag, 2008, p. 419–441.
- [3] R. CHAND, M. COSNARD, 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.
- [4] R. CHAND, M. COSNARD, L. LIQUORI. *Powerful resource discovery for Arigatoni overlay network*, in "Future Generation Computer Systems", vol. 1, n^o 21, 2008, p. 31–38.
- [5] R. CHAND, L. LIQUORI, M. COSNARD. *Improving Resource Discovery in the Arigatoni Overlay Network*, in "ARCS, International Conference on Architecture of Computing Systems", Lecture Notes in Computer Science, vol. 4415, Springer-Verlag, 2007, p. 98-111.

- [6] M. COSNARD, L. LIQUORI, R. CHAND. *Virtual Organizations in Arigatoni*, in "DCM, International Workshop on Developpment in Computational Models. Electr. Notes Theor. Comput. Sci.", vol. 171, n^o 3, 2007.
- [7] L. LIQUORI, D. BORSETTI, C. CASETTI, C.-F. CHIASSERINI. *An Overlay Architecture for Vehicular Networks*, in "IFIP Networking, International Conference on Networking", Lecture Notes in Computer Science, vol. 4982, Springer-Verlag, 2008, p. 60–71.
- [8] L. LIQUORI, M. COSNARD. *Logical Networks: Towards Foundations for Programmable Overlay Networks and Overlay Computing Systems*, in "TGC, Trustworthy Global Computing", Lecture Notes in Computer Science, vol. 4912, Springer-Verlag, 2007, p. 90–107.
- [9] L. LIQUORI, 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.
- [10] L. LIQUORI, B. P. SERPETTE. *iRho: An Imperative Rewriting-calculus*, in "MSCS, Mathematical Structures in Computer Science", vol. 18, n^o 3, 2008, p. 467-500.
- [11] L. LIQUORI, A. SPIWACK. *Extending FeatherTrait Java with Interfaces*, in "TCS, Theoretical Computer Science", vol. 398, n^o 1-3, 2008, p. 243–260, Calculi, types and applications: Essays in honour of M. Coppo, M. Dezani-Ciancaglini and S. Ronchi Della Rocca.
- [12] L. LIQUORI, A. SPIWACK. *FeatherTrait: A Modest Extension of Featherweight Java*, in "TOPLAS, ACM Transaction on Programming Languages and Systems", vol. 30, n^o 2, 2008.

Year Publications

Articles in International Peer-Reviewed Journal

- [13] E. CARON, F. DESPREZ, F. PETIT, C. TEDESCHI. *Snap-Stabilizing Prefix Tree for Peer-to-Peer Systems*, in "Parallel Processing Letters", jan 2009, To appear.

International Peer-Reviewed Conference/Proceedings

- [14] L. LIQUORI, C. TEDESCHI, F. BONGIOVANNI. *Babelchord: a social tower of DHT-based overlay networks*, in "IEEE, ISCC", 2009, p. 307-312.

Scientific Books (or Scientific Book chapters)

- [15] E. CARON, F. DESPREZ, F. PETIT, C. TEDESCHI. *DLPT: A P2P tool for Service Discovery in Grid Computing*, in "Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications", N. ANTONOPOULOS, G. EXARCHAKOS, M. LI, A. LIOTTA (editors), IGI Global, 2009.

Other Publications

- [16] D. DOUGHERTY, L. LIQUORI. *Logic and computation in a lambda calculus with intersection and union types*, 2009, submitted.

- [17] L. LIQUORI, C. TEDESCHI, L. VANNI, F. BONGIOVANNI, V. CIANCAGLINI, B. MARINKOVIC. *Synapse: A Scalable Protocol for Interconnecting Heterogeneous Overlay Networks*, 2009, submitted.

References in notes

- [18] J. W. BACKUS. *The IBM 701 Speedcoding System*, in "J. ACM", vol. 1, n^o 1, 1954, <http://doi.acm.org/10.1145/320764.320766>.
- [19] D. BENZA, M. COSNARD, L. LIQUORI, M. VESIN. *Arigatoni: Overlaying Internet via Low Level Network Protocols*, n^o 5805, INRIA, 2006, Technical report.
- [20] D. EPPSTEIN, Z. GALIL, G. ITALIANO. *Dynamic graph algorithms*, in "Handbook of Algorithms and Theory of Computation", chap. 22, CRC Press, 1998.
- [21] M. FIORE, J. HÄRRI, F. FILALI, C. BONNET. *Vehicular Mobility Simulation for VANETs*, in "Annual Simulation Symposium", 2007, p. 301-309.
- [22] A. RAPOPORT. *Mathematical models of social interaction*, vol. II, John Wiley and Sons, 1963, p. 493–579.
- [23] I. STOICA, R. MORRIS, D. KARGER, M. KAASHOEK, H. BALAKRISHNAN. *Chord: A Scalable Peer-to-Peer Lookup service for Internet Applications.*, in "ACM SIGCOMM", 2001, p. 149-160.
- [24] W. M. P. VAN DER AALST, A. H. M. TER HOFSTEDÉ. *YAWL: yet another workflow language*, in "Information System", vol. 30, n^o 4, 2005, p. 245-275.
- [25] J. VON NEUMANN. *The Principles of Large-Scale Computing Machines*, in "IEEE Ann. Hist. Comput.", vol. 10, n^o 4, 1988, p. 243–256.