Implementation Specification of Channel Reflector

Hitoshi Asaeda, Wacharapol Pokavanich

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

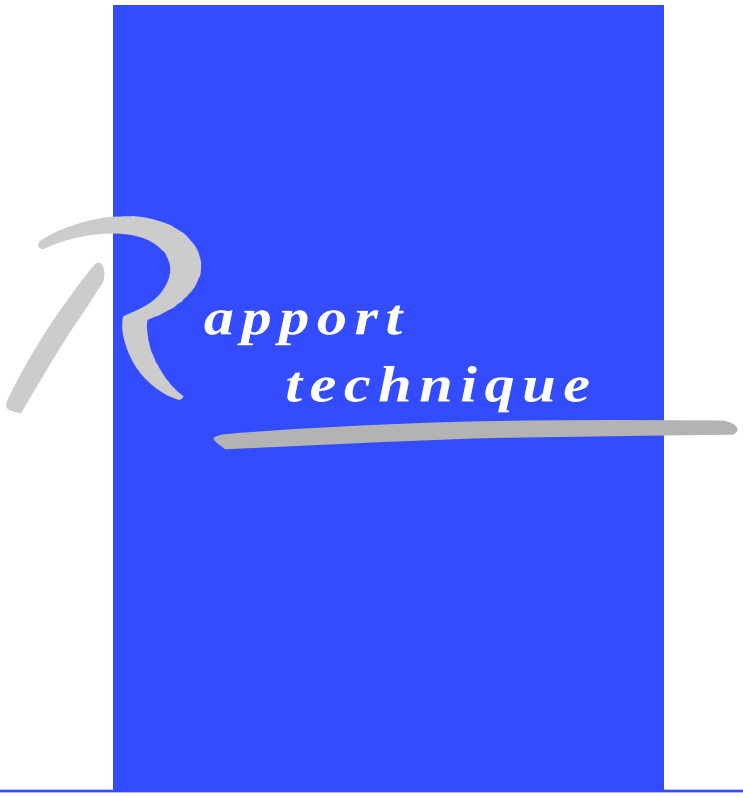
Implementation Specification of Channel Reflector

Hitoshi Asaeda — Wacharapol Pokavanich

N° 0305

April 2005

Thème COM



ISSN 0249-0803

Implementation Specification of Channel Reflector

Hitoshi Asaeda * , Wacharapol Pokavanich †

Thème COM — Systèmes communicants
Projet Planète

Rapport technique n° 0305 — April 2005 — 13 pages

Abstract: In this document, we detail an implementation specification of "Channel Reflector". Channel Reflector (CR) offers an effective policy and scope control technique for multicast channel announcement without use of a traditional SAP advertisement. It appears as a hierarchical directory system and each end user accesses this system as a regular Web server and retrieves available channel information via his Web browser.

According to this property, we have designed the CR implementation which can work with a common and well-known http server and database implementations. More precisely, our CR has been implemented as modules used with an Apache http server. In addition, while the channel information including the policy configuration is defined in XML format in CR, all the information are converted and stored in MySQL efficiently. Such the implementation strategy reduces the operator's maintenance cost and finally contributes to the easy deployment.

Key-words: Channel Reflector, multicast, implementation, channel announcement

* Hitoshi.Asaeda@sophia.inria.fr
† wach@cs.ait.ac.th

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Considération sur une implémentation d'un Réflecteur de Canal

Résumé : Dans ce rapport, nous détaillons une implémentation d'un "Réflecteur de Canal (Channel Reflector – CR)" qui est utilisé comme mécanisme d'annonce des sessions multipoints en remplacement du protocole SAP. Le concept principal est basé sur un système d'annuaire hiérarchique de sessions multipoints: l'utilisateur final accède à ce système comme un serveur de Web classique et récupère l'information des sessions multipoints à travers son navigateur.

En utilisant cette propriété, nous avons élaboré un CR qui peut communiquer avec un serveur http commun et connu ainsi qu'avec différentes implémentations de bases de données. Plus précisément, notre CR a été implanté en modules utilisés par un serveur http Apache. Alors que l'information sur les sessions multipoints ainsi que la politique de configuration sont décrites en format XML dans le CR, la totalité de l'information est convertie et emmagasinée dans MySQL de manière efficace. Une telle stratégie d'implémentation vise à réduire le coût d'entretien de l'opérateur et permet de rendre plus facile son déploiement.

Mots-clés : Réflecteur de Canal, multipoint, implémentation, annonce des sessions multipoints

1 Introduction

"Channel Reflector (CR)" [1, 2] aims to provide a feasible channel information distribution mechanism that accommodates any flavor of multicast services in the Internet. Since CR works as a Web-based channel directory system and does not require any protocol change to end users, end users can retrieve available channel information via their own Web browser. On the other hand, it does not cause any contradiction to the dynamic nature of the channel announcement scheme. This is because it provides all functions to manage Channel Entry and Scope List. Channel Entry is transmitted and synchronized with all particular CR systems that correspond to the specified Scope Label. Scope List is also transmitted along a tree-based structure (as in Figure 1) and synchronized when update is required. Regarding the advantage for a network administrator, it can transparently provide a multicast channel announcement policy and define a data distribution area by effective scoping technique without using SAP [3].

According to such the CR's property and considering the deployment advantage, however, it is an additional important factor that the operator's maintenance cost should be minimized. This point could be susceptible to the system implementation design and its installation cost.

In this report we explain our CR implementation design and detail[1]. Our implementation strategy follows above consideration; it cooperates with existing libraries and software implementations. For instance the CR implementation uses various CPAN modules and it is itself embedded as a module set used on top of an Apache http server. And while the multicast channel information including the policy configuration defined by an administrator is expressed with XML format, all of the information kept in CR are converted and stored in a well-known MySQL database manager efficiently.

2 Implementation

The CR system is implemented as perl modules and run on top of Apache http server version 2. Each module implementation is independent and therefore it could be easy to enhance and maintain the embedded functions. Our database implementation cooperates with MySQL database manager and all the information kept in CR are converted and stored in MySQL database efficiently. A user can easily access all CR's functions through two configuration interfaces; one is Web browser and the other is a command line interface (CLI). A user can also import channel entry from an XML data file (ascii format) and export to XML file through the CLI.

[1] This report is based on our CR-0.99.1 implementation (meaning the version number is 0.99.1). Some function and file name in this report may be different from the one in the later version.

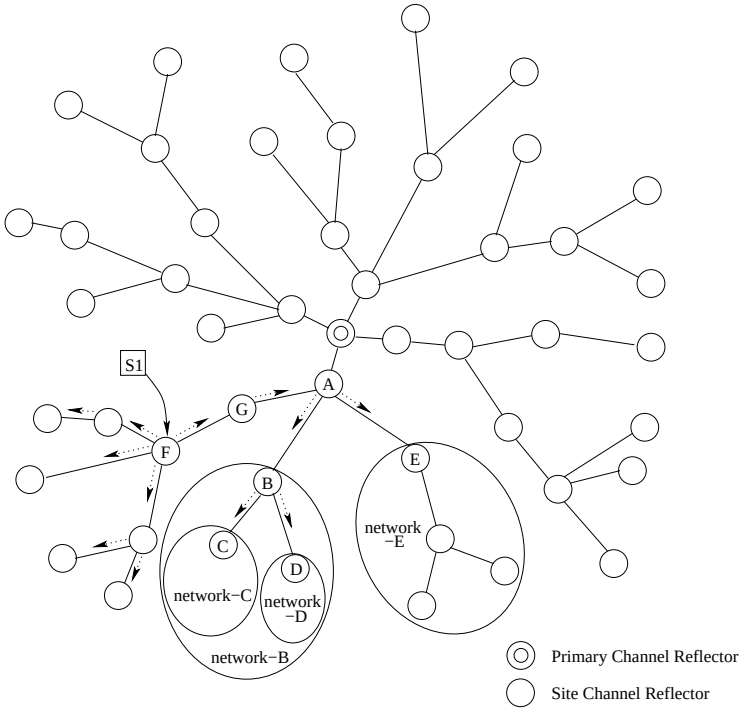


Figure 1: Channel Reflector policy tree

2.1 Prerequisite Components

Followings are the prerequisite modules for our implementation. Some of them may have own prerequisite modules and therefore the installation document "README" for each module should be also carefully referred.

- Perl 5.8.0 or later (version 5.8.6 is used for our current implementation)
- MySQL version 4.0 or later (version 4.0.22 is used)
- Apache version 2 (version 2.0.52 is used)
- CPAN perl modules:
  - CGI.pm-3.05
  - DBI-1.46
  - DBD-mysql-2.9004

  - libwww-perl-5.800 (Following modules are its prerequisite.)
    * MIME-Base64-3.05
    * URI-1.35
    * HTML-Parser-3.45 (Following module is its prerequisite.)
      · HTML-Tagset-3.04
    * libnet-1.19
    * Digest-MD5-2.33
    * Compress-Zlib-1.33
  - DBIx-XML_RDB-0.05
  - HTML-TokeParser-Simple-3.13
  - XML-XPath-1.13
    * XML-Parser-2.34

- Mod Perl version 2 (version 1.99.15 is used with our implementation)

2.2 Installation

1. Script Installation
   Extract CR source distribution file (CR-*version*.tar.gz where *version* shows the version number) under the directory which you want. For example you may see the following files:

```
./CR-0.99.1/index.html
./CR-0.99.1/COPYRIGHT
./CR-0.99.1/README

./CR-0.99.1/admin/register.cgi
./CR-0.99.1/admin/retrieve.cgi
./CR-0.99.1/admin/scope.cgi

./CR-0.99.1/conf/cr.conf
./CR-0.99.1/conf/httd.conf.example

./CR-0.99.1/img/back.jpg
./CR-0.99.1/img/but_ac.gif
./CR-0.99.1/img/but_inac.gif
./CR-0.99.1/img/cal.gif
./CR-0.99.1/img/forward.jpg

./CR-0.99.1/module/README
./CR-0.99.1/module/Makefile.PL
```

```
./CR-0.99.1/module/lib/CR.pm

./CR-0.99.1/tool/calendar.js
./CR-0.99.1/tool/calendar-en.js
./CR-0.99.1/tool/calendar-setup.js
./CR-0.99.1/tool/calendar-win2k-cold-1.css
./CR-0.99.1/tool/create_cr_table.sql

./CR-0.99.1/user/about.cgi
./CR-0.99.1/user/about.txt
./CR-0.99.1/user/search.cgi
```

2. CR Module Installation

```
% cd CR-0.99.1/module
% perl Makefile.PL
% make
# make install
```

2.3 Configuration

1. Configure a database and session information in "cr.conf".

```
$DB = database_name;
$DB_User = user_name; # user name of an administrator
$DB_Password = password; # password for an administrator
$BW_CHECK = 1; # specify whether bandwidth limit is checked or not
$BW_LIMIT = 2048; # bandwidth limit in kbps
@CATEGORY = ("Movie", "Music", "Sport");
@SESSION_TYPE = ("Test", "Meeting", "Conference", "Education", "Other");
```

   `@CATEGORY` and `@SESSION_TYPE` can be modified, but each name should be common among other CRs.

2. Configure an apache virtual host and its document root. Followings are the example configuration.

```
NameVirtualHost *:8081

<Directory /usr/local/CR-0.99.1>
    Options Indexes FollowSymLinks ExecCGI
    Allow from example.com
    AddHandler cgi-script .cgi
</Directory>
```

```
<VirtualHost *:8081>
  ServerAdmin admin@example.com
  DocumentRoot /usr/local/CR-0.99.1
  ServerName cr.example.com
  ErrorLog /var/log/httpd/cr.error_log
</VirtualHost>
```

3. Configure an access control. CR provides the CGI interfaces for network operators or administrators to configure the site CR or to register channel entries (under `admin` directory) and for the local users who are permitted to retrieve the channel information registered in the site CR (under `user` directory). An access control is therefore necessary to differentiate the network administrators and the local users. In our experience, `htpasswd` command helps setting up the configuration for `admin` directory. In this case, defining a virtual host for this directory could be also possible. Specifying "`Allow`" statement in "httpd.conf" would be one of the easy solutions for allowing the access from the local uses (without use of `.htpasswd` for them, as shown in above example).

   The detail information for configuring such the access controls can be referred in `http://httpd.apache.org/docs-2.0/howto/auth.html`.

4. After creating a new database, the administrator formats appropriate tables. `create_db_cr.sql` included in `tool` directory in the CR source distribution helps the procedure as follows. *database_name* in this reference must be the same name of `DB` in "cr.conf".

   ```
   % mysql -u user_name -p
   Enter password:
   mysql> create database database_name;
   mysql> use database_name;
   mysql> \. CR-0.99.1/tool/create_table_cr.sql
   ```

5. CR is now available. Open Web browser and go to, e.g. http://cr.example.com/index.html. Then you can click "Manage Channel Reflector" and add the local server information by using "Management" tab and "Define Local". "Scope Label" must be same of the virtual host name specified in `ServerName` in "httpd.conf". If the port number is not 80, you must also specify the port number just after the IPv4 address with ":" in "IP Address" field.

3 Database Design

We define two tables, "`channel`" and "`scope`", for storing data in CR. "`channel`" table is designed for storing multicast channel entries. All fields in this table follows the syntax and

the requirement defined in SDPng [4]. "scope" table is for storing the Scope List. All data types inherit MySQL data types. The schema of each table is listed below:

```
channel
- id                            bigint(20)
- multicast_addr                varchar(50)
- multicast_rtp_port            bigint(10)
- multicast_rtcp_port           bigint(10)
- m_addrtype                    varchar(10)
- scope_label                   varchar(50)
- scope_other                   varchar(50)
- ttl                           smallint(6)
- session_type                  varchar(50)
- session_other                 varchar(50)
- category                      varchar(50)
- start_date                    date
- start_time                    time
- end_date                      date
- end_time                      time
- session_info                  varchar(200)
- user_name                     varchar(50)
- user_email                    varchar(50)
- user_website                  varchar(100)
- source_addr                   varchar(50)
- s_addrtype                    varchar(10)
- alt_name                      varchar(50)
- alt_max                       varchar(10)
- rtp_name                      varchar(50)
- rtp_pt                        varchar(10)
- codec_name                    varchar(50)
- codec_encoding                varchar(50)
- codec_sampling                varchar(50)
- codec_channels                varchar(50)
- bw_limit                      double
- is_local                      tinyint(1)
- enabled                       tinyint(1)

scope
- id                            bigint(20)
- server_name                   varchar(50)
- server_addr                   varchar(50)
- app_path                      varchar(100)
- server_type                   varchar(10)
```

We also use an additional table, "`channel_tmp`". The structure of this table is exactly the same of "`channel`" table. This "`channel_tmp`" table is used in the channel retrieval part just for temporarily keeping the retrieved channel entry by several command (explained later).

4 Module Specification

A CR system consists of five perl modules; one is the core module that implements shared functions for the CR system, and the other modules provide congruent functions that are categorized by each function used with the core module. In these modules, the following external modules also cooperate with various common functions.

`use CGI;`
`use CGI::Carp qw(fatalsToBrowser);`
These two modules manage a CGI environment of the CR system.

`use DBI;`
This is an abstract layer module for database connection. `DBI` can connect to various database systems. Since our system currently uses MySQL, `DBD::MySQL` has to be also installed.

`use LWP;`
`use HTTP::Request::Common qw(POST);`
LWP and HTTP are used for remote server communication efficiently. They are used in a channel information distribution part.

`use DBIx::XML_RDB;`
This module is used for exporting data from database to XML format file.

`use HTML::TokeParser::Simple;`
This module extracts data from HTML format files.

`use XML::XPath;`
This module helps finding the specific XML tag and extracts the data from XML file. It is used for importing data from XML file to database.

4.1 Core Module (`CR.pm`)

This module is specified as "`use CR`" at the beginning of each CGI script. It creates the new CR object by calling `$CR = new CR`. We also can access the function implemented in this module by `$ra_query_results = $CR->RunQuery($dbh, $sql)`. Thanks to this module, the cost of code maintenance or additional coding for the congruent modules could be reduced.

4.2 Channel Management Module (`register.cgi`)

This module is the main part of maintaining the channel information and CR's configuration. It is called by the "Register" page which is in general accessed network operators or administrators of each site CR. This module supports the following functions. (The data flow of each function is controlled by corresponding CGI parameter named "`submit_xxx`" where xxx indicates an action.)

- Add Channel (`submit_add`)
  After the administrator inputs all required data for a new multicast channel entry, the site CR performs to register the channel entry to the local database. Firstly, the site CR verifies the validity of the channel information and checks whether the same multicast channel (using the same (S,G) pair) is registered in (or announced to) the site CR or not. If there is the same entry, the registration action is discarded. The CR also checks the bandwidth limitation if `$BW_CHECK` is defined in `cr.conf`. If the expected bandwidth of the new channel exceeds the limitation of the site CR, the channel entry will be disabled (i.e. `enabled` field is set to 0).

  The site CR next checks the specified Scope Label. If the Scope Label is not a local scope, a channel synchronization procedure among scoped CRs (i.e. all child CRs and their adjacent CRs toward the leaf CRs, and the parent CR toward the scope boundary CR) is performed. This procedure transmits the channel information to the scoped CRs by LWP (i.e. `HTTP::POST`).

  When the scoped CRs receive the request to register channel information by `register.cgi`, they again check the multicast channel. (This verification avoids an infinite loop in the environment that the server has a cyclic loop structure.) If the same multicast channel does not exist, the CR inserts the channel entry into the local database and checks the Scope Label whether the synchronization to other CRs is still required. The synchronization procedure is done hop-by-hop, and is terminated when all corresponding CRs successfully receive the new channel entry.

- Cancel Channel (`submit_delete`)
  An administrator can cancel a channel entry previously synchronized. This function requests to delete a specified channel entry from all scoped CRs to which the entry has been synchronized. This function is executed only on the CR on which the channel entry was originally registered. In detail, the implementation checks the permission by referring a value of `is_local` in "channel" table. If `is_local` is 1, it allows to delete the channel entry; otherwise, the request is rejected.

  If the Scope Label of the deleted channel is not local, the system calls LWP to delete the channel registered on other scoped CRs for the synchronization purpose. Each scoped CR then deletes the channel from each local database and forwards the same request to the adjacent CRs within the scope.

- Hide Channel (`submit_disable`)
  An administrator can hide a channel entry to the local users. This action aims to

filtering or disabling a channel announcement from the neighbor (congruent) CRs. This function does not affect anything to the neighbor CRs. In the program, it only sets the value of the field "enabled" to 0.

- Enable Channel (`submit_enable`)
  This function resumes the channel entry which was previously hidden (disabled), but does not affect anything to the neighbor CRs. The implementation sets value of the field "enabled" to 1.

- Export Channel (`submit_xml`)
  This function exports data from the local database to XML format (i.e. SDPng) data. An administrator can either specify only one channel entry or all channel entries to export. The output file is `conf/channel.xml`. The implementation of this part uses the functions derived from `DBIx::XML_RDB`.

- Import Channel (`submit_xml_in`)
  This function imports XML format data (`conf/channel.xml`) to the local database. `XML::XPath` is used for finding the specific XML tag and then the system can easily extract the data from each field of xml. Then all fields are inserted in the local database.

4.3 Scope Management Module (`scope.cgi`)

This module provides a user interface to an administrator to deal with a Scope List management. After the definition of a Scope Label of a site CR, the CR can next configure (add or delete) one parent CR and one or more child CRs based on the policy.

- Define Local (`submit_do_add_local`)
  This function defines a Scope Label of a site CR. It can be used not only for the initial configuration but for the modification of the Scope Label. On the other hand, this function does not send any message to other CRs and does not change nor affect any configuration on other CRs. Hence if an administrator modifies the Scope Label of the site CR, the parent and child CR's configurations may have to be changed (synchronized) as well, in order that the relevant configutations must not be inconsistent among other CRs.

- Add Child CR (`submit_do_add_child`)
  This function is used for defining a new child CR on a site CR. A site CR can repeat the configuration for adding moltiple child CRs. This function inserts a new child CR in "scope" table in the database, but the child CR is *not enabled* at that moment. In other words, while the site CR permits the configured node as the child CR but does not transmits any message until the child CR sends a `SCOPE_JOIN` message to this site CR.

- Delete Child CR (`submit_do_del_child`)
  This function is used for deleting a child CR configuration from a site CR's database. It is triggered when an administrator selects the menu from a user interface or when a site CR receives a `SCOPE_LEAVE` message from a child CR. In the latter case, this function makes the site CR disable the child CR (i.e. message sender). In other words, the site CR sets the field "enabled" for the child CR to 0 (which means disable), but the configuration is not deleted from the database. This function distinguishes these behaviors by specified CGI parameters.

- Add Parent CR (`submit_do_add_parent`)
  This function is used for defining a parent CR. A site CR sends a `SCOPE_JOIN` message (over LWP) to the parent CR. If the parent CR has an entry of the child CR in its accepted list, it permits (i.e. enables) the site CR as its child CR and sends the latest Scope List back to the site CR.

  After the site CR receives HTTP response from the parent CR and obtains the Scope List, it needs to synchronize the Scope List to its child CRs. Since each child CR synchronizes the Scope List to own child CRs by hop-by-hop manner as well, the Scope List is finally synchronized to all site CRs towards the leaf CRs along the policy tree. All of these functions are implemented in `SyncToChild` subroutine.

  In such scope synchronization procedures, each site CR appends own FQDN to the Scope List when it forwards its child CRs.

- Delete Parent CR (`submit_do_del_parent`)
  This function performs to delete a parent CR configuration. The site CR sends a `SCOPE_LEAVE` message (over LWP) to its parent CR and removes its parent CR's entry and its Scope List from the local database. The site CR also sends the same message to its child CRs for the synchronization purpose.

  When the parent CR receives the `SCOPE_LEAVE` message, it uses `submit_do_del_child` and disables the child CR by setting field "enabled" to 0 as explained above.

  When the child CR receives the `SCOPE_LEAVE` message, it deletes the site CR's parent CR and its upper CRs from own Scope List.

  Let us show an example. In Figure 1, CR-G sends a `SCOPE_LEAVE` message to CR-A and CR-F. While CR-A just disables CR-G as its non-child CR, CR-F deletes CR-A, CR-A's parent CR and upper CRs (i.e. primary CR) from its Scope List. CR-F then forwards the same `SCOPE_LEAVE` message to its child CRs to synchronize the deletion of these upper CRs. The implementation is done by hop-by-hop manner until the message reaches to the leaf CR of the policy tree.

- Export Configuration (`submit_xml`)

- Import Configuration (`submit_xml_in`)
  Channel information is also able to be imported or exported to or from specified XML formatted file. (`conf/channel.xml` is used if the file name is not specified.)

4.4 Channel Search Module (`search.cgi`)

This module provides a user interface for searching the channel entry from a local database on a site CR. A user can specify the search key words, e.g. multicast address, source address, category of the channel, user name, start date and end date. The system gets the criteria from the user input and generates an SQL statement (`select * from channel where ...`). It also supports both "exact match" and "partial" search. The response of the query is sorted by multicast address and displayed on the user interface.

4.5 Channel Retrieve Module (`retrieve.cgi`)

This module provides a user interface for searching the channel entry from a database on remote CRs (i.e. own parent or child CRs). An administrator can specify the search key words as with `search.cgi`. The system gets the criteria from the user input, generates an SQL statement and queries it from the remote database. The remote CR sends back the result set in HTML format (HTTP response). The system then extracts all results and stores them into the "`channel_tmp`" table and finally displays them on the user interface.

After the results are shown on the user interface, the administrator can select the channel entries to import it to the local database. If the selected channel entry does not exist in the local database, the system inserted it into "channel" table.

5 Acknowledgement

The authors would like to thank Mihai Bazon (mishoo@infoiasi.ro) who originally developed a beautiful DHTML calendar used in our implementation.

References

[1] H. Asaeda and V. Roca, "Consideration of Multicast Channel Announcement Architecture", INRIA Research Report, RR-4762, March 2003.

[2] H. Asaeda and V. Roca, "Policy and Scope Management for Multicast Channel Announcement", IEICE Transactions on Information and Systems, 2nd Quarter 2005.

[3] M. Handley, C. Perkins and E. Whelan, "Session Announcement Protocol", RFC2974, October 2000.

[4] D. Kutscher, J. Ott and C. Bormann, "Session Description and Capability Negotiation", Internet Draft - work in progress, draft-ietf-mmusic-sdpng-07.txt, October 2003.

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

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

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803