NOUVELLE FORMULE - NOUVELLE FORMULE - NOUVELLE FORMULE

N°171 ADMINISTRATION ET DÉVELOPPEMENT SUR SYSTÈMES OPEN SOURCE ET EMBARQUÉS

GNU LINUX MAGAZINE / FRANCE

PYTHON / ACTU
Découvrez les nouveautés apportées par Python 3.4 p.06

ANDROID / KITKAT
Plongez dans les entrailles de la dernière version d'Android p.64

SYSADMIN / ANALYSE
Paramétrez finement votre couple Collectd / PerfWatcher p.32

ALGO / IA
RENDEZ VOS PROGRAMMES VIVANTS
avec les algorithmes génétiques p.20

CODE / C
Utilisez tous les outils du développeur C p.72

REPÈRES / LIBRE
À cause d'une imprimante et d'un bourrage papier... p.18

HUMEUR / CODE
Dr KissCool revient sur les notions de sécurité p.14

L 19275 - 171 - F: 7,90 € - RD

France METRO : 7,90 € – CH : 13 CHF – BEL/PORT.CONT : 8,90 € – DOM TOM : 8,50 € – CAN : 14 $ cad – TUNISIE : 18 TND – MAR : 100 MAD

Une infogérance de qualité grâce aux solutions Open Source

DBM TECHNOLOGIES

UNE SOCIÉTÉ À TAILLE HUMAINE VOUS OFFRE LA COMPÉTENCE DES GRANDS

»» Venez nous rencontrer
Salon Solutions
Linux 2014 à Paris

Optimisez vos systèmes d'informations

Nous utilisons le meilleur de la technologie

- Virtualisation
- Messagerie / Groupware
- Sécurité réseaux (VPN, Proxy..)
- Téléphonie sur IP

Nous vous offrons des services personnalisables

- Adaptés à vos besoins
- Un suivi client maîtrisé
- Support téléphonique en France
- Développement spécifique

Rejoignez notre équipe
poste de Administrateur système et réseau Linux Junior

www.dbmtechnologies.com +33 (0)4 72 83 52 28 contact@dbmtechnologies.com

GNU/Linux Magazine France
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND

B.P. 20142 – 67603 Sélestat Cedex
Tél. : 03 67 10 00 20 – **Fax :** 03 67 10 00 21
E-mail : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com – boutique.ed-diamond.com

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Tristan Colombo
Réalisation graphique : Kathrin Scali & Jérémy Gall
Responsable publicité : Valérie Fréchard,
Tél. : 03 67 10 00 27
v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne

Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, **N° ISSN :** 1291-78 34
Commission paritaire : K78 976

Périodicité : Mensuel
Prix de vente : 7,90 €

FSC MIXTE Papier issu de sources responsables FSC® C015136

Membre April www.april.org

ÉDITORIAL

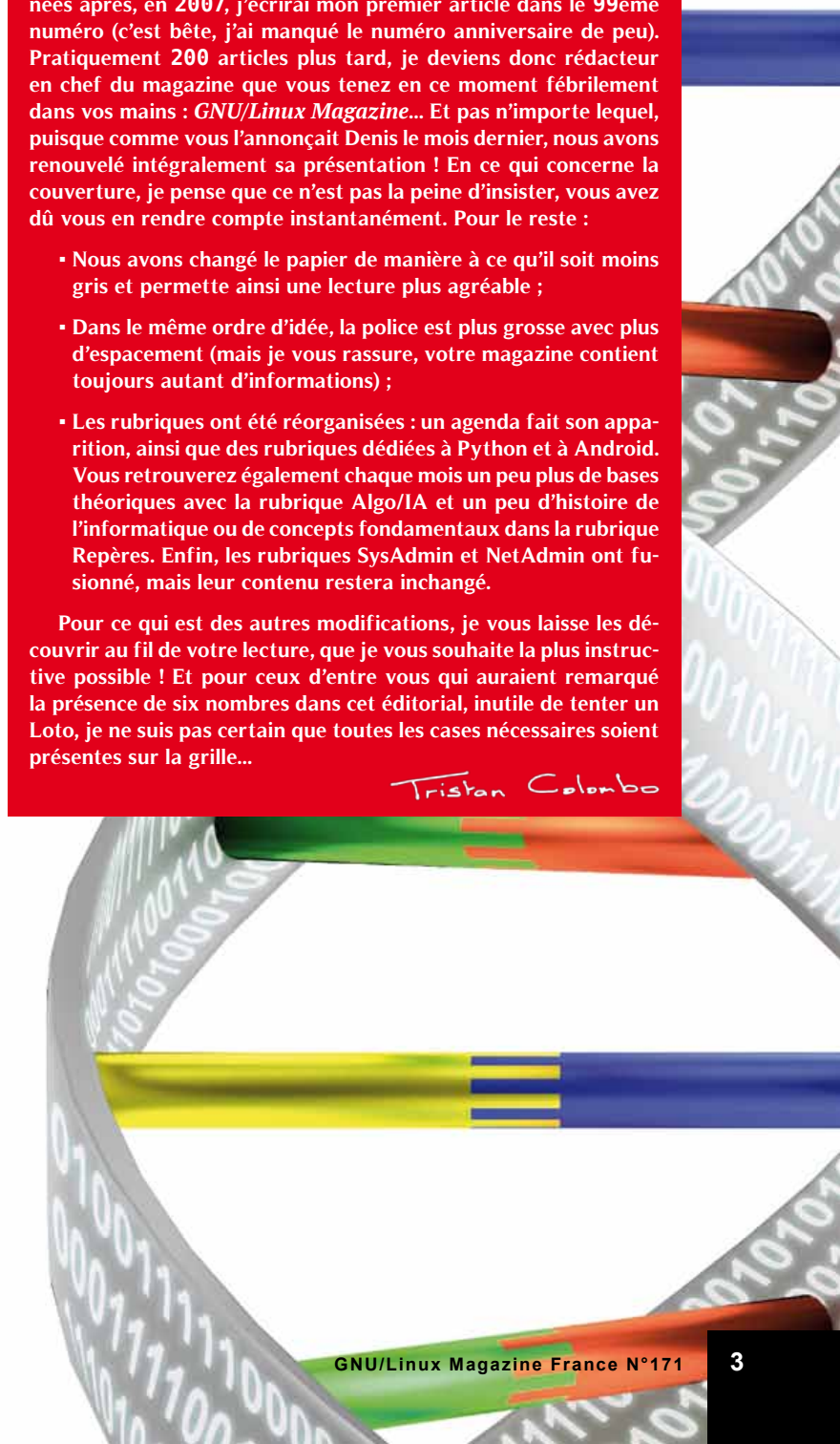
Et non... nous sommes en mai et Denis Bodor n'est plus présent dans ces colonnes. Ce n'était donc pas un poisson d'avril ! Ce 171ème numéro se fera donc sans lui... Enfin, quand je dis « sans lui », il n'est pas très loin non plus.

Je me souviens de la sortie de *Linux Magazine France* en 1998. J'étais alors en licence d'informatique et tout le monde cherchait désespérément à se procurer le précieux magazine alors fort peu distribué. Résultat : je n'ai jamais lu le premier numéro, je n'ai pu commencer qu'à partir du numéro 2 sans savoir que, des années après, en 2007, j'écrirai mon premier article dans le 99ème numéro (c'est bête, j'ai manqué le numéro anniversaire de peu). Pratiquement 200 articles plus tard, je deviens donc rédacteur en chef du magazine que vous tenez en ce moment fébrilement dans vos mains : *GNU/Linux Magazine*... Et pas n'importe lequel, puisque comme vous l'annonçait Denis le mois dernier, nous avons renouvelé intégralement sa présentation ! En ce qui concerne la couverture, je pense que ce n'est pas la peine d'insister, vous avez dû vous en rendre compte instantanément. Pour le reste :

- Nous avons changé le papier de manière à ce qu'il soit moins gris et permette ainsi une lecture plus agréable ;
- Dans le même ordre d'idée, la police est plus grosse avec plus d'espacement (mais je vous rassure, votre magazine contient toujours autant d'informations) ;
- Les rubriques ont été réorganisées : un agenda fait son apparition, ainsi que des rubriques dédiées à Python et à Android. Vous retrouverez également chaque mois un peu plus de bases théoriques avec la rubrique Algo/IA et un peu d'histoire de l'informatique ou de concepts fondamentaux dans la rubrique Repères. Enfin, les rubriques SysAdmin et NetAdmin ont fusionné, mais leur contenu restera inchangé.

Pour ce qui est des autres modifications, je vous laisse les découvrir au fil de votre lecture, que je vous souhaite la plus instructive possible ! Et pour ceux d'entre vous qui auraient remarqué la présence de six nombres dans cet éditorial, inutile de tenter un Loto, je ne suis pas certain que toutes les cases nécessaires soient présentes sur la grille...

Tristan Colombo



AGENDA

Dans cette nouvelle rubrique, nous vous présenterons les événements marquants des prochains mois, de manière à ce que vous ne ratiez plus les conférences importantes, ni les appels à contribution.

Si vous organisez des conférences techniques et que vous souhaitez que nous publiions un appel à contribution, n'hésitez pas à contacter la rédaction, mais pensez à le faire quelques mois avant la date limite de soumission !

Le Tremplin linuxembedded.fr

Date de remise du dossier :

- 1ère session : 15 mai
- 2ème session : 15 septembre

Site officiel : http://www.linuxembedded.fr/tremplin/

Concours apportant une aide concrète aux projets open source dédiés au domaine de l'embarqué. À gagner : 3000 € sous la forme de six chèques d'expertise (chaque chèque correspond à l'intervention d'un expert pendant une demi-journée (domaine choisi par les lauréats : technique, communication, organisation, etc.).

DjangoCon Europe 2014

Date : du 13 au 17 mai

Lieu : Ile des Embiez – France

Site officiel : http://2014.djangocon.eu

Conférence européenne des développeurs et utilisateurs de Django : « le framework Python pour les perfectionnistes pressés ». Les conférences auront lieu du 13 au 15 mai et les sprints (ou mises en pratique) auront lieu les 15 et 17 mai.

Sud Web 2014

Date : les 16 et 17 mai

Lieu : Toulouse – France

Site officiel : http://www.sudweb.fr/2014

L'événement, axé sur tout ce qui touche au Web, se décomposera en deux journées : une journée de partages d'expériences avec des conférences plénières et une journée d'expérimentation et de mises en pratique.

PGDay France

Date : les 5 et 6 juin

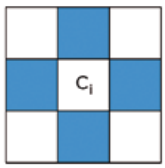
Lieu : Toulon – France

Site officiel : http://www.pgday.fr

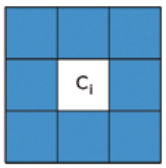
Couplé avec les dix ans de l'association PostgreSQL.fr, cet événement revêtira donc cette année un caractère particulier. Bruce Momjian, membre de l'équipe de développement (Core Team), sera présent.

SOMMAIRE

GNU LINUX MAGAZINE FRANCE
N° 171



Voisinage de Von Neumann



Voisinage de Moore

Les journées Perl

Date : les 13 et 14 juin

Lieu : Paris – France

Site officiel : http://www.journeesperl.fr/fpw2014

Événement gratuit, inscriptions sur le site web.

Échange et partage d'expérience autour du langage Perl quel qu'en soit le lien. L'événement est organisé par les Mongueurs de Perl autour de présentations de 5, 20 ou 45 minutes, qui s'adresseront à tout type de public.

PHP Tour 2014

Date : les 23 et 24 juin

Lieu : Lyon – France

Site officiel : http://www.afup.org/pages/phptourlyon2014

La thématique de cette année sera « Méthodologie de travail et industrialisation ». Le programme n'est pas encore connu au moment où ces lignes sont écrites, mais la présence de Rasmus Lerdorf, le créateur de PHP, est confirmée.

Exceptional Hardware Software Meeting

Date : les 27 et 29 juin

Lieu : Hambourg – Allemagne

Site officiel : http://www.ehsm.eu

Cette année l'EHSM se déroulera au DESY, le deuxième plus grand laboratoire de physique des particules d'Europe. Vous pourrez y suivre des conférences sur des sujets très variés tels que la fusion nucléaire, la conception de puces, le séquençage de génomes, etc. Des tutoriels auront lieu pour les publics de tous niveaux, y compris les enfants (apprendre à souder par exemple). ■

Donnons la priorité au logiciel libre – adhérez maintenant à l'April !

L'April (http://www.april.org) a pour mission de promouvoir le logiciel libre dans tous les domaines et de défendre les auteurs et les utilisateurs de logiciels libres face aux menaces telles que les DRM, les brevets logiciels… Depuis 1998, grâce à l'engagement de ses bénévoles, permanents, membres, sympathisants, l'April participe activement à la prise de conscience du public que la liberté informatique est un enjeu de société et au développement d'un environnement politique favorable au logiciel libre. En 2013, nos actions ont ainsi favorisé l'adoption, pour la première fois, d'une disposition législative donnant la priorité au logiciel libre.

L'April a lancé une campagne d'adhésion sur le thème « donnons la priorité au logiciel libre », afin d'augmenter sa capacité d'action, de donner la priorité au logiciel libre et de contribuer à construire une société plus libre, plus égalitaire et plus fraternelle.

Donnez-lui les moyens de poursuivre sa tâche, contribuez par votre nom à renforcer sa crédibilité. Adhérez dès maintenant : https://www.april.org/adherer. ■

PYTHON 3.4, L'ULTIME ÉTREINTE

par **Sébastien Chazallet** *[Ingénieur logiciels libres]*

Notre langage constricteur préféré vient de fournir une nouvelle version. Au programme : des nouveautés, certes, mais pas n'importe lesquelles ! En effet, si Python 3 a permis un retour aux sources (PEP 20, entre autres) et des améliorations considérables, il restait encore quelques petits détails qui faisaient que Python restait génial, mais pas parfait. Là, on se rapproche sacrément du Graal.

Python 3.4 venant tout juste de sortir, il n'est pas encore dans les dépôts des distributions. En général, ces dernières attendent qu'une version soit plus ou moins mature (en fonction du paramétrage) pour rendre les nouvelles versions disponibles.

Pour l'installer, vous devez donc télécharger les sources et les compiler. Pas d'inquiétude, c'est une opération facile à réaliser. Tout d'abord, il faut se rendre sur le site python.org. Là, on se doit de prendre un peu de temps pour admirer le nouveau site et pour se balader dans les menus. Une fois la balade terminée, on peut aller dans l'onglet ***Downloads***, puis ***Source code*** et enfin, cliquer sur ***Python 3.4***. Il ne reste plus qu'à décompresser l'archive, aller dans le répertoire, puis réaliser en tant que **root** le traditionnel :

```
./configure --prefix=/opt/python34 && make &&
make install
```

Cette opération installera Python 3.4 dans le répertoire **/opt** destiné à accueillir les logiciels installés manuellement.

Avant d'entrer dans le vif du sujet, il est important de se remémorer les principaux avantages de la version 3 de Python.

On ne peut pas ne pas commencer par citer la résolution d'un des problèmes principaux de Python 2, à savoir le GIL (*Global Interpréteur Lock*), qui certes avait eu son utilité, mais qui devenait un problème de plus en plus insurmontable pour les concepteurs du langage.

Le passage à l'Unicode est également l'un des points les plus importants. Certes, on n'a pas la culture de mettre des accents dans nos noms de méthodes ou de classes, mais c'est une transition qui a énormément de répercussions et qui fait qu'un langage est dit « moderne », car c'est quelque chose qui est devenu indispensable. Du coup, on a aussi des octets qui sont maintenant des octets, et réellement des octets. La séparation entre les deux types est devenue limpide et maintenant, les choses sont simples. On manipule du texte : chaînes de caractères Unicode. On échange des données sur du réseau : octets.

Viennent ensuite la réorganisation de la bibliothèque standard (un exemple représentatif est le module **io**, ou encore la fusion de **urllib** et **urllib2**), mais aussi l'adoption massive des générateurs et l'apport de **yield from**.

On ne peut pas ne pas citer l'amélioration de la sémantique objet :

```
class Python2Exemple1(object):
  pass

class Python3Exemple1:
  pass
```

Pour les méta-classes :

```
class Python2Exemple2(object):
  __metaclass__ = MetaClasse

class Python3Exemple2(metaclass=MetaClasse):
  pass
```

Ou encore pour l'utilisation de **super** (je continue de préférer pour ma part l'utilisation des appels statiques) :

```
class Python2Exemple3(object):
  def __init__(self):
      super(Parent, self).__init__()

class Python3Exemple3:
  def __init__(self):
      super().__init__()
```

Comme on peut le voir dans ces quelques lignes, les améliorations de Python 3 sont légion et vont dans le sens de ce que les développeurs attendent du langage.

Le présent article va s'attacher à décrire les nouveautés de la branche mineure 3.4.

1 Nouveautés

1.1 Module Enum, PEP 435

Un nouveau module Enum permet de gérer les énumérations. La documentation officielle est http://docs.python.org/3.4/library/enum.html. Pour faire simple, et pour donner des informations complémentaires de ce que vous trouverez dans la documentation, voici un petit exemple :

```
>>> from enum import Enum
>>> Actors = Enum('Actors', ['GrahamChapman', 'JohnCleese', 'EricIdle', 'MichaelPalin', 'TerryJones',
'TerryGilliam'])
>>> Actors
<enum 'Actors'>
>>> list(Actors)
[<Actors.GrahamChapman: 1>, <Actors.JohnCleese: 2>, <Actors.EricIdle: 3>, <Actors.MichaelPalin: 4>,
<Actors.TerryJones: 5>, <Actors.TerryGilliam: 6>]
```

Cela ressemble furieusement à un **namedtuple**, et ce n'est pas un hasard. Comme on pourra le remarquer, une variable d'énumération s'écrit en majuscules, à la façon d'une classe. Il s'agit en réalité d'une classe et l'écriture précédente en est une simplification. La même classe peut s'écrire ainsi :

```
class Actors(Enum):
  GrahamChapman = 1
  JohnCleese = 2
  EricIdle = 3
  MichaelPalin = 4
  TerryJones = 5
  TerryGilliam = 6
```

L'avantage de cette solution est que l'on peut choisir soi-même les numéros à affecter à chaque élément de l'énumération. Potentiellement, on peut d'ailleurs affecter un même numéro à différents éléments. À quoi cela peut-il servir d'avoir deux fois le même numéro ? À utiliser des alias. Chaque élément portant le même numéro est un alias, tout simplement. Du coup, cela reste logique. Pour s'assurer de leur unicité, on peut précéder la déclaration de la classe par le décorateur **enum.unique**.

Au final, les énumérations sont des objets différents, parce qu'utilisant une méta-classe différente. Elles utilisent les propriétés de Python différemment de la manière traditionnelle pour les adapter à ce que l'on attend d'elles.

1.2 Module statistics, PEP 450

Un des principaux reproches fait à Python était qu'il ne disposait pas de fonctions statistiques de base, par exemple calculer la moyenne d'une série de nombre. Bien entendu, Python dispose de *killer-modules*, tels que Numpy ou Scipy, qui ont tout ce dont on peut rêver dans le domaine, voire beaucoup plus. Mais installer et importer un énorme module juste pour une toute petite fonction, ce n'est pas la solution idéale.

Voici donc la raison de l'apparition du module **statistics** :

```
>>> liste = [1, 2, 2, 2, 3, 4, 7]
>>> mean(liste)
3.0
>>> median(liste)
2
>>> median_high(liste)
2
>>> median_low(liste)
2
>>> median_grouped(liste)
2.3333333333333335
>>> mode(liste)
2
```

Dans l'exemple précédent, on a une liste de 7 éléments. L'élément médian est celui qui se trouve au milieu

de la liste. Forcément, la médiane basse et haute ont la même valeur. Ce n'est pas le cas pour des listes qui ont un nombre d'éléments pair :

```
>>> liste2 = [1, 2, 2, 3, 4, 6]
>>> mean(liste2)
3.0
>>> median(liste2)
2.5
>>> median_high(liste2)
3
>>> median_low(liste2)
2
```

Ici, la médiane basse est le nombre sur la gauche du milieu de la liste et la médiane haute est celui sur la droite. Arbitrairement, la médiane est la moyenne entre ces deux nombres.

Voici enfin des fonctionnalités plus poussées, respectivement la déviation standard et la variance standard, puis la même chose, mais sur un échantillon. Pour avoir les formules, je vous renvoie à la Khan Academy !

```
>>> pstdev(liste)
1.8516401995451028
>>> pvariance(liste)
3.4285714285714284
>>> stdev(liste)
2.0
>>> variance(liste)
4.0
>>> liste = [1, 2, 2, 3, 7]
>>> mean(liste)
3.0
>>> median(liste)
2
```

1.3 Module pathlib, PEP 428

Ce module permet d'utiliser les chemins comme des objets et de les utiliser non plus comme des chaînes de caractères à travailler, mais comme des objets qu'il faut manipuler.

```
>>> from pathlib import Path
>>> mypath = Path('/opt')
>>> mypath.exists()
True
>>> mypath.is_dir()
True
```

On n'y retrouve pas que les équivalents objets de fonctions basiques, mais aussi d'autres très utiles :

```
>>> mypath = Path('/var/www')
>>> list(mypath.glob('**/*.html'))
[PosixPath('index.html'), PosixPath('doc/index.html')]
```

C'est un exemple particulièrement frappant des possibilités basiques que permet un langage comme Python. En effet, on redéfinit ici totalement les opérateurs pour leur donner une signification particulière :

```
>>> myfile = mypath / 'doc' / 'index.html'
>>> myfile.is_dir()
False
>>> doc, index = PurePath('doc'), PurePath('index.html')
>>> myfile = mypath / doc / index
```

Pour terminer, ce module permet également d'ouvrir des fichiers :

```
>>> with myfile.open() as f:
...     f.readline()
'<html>\n'
```

Et il y a encore pas mal d'autres possibilités intéressantes, simplifiant réellement la gestion des chemins.

La documentation se trouve ici : http://docs.python.org/3.4/library/pathlib.html.

1.4 Fonctions single-dispatch génériques, PEP 443

L'idée derrière ce nouvel utilitaire est de permettre de faire du polymorphisme à la Java. On peut ainsi disposer d'une seule fonction permettant de recevoir des signatures différentes. Attention cependant, on travaille sur un seul argument !

On commence par définir la fonction générique, puis les fonctions spécialisées pour des classes particulières, qui peuvent être des classes standards ou personnalisées :

```
>>> from functools import singledispatch
>>> @singledispatch
... def ma_fonction(arg):
...     print('Comportement par défaut')
...
>>> @ma_fonction.register(int)
... @ma_fonction.register(float)
... def _(arg):
...     print("Comportement pour un nombre")
...
>>> class Custom:
...     pass
...
>>> @ma_fonction.register(Custom)
... def _(arg):
...     print("Comportement pour une classe de custom")
...
```

Comme on peut le voir, on peut enregistrer une fonction pour plusieurs types en utilisant deux fois le décorateur. Ce dernier va en réalité tenir un registre contenant

1&1 DOMAINES

0,99 €*

Votre nom de domaine .fr .com

- **Inclus avec chaque nom de domaine :** redirections, gestion complète du domaine, création de sous-domaines
- **Support expert** 7j/7, 24h/24 via hotline non surtaxée et email

1&1

MEMBRE DE united internet

DOMAINES | MAIL | HÉBERGEMENT | E-COMMERCE | SERVEURS

0970 808 911 (appel non surtaxé)

1and1.fr

* Noms de domaine .fr et .com : 1 an à 0,99 € HT (1,19 € TTC). À l'issue de la 1ère année, le prix habituel s'applique. Offre à durée limitée et soumise à conditions détaillées disponibles sur 1and1.fr.

la liste des types et les fonctions associées. Ce registre peut être obtenu en faisant :

```
>>> ma_fonction.registry.keys()
```

Voici un exemple du résultat :

```
>>> ma_fonction('chaine')
Comportement par défaut
>>> ma_fonction([])
Comportement par défaut
>>> ma_fonction(1)
Comportement pour un entier
>>> ma_fonction(Custom())
Comportement pour une classe de custom
```

1.5 Module asyncio, PEP 3156

Python disposait d'un module **asyncore** qui était intéressant, mais qu'il fallait revoir parce que trop complexe, difficile à utiliser ou à intégrer. Avec le succès grandissant de solutions telles que Node.js, dû justement aux possibilités offertes par la programmation asynchrone, Python ne pouvait pas en rester là.

Il y a donc eu un *reboot* (ou *reborn*) et tels les héros de Marvel ou DC, la programmation asynchrone est revenue encore plus forte. Les fonctionnalités sont plus léchées, plus performantes, plus simples à utiliser.

C'est ainsi que l'on peut découvrir un tout nouveau module qui est à la hauteur des espérances, mais commençons par le commencement. Il s'agit d'un module de bas niveau, dont l'enjeu principal est très technique. Pour ceux que ça intéresse, autant aller à la source de l'information : http://legacy.python.org/dev/peps/pep-3156/.

Voici maintenant la documentation officielle http://docs.python.org/3.4/library/asyncio.html (qui permet de voir qu'il s'agit là d'un très gros morceau) et un petit exemple d'illustration, basé sur la résolution du problème des N-Reines.

Voici la résolution basique, faite de manière linéaire et synchrone :

```
from time import time
from itertools import permutations

def nqueens_basic(n):
    columns=range(n)
    for board in permutations(columns):
      if n == len(set(board[i]+i for i in columns)) \
           == len(set(board[i]-i for i in columns)):
              yield board
```

On voit que l'on peut parfaitement utiliser les avantages des générateurs, ce qui permet d'afficher les résultats au fur et à mesure qu'ils arrivent, en utilisant une boucle qui réalise un **print**.

On peut aussi s'amuser à mesurer son temps d'exécution ainsi :

```
t0=time()
res=list(nqueens_basic(10))
t1=time()
print('10-Dames basique        : %12.9f secondes' % (t1-t0))
```

Pour rappel, Python 3 avait apporté **concurrent.futures**, ce qui donnait une interface de haut niveau pour réaliser la tâche de manière concurrente, et qui pouvait améliorer les performances en utilisant le potentiel de bonnes machines :

```
from concurrent.futures import ProcessPoolExecutor, as_completed

def nqueens_workers_partial(n, max, current):
    columns, result = range(n), []
    for board in (p for i, p in enumerate(permutations(columns))
if i % max == current):
        if n == len(set(board[i]+i for i in columns)) \
             == len(set(board[i]-i for i in columns)):
                result.append(board)
    return result

def nqueens_workers(n, workers=4):
    result = []
    with ProcessPoolExecutor(max_workers=workers) as executor:
        futures={executor.submit(nqueens_workers_partial, n,
workers, i): n for i in range(workers)}
        for future in as_completed(futures):
            if future.exception() is not None:
              print(future.exception())
            else:
              if future.result():
              result.extend(future.result())
    return result
```

La méthode n'est pas forcément triviale. L'idée est de diviser le travail de manière intelligente, d'écrire une fonction qui prenne en charge une partie du travail, puis d'utiliser cette fonction dans le cadre d'un exécuteur.

C'est une fois que tout le monde a terminé son travail que les résultats sont rassemblés et peuvent être exploités. On perd le bénéfice du générateur, mais on gagne celui du temps de calcul (à partir du moment où l'on a la machine qui va avec, sinon on paie le coût de création des processus et ils s'attendent les uns les autres).

On peut faire une mesure du temps de traitement ainsi :

```
t0=time()
res=nqueens_workers(10)
t1=time()
print('10-Dames avec 4 workers : %12.9f secondes' % (t1-t0))
```

Il est temps maintenant d'aborder la nouveauté. Contrairement à l'exemple qui précède, l'idée est de travailler différemment, de manière asynchrone. On va ainsi diminuer le temps de traitement dans des contextes particuliers, notamment dans le contexte où l'on a des temps d'attente, que l'on peut maintenant éliminer.

Voici un nouveau code permettant de faire ceci :

```
import asyncio

@asyncio.coroutine
def nqueens_async_coroutine(future, n):
    columns=range(n)
    for board in permutations(columns):
        if n == len(set(board[i]+i for i in columns)) \
             == len(set(board[i]-i for i in columns)):
            future.set_result(board)

def nqueens_async(n):
    loop = asyncio.get_event_loop()
    future = asyncio.Future()
    asyncio.Task(nqueens_async_coroutine(future, n))
    loop.run_until_complete(future)
    #print(future.result())
    loop.close()
```

Là encore, on peut mesurer les performances facilement :

```
t0=time()
res=nqueens_workers(10)
t1=time()
print('10-Dames en asynchrone  : %12.9f secondes' % (t1-t0))
```

Il est à noter que l'asynchrone n'est pas une réponse universelle, et n'est pas non plus utilisé à des seules fins d'amélioration de performances, mais c'est une solution qui a le vent en poupe, en particulier dans le domaine, vaste, du Web.

2 Améliorations

2.1 Sécurisation de la destruction d'objets, PEP 442

Suite à une volonté d'améliorer la gestion de la mémoire et de la performance en général, et également pour résoudre plusieurs anomalies détectées ou fuites de mémoire constatées, en particulier lors d'une utilisation intense de générateur, la manière dont les objets sont détruits a été revue.

En effet, les objets peuvent disposer d'une méthode dédiée à la destruction des objets – il s'agit de la méthode **`__del__`** – et on est certain que celle-ci sera appelée une et une seule fois lorsque l'objet est supprimé. Par contre, rien ne permet de prédire l'ordre d'appel de ces méthodes.

Une fois qu'une méthode de destruction est appelée, le ramasse-miettes peut faire son travail.

2.2 SipHash, PEP 456

Les fonctions de hachage sont au cœur du langage de programmation Python et sont utilisées pour réaliser énormément de choses extrêmement basiques (clés de dictionnaire, ...). L'algorithme de hachage de Python, jusque-là, était une version de *Fowler-Noll-Vo* améliorée pour faire face à des problématiques de collision, entre autres.

Python rejoint maintenant Redis, Perl ou encore Haskell, en choisissant d'utiliser un nouvel algorithme qui permet à la fois de résoudre les problèmes qui sont actuellement rencontrés, mais qui est aussi très performant. Les détails de l'algorithme sont là : http://en.wikipedia.org/wiki/SipHash.

2.3 Nouveau protocole pour les objets picklés, PEP 3154

Pour rappel, **`pickle`** est un module qui permet de sérialiser un objet Python, ceci afin de l'échanger via le réseau à l'aide de différents services ou encore de l'enregistrer dans un fichier, par exemple, mais aussi dans des buts purement techniques. En tant que partie importante du langage et en tant que module de la bibliothèque standard, Python 3.0 avait introduit un nouveau protocole pour « pickler » un objet, la version 3, laquelle avait permis de lever les incompatibilités liées aux changements dans le langage.

Python 3.4 apporte la version 4 et cette fois-ci, on prend le temps de placer des améliorations, ce qui n'avait pas pu être réalisé il y a quelques années. Aussi, toutes les améliorations réalisées depuis le début de la version 3 ont été collectées et rassemblées afin d'être livrées en même temps, le changement de protocole **`pickle`** devant rester quelque chose d'exceptionnel.

Pour faire simple, le protocole est plus propre, a été amélioré au niveau des performances, mais il peut aussi s'appliquer sur plus d'objets et prend en compte le 64 bits.

2.4 Nouveau module tracemalloc, PEP 454

Une des principales difficultés lorsque l'on veut évaluer la performance d'une application Python consiste à évaluer la consommation mémoire. On est en effet parfaitement capable de déterminer les performances en termes de temps de traitement, on sait dire combien de fois une méthode a été appelée, on sait identifier les parties posant des problèmes de performances et on a même des outils graphiques pour visualiser ces données (KCacheGrind, pour ne pas le nommer).

Par contre, pour estimer la consommation mémoire, c'est beaucoup plus dur. On ne disposait jusqu'à présent que de quelques astuces ou quelques techniques difficiles à mettre en œuvre. Mais ce temps est révolu ! On peut maintenant avoir des détails et faire des évaluations.

Voici un premier exemple, tiré de la documentation officielle permettant d'avoir une idée de ce que l'on peut faire. Ici, il s'agit de trouver les 10 types d'objets les plus gourmands en mémoire :

```
import tracemalloc

tracemalloc.start()

# ... run your application ...

snapshot = tracemalloc.take_snapshot()
top_stats = snapshot.statistics('lineno')

print("[ Top 10 ]")
for stat in top_stats[:10]:
  print(stat)
```

Voici une ligne de ce résultat :

```
/usr/lib/python3.4/collections/__init__.py:368: size=244 KiB,
count=2315, average=108 B
```

On voit la ligne de code créant l'objet consommant de la mémoire, ici des n-uplets nommés. On peut ainsi lire que 2315 objets ont été créés, qu'ils font une taille moyenne de 108 octets pour un total de 244 kibioctets (1 KiB = 1024 octets).

Voici un autre exemple d'utilisation, appliqué à la détection d'une fuite de mémoire à un endroit particulier du code :

```
import tracemalloc
tracemalloc.start()
# ... start your application ...

snapshot1 = tracemalloc.take_snapshot()

# ... call the function leaking memory ...
snapshot2 = tracemalloc.take_snapshot()

top_stats = snapshot2.compare_to(snapshot1, 'lineno')

print("[ Top 10 differences ]")
for stat in top_stats[:10]:
  print(stat)
```

L'idée est de prendre deux captures et de les comparer pour faire émerger les objets ayant consommé de la mémoire entre-temps.

3 Du côté technique

3.1 Installateur embarqué, PEP 453

PIP est devenu l'installateur de paquet Python de référence. Une des étrangetés de Python est qu'il devait être installé à l'aide d'un autre installateur de paquet, **`easy_install`**, lequel devait à son tour s'installer à l'aide d'un paquet dédié. Historiquement, cela se tient. On comprend tous pourquoi on a cette situation, elle est un héritage du passé.

Mais d'un côté, PIP est maintenant stable et bien répandu et d'un autre côté, on ne respecte plus la PEP 20. Il était temps de simplifier les choses et de disposer maintenant directement de PIP dès lors que l'on a Python. D'une part, l'utilitaire est indispensable, d'autre part, il était temps de faire un peu de ménage.

3.2 Gestion des arguments pour les builtins, PEP 436

Les *builtins* sont des fonctionnalités de base du langage Python. Elles sont écrites en C et utilisables via une API qui réalise le pont entre l'implémentation de ces fonctions en C et leur utilisation au niveau de Python.

Cette API est relativement complexe, l'est devenue au cours du temps. L'idée est de proposer une alternative qui permette de gérer très simplement les signatures de fonctions et le *parsing* d'arguments, basée sur un prototype nommé *Argument Clinic*, que l'on pourrait traduire en consultation d'arguments.

L'idée est de faire quelque chose qui soit plus performant, mais surtout plus simple et plus familier pour les utilisateurs de Python.

3.3 Personnaliser l'allocation de la mémoire, PEP 445

Cette amélioration consiste en une nouvelle API C qui permet de mieux gérer la mémoire. Mieux dans le sens où l'on peut plus facilement personnaliser la manière dont la mémoire est gérée. Côté performance, la solution actuelle ne coûte pas plus cher que l'ancienne solution.

On dispose de structures C permettant de gérer des allocateurs de mémoire et de fonctions permettant de l'allouer ou la libérer, d'outils pour détecter des violations ou des problèmes divers et on n'utilise plus directement le `malloc`.

3.4 Descripteurs de fichiers, PEP 446

Les descripteurs de fichiers sont devenus des objets dont on ne peut hériter, de manière à prévenir les problèmes que cela peut causer, mais surtout, pour éviter des risques au niveau de la sécurité. Cette modification-là peut avoir des répercussions et entraîner des problèmes de rétro-compatibilité, ce qui déroge à une règle sacrée en Python. Toutefois, elle est quand même réalisée pour le « bien de l'humanité » (cf. la PEP), ce que je trouve plutôt cool, parce que ça claque !

3.5 Standardisation des métadonnées du mécanisme d'import des modules, PEP 451

Lorsque l'on exécute une instruction d'import, on utilise les noms des modules. Chaque module, en fonction de l'organisation des répertoires et fichiers Python, dispose d'un **nom qualifié** (*fully-qualified name*).

Un **chargeur** (*loader*) sera alors utilisé par le mécanisme d'import pour réaliser le chargement effectif du module, ce qui se fait en appelant la méthode `load_module` du chargeur, qui est une sorte de recette standardisée réalisant une succession d'opérations qui devraient être isolées.

Auparavant, il faut passer par l'utilisation d'un **chercheur** (*finder*), dont le rôle consiste à identifier le bon chargeur à utiliser, ce qui, jusqu'à maintenant, était réalisé à l'aide d'une méthode nommée `find_module`, qui renvoyait directement le chargeur.

Cette évolution ajoute une nouvelle classe nommée `ModuleSpec` au module déjà existant `importlib.machinery`. Cette nouvelle classe permet de capitaliser et fournir toutes les informations utilisées par le processus de chargement d'un module. Ces informations sont déconnectées du processus de chargement d'un module et elles peuvent ainsi être disponibles sans avoir à changer ledit module au préalable.

Les chercheurs pourront alors se contenter de fournir la **spécification** d'un module au lieu du chargeur, lequel pourra cependant toujours être fourni indirectement. La procédure de chargement est également ajustée pour bénéficier des avantages que cela apporte.

Conclusion

Dès Python 3.0, on savait que le nouveau langage était prometteur, il est sorti assez rapidement, ainsi que les outils nécessaires pour la migration ; et si tout n'était pas encore prêt, c'était largement plus qu'un simple *proof of concept*.

Python 3.1 a pu démontrer qu'il ne s'agissait pas que d'une simple promesse, a constitué une étape importante dans la continuité et a permis de terminer le plus gros du travail.

Python 3.2 a été la première version parfaitement opérationnelle et largement adaptée, elle a marqué l'adoption du langage par de gros projets, sans crainte de problèmes de performances ou autres. Cependant, la migration des principales bibliothèques restait difficile et longue.

Python 3.3 a été un tournant à ce niveau-là, puisque le langage a été assoupli pour permettre de faciliter encore plus la migration ; il a également apporté quelques améliorations notables. Un des résultats est que plus des trois quarts des bibliothèques importantes ont réalisé la migration (http://python3wos.appspot.com/).

Python 3.4 est une amélioration notable, fonctionnelle autant que technique, qui permet de démontrer que le travail fait précédemment ouvre de nouveaux champs et de nouvelles possibilités. On se rapproche toujours plus d'un langage plus propre, plus clair, plus sain, plus simple, plus *hype*, plus performant, plus solide, plus flexible, plus lisible, plus cohérent. Il s'agit d'un très bon cru, disposant d'améliorations dont certaines sont demandées par les développeurs depuis très longtemps. La migration vers Python 3 est un processus lent, prudent, mais parfaitement maîtrisé.

Il est important de terminer cet article en soulignant le fait que la communauté Python fait vraiment bien les choses. Vive la communauté, vive son système de gouvernance, vive les PEP, vive Python ! ■

LA SÉCURITÉ DANS UN CODE... MAIS POURQUOI ?

par ***Dr KissCool*** *(Attention au deuxième effet...)*

« Et surtout, n'oublie pas de vérifier les entrées utilisateurs ! ». Combien de fois avons-nous tous entendu cette phrase ? Pourquoi vérifier les entrées utilisateurs ? Il va essayer de faire passer de la drogue ??

De nos jours, il faut tout sécuriser : les cours d'école et les parcs sont revêtus d'un sol élastique, on installe des caméras de surveillance (attention, rien à voir avec la surveillance de vos faits et gestes, c'est pour « sécuriser » les rues) et donc, on va même jusqu'à vouloir sécuriser les programmes sous prétexte d'injection de code...

Avant de profiter des quelques lignes qui me sont offertes ici pour vous apporter de véritables informations non censurées, je tenais à me réjouir avec vous du départ de l'adorateur du Grand Serpent ! Grâce à notre action mes amis, nous avons réussi à repousser Denis Bodor hors de notre beau *GNU/Linux Magazine* ! Bientôt je ne serai plus confiné à mes deux petites pages, bientôt je prendrai les rênes de ce magazine et enfin, vous pourrez retrouver chaque mois avec joie des informations sensées, qui vous feront gagner un temps considérable. J'attends donc que la direction m'appelle pour m'annoncer ma promotion... J'attends toujours... Eh oh ? J'attends un coup de fil là ! Ah, mon oreillette crépite, ils ont préféré utiliser la technique désuète de Denis... Un hommage sans doute...

Quoi ?! Je viens d'apprendre que ce n'est pas moi qui aurai le poste ! C'est Tristan Colombo, un autre adepte du Grand Serpent, peut-être même pire que le précédent !!! Et en plus, une nouvelle rubrique « Python » apparaît ? Mais c'est une honte, c'est... Ouh là, ça n'a pas l'air de rigoler avec le petit nouveau, je viens de voir passer un auteur en larmes parce qu'il n'avait pas respecté les directives d'insertion de figures dans les articles... Bon, je n'ai pas encore eu droit aux électrochocs, donc je vais essayer de me calmer pour voir comment ça se passe. Peut-être qu'en agissant sournoisement j'arriverai à caser un article sur PHP...

Revenons donc à la sécurité des données. Vu la place qui m'est gracieusement octroyée, je ne rentrerai pas trop dans les détails et me cantonnerai aux injections SQL, aux injections de code et aux sommes de contrôle.

1 Injections SQL

Comme le disait notre « bien aimé » rédacteur en chef dans son « magnifique » article paru 2011 dans *Linux Pratique* n°63 intitulé « Applications et sites web : avez-vous pensé à la sécurité ? », une injection SQL consiste à rendre une requête SQL valide alors même qu'elle n'est pas entièrement exécutée (vous n'imaginez pas le temps que j'ai passé en recherches pour trouver ce torchon qui... Ah non, tiens, à l'époque les exemples étaient en PHP, il n'est peut-être pas complètement irrécupérable). Prenons donc un exemple où l'on souhaite se connecter à un système à l'aide d'un identifiant et d'un mot de passe. La requête sera :

```
SELECT `ID_USER` FROM `SECURITY`.`USER` WHERE
`LOGIN`='login' AND `PASSWORD`=SHA1('password')
```

Notez déjà ici que le pauvre développeur aura stocké le mot de passe de manière chiffrée dans la base, alors que ça aurait été tellement plus simple de l'enregistrer en clair ! En plus, en cas d'oubli, la procédure de récupération du mot de passe serait bien plus rapide : un mail avec votre mot de passe et le tour est joué ! Mais bien sûr... ah, ça y est, l'oreillette... il ne lui aura pas fallu longtemps pour adopter les techniques de son prédécesseur... Donc, on me dit que si la base de données est crackée, les hackers auront accès à l'ensemble

des couples d'identifiants/mots de passe. Moui... c'est vrai... mais c'est un faux problème ! On essaye encore de me faire dire n'importe quoi ! C'est plus simple de ne pas chiffrer, donc inutile de se compliquer la tâche !

Ici, l'utilisateur va ensuite saisir dans des champs son identifiant et son mot de passe. Si l'utilisateur ne saisit pas son vrai nom (avouez qu'il faudrait être vraiment tordu, mais soit, admettons), la requête peut devenir :

```
SELECT `ID_USER` FROM `SECURITY`.`USER` WHERE `LOGIN`='' OR 1=1
--' AND `PASSWORD`=SHA1('password')
```

Comme **--** signifie un début de commentaire, la requête correspond en fait à :

```
SELECT `ID_USER` FROM `SECURITY`.`USER` WHERE `LOGIN`='' OR 1=1
```

En regardant une table de vérité du **OR**, on s'aperçoit que ce test sera toujours vérifié :

x	y	x OR y
FAUX	FAUX	FAUX
FAUX	**VRAI**	**VRAI**
VRAI	**FAUX**	**VRAI**
VRAI	VRAI	VRAI

Nous serions donc censés vérifier toutes les saisies utilisateurs pour qu'elles ne détournent pas le sens d'une requête SQL... Non mais vous vous rendez compte du boulot ?! Vous pensez sincèrement que des gens s'amusent à faire ça avec, comme on vient de le voir, le temps que ça peut prendre ? Allons, allons, un peu de bon sens ! Ma démonstration est implacable : inutile de se soucier des injections SQL !

2 Injections de code

L'injection de code est similaire à l'injection SQL, mais directement dans un programme. Et là, les amis, nous allons pouvoir nous faire plaisir ! Là, nous allons montrer à la face du monde ce que vaut réellement ce pseudo langage qu'est Python !!! Prenons donc un petit serpent en version 2.7 et demandons à un utilisateur de saisir un entier :

```
>>> n = input("Veuillez saisir un entier : ")
Veuillez saisir un entier : 2
```

Bien sûr, en ayant saisi un entier, tout se passe bien...

```
>>> print n
2
```

Donnons maintenant une chaîne de caractères :

```
>>> n = input("Veuillez saisir un entier : ")
Veuillez saisir un entier : linux
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name 'linux' is not defined
```

Ah ? On ne nous dit pas qu'il ne s'agit pas d'un entier, mais que « linux » n'est pas défini... Étrange... Essayons donc de définir une variable **linux** :

```
>>> linux = 12
>>> n = input("Veuillez saisir un entier : ")
Veuillez saisir un entier : linux
>>> print n
12
```

Eh oui... Je vois que du côté des adorateurs du Grand Serpent on commence à faire la tête ! La valeur contenue par la variable a été utilisée et stockée dans **n**.

Ahahah ! Et c'est un langage ça ?! Allez, allons un peu plus loin et regardons donc ce que fait la fonction **input** :

```
>>> help(input)
Help on built-in function input in module __builtin__:

input(...)
    input([prompt]) -> value

    Equivalent to eval(raw_input(prompt)).
```

Oui, vous avez bien lu : la fonction **input** effectue un **eval** d'une autre fonction ! **eval** sert à évaluer une chaîne de caractères comme s'il s'agissait d'un code. Mais alors, ça veut dire que l'on peut exécuter arbitrairement un code ? Non... ça serait trop beau... Ahah allez, tentons-le :

```
>>> n = input("Veuillez saisir un entier : ")
Veuillez saisir un entier : __import__("os").system("rm
-R *")
```

Ça marche ! Ça maaarche !!! Ahahah, en voilà un langage intéressant... ahah... mais que... il est en train d'effacer mon disque ! J'aurais dû tester avec un **ls** !

Bon, j'ai réussi à sauver mon article. Alors que dire face à ça ? Simplement que, pour une fois, Python a appliqué une philosophie compréhensible par tous : l'utilisateur est gentil, il ne va pas essayer de tout casser et il est donc inutile de tester tout ce qu'il envoie à notre code. J'avoue tout de même avoir pris un malin plaisir à tordre ce reptile, mais il s'agissait d'une expérience purement scientifique visant à montrer que les partisans de ce langage, fort enclins à la critique envers les autres langages, étaient en fait logés à la même enseigne et que... encore l'oreillette, je vous prie de m'excuser... Comment ? En Python 3 ça a été corrigé ?? Mais c'est une hérésie !!! Eh bien pour me venger, je vais me lancer dans une grande croisade à la recherche des codes Python 2.7 dans lesquels je ferai des injections de code.... Mouuuuahahahah !

3 Sommes de contrôle

Reprenons nos esprits. La somme de contrôle est un mécanisme permettant de s'assurer que l'information n'a pas été corrompue. Pour cela, on utilise des fonctions de hachage dont le **md5**. Voici un exemple d'application en PHP :

```
<?php

    $a = md5('240610708');
    $b = md5('QNKCDZO');

    echo "$a\n";
    echo "$b\n";
    echo "\n";

    var_dump($a == $b);

     if ($a == $b)
    {
         print "OKY\n";
    }
```

En exécutant ce code, nous obtenons :

```
0e462097431906509019562988736854
0e830400451993494058024219903391

bool(true)
OKY
```

Certains diront que cela vient du fait que PHP est un langage à typage dynamique faiblement typé (notez l'utilisation du terme dévalorisant « faiblement ») et qu'il faut utiliser l'opérateur d'identité et non d'égalité (encore des termes barbares pour nous embrouiller). Mais si ça marche comme ça en PHP, c'est que c'est la bonne façon de faire : une somme **md5** ça sert à rassurer les gens ; après, que les résultats soient ou non égaux n'a que très peu d'importance. J'ai même vu dans des entreprises des responsables (sérieux pour une fois), demander à leurs équipes de coder les valeurs des tests en dur. Ainsi, on obtient par exemple :

```
if (True)
{
    print "Test de somme : OK"
}
```

Du coup, tout le monde est content ! Les développeurs qui ont économisé du temps et les clients qui peuvent voir que les valeurs sont correctes !

D'ailleurs, qu'est-ce que donne cet exemple dans le langage adoré de notre « cher » rédacteur en chef ?

```
>>> import md5
>>> a = md5.md5('240610708')
>>> b = md5.md5('QNKCDZO')
>>> a == b
False
```

Hum... coup de chance... on oublie.

Conclusion

Nous l'avons vu, il existe des solutions pour gagner un temps précieux. La meilleure d'entre elles est la technique consistant à faire confiance à vos utilisateurs et à tous les considérer comme bienveillants. Comment ça « technique Bisounours » ?! Ah, il ne lui a vraiment pas fallu longtemps au gourou du Grand Serpent pour s'adapter et... aïeuuuh ! J'ai même droit aux électrochocs ! C'est scandaleux ! Autant garder l'autre si c'est comme ça : Denis, revieennnnnt !!!

Et n'oubliez pas :

« Il n'y a pas de mauvais langage... il n'y a que de mauvais développeurs ». ■

PROFESSIONNELS DES TICE, COLLECTIVITÉS, ÉCOLES D'INGÉNIEURS, UNIVERSITÉS, R & D, ENSEIGNANTS, ...

GNU LINUX MAGAZINE / FRANCE

VOUS PROPOSE 2 NOUVEAUX SERVICES !

1 VOUS SOUHAITEZ LIRE GNU/LINUX MAGAZINE EN VERSION PDF ?

PDF

LINUX MAGAZINE / FRANCE 11 NOS

VOICI LES ABONNEMENTS PDF COLLECTIFS !

Ce service vous permet d'abonner votre structure (écoles, collectivités, entreprises, etc.) à l'édition PDF de nos magazines afin d'en profiter dès leur parution chez les marchands de journaux.

Sans DRM, téléchargez simplement, lisez et annotez vos eBooks sur votre PC, smartphone ou liseuse électronique.

2 VOUS SOUHAITEZ RETROUVER ET CONSULTER LES ARTICLES DE GLMF ?

BASE DOCUMENTAIRE LINUX MAGAZINE / FRANCE + HORS-SÉRIES

VOICI LA BASE DOCUMENTAIRE !

L'accès à la base documentaire en ligne de GNU/Linux Magazine et de ses Guides vous permettra d'effectuer des recherches dans la majorité des articles parus, qui seront disponibles 6 mois après leur parution en magazine. Vous pourrez ainsi effectuer des recherches sur les articles indexés, copier les codes, etc.

La consultation s'effectue sur notre nouveau service **connect.ed-diamond.com** qui est en place depuis janvier 2014 (n'hésitez pas à le visiter !)...

connect.ed-diamond.com

N'hésitez pas à consulter notre offre sur **boutique.ed-diamond.com**,

« **Base Documentaire TOTALE** » à **399 € HT/an**

(5 connexions comprises) pour profiter de la base documentaire de l'ensemble des parutions des Éditions Diamond !

BESOIN DE RENSEIGNEMENTS SUPPLÉMENTAIRES OU D'UN DEVIS SUR MESURE ?

N'hésitez pas à envoyer un e-mail à **aboprof@ed-diamond.com** ou à téléphoner au **+33 (0)3 67 10 00 27**

À CAUSE D'UNE IMPRIMANTE ET D'UN BOURRAGE PAPIER

par ***Tristan Colombo***

Tout le mouvement du logiciel libre a été lancé par un événement en apparence mineur, lié à un objet aujourd'hui banal : une simple imprimante.

Au début des années 1980, lorsque Xerox offre une nouvelle imprimante au laboratoire d'Intelligence Artificielle (AI Lab) au *Massachussets Institute of Technology* (plus connu sous le nom de MIT), ils sont loin de penser aux implications que va avoir leur simple geste commercial. Un personnage, bien connu du petit monde du logiciel libre, va se révéler, se projeter sur le devant de la scène, à cause d'une imprimante. Il s'agit bien sûr de Richard Stallman alors développeur au sein du laboratoire. Comme il existe de nombreuses versions de cette histoire contenant toutes de légères différences, j'ai décidé de ne me baser que sur les propos validés par Richard Stallman : sa biographie qu'il a lui-même relue et corrigée **[1]** et l'un de ses nombreux entretiens **[2]**. Revenons au début de l'histoire...

Au commencement était donc cette fameuse imprimante livrée au AI Lab par Xerox. La date exacte n'est pas indiquée, mais Richard Stallman ayant 27 ans à l'époque et 61 aujourd'hui, on peut en déduire que c'était en 1980. Il ne s'agissait pas de n'importe quelle imprimante, puisque c'était une imprimante laser, une révolution pour l'époque : l'impression était beaucoup plus rapide et les graphismes plus nets. Le laboratoire l'adopta donc immédiatement... Et ils découvrirent rapidement le problème majeur de cette imprimante : les bourrages papier. Pour être sûr que le processus d'impression se déroule sans interruption, ou qu'il n'ait pas été mis en file d'attente à cause d'un autre travail interrompu pour cause de bourrage papier, il fallait donc rester planté devant la machine, prêt à intervenir. Sur la précédente imprimante, Richard Stallman avait trouvé une solution élégante en écrivant un programme qui allait régulièrement analyser l'état des impressions et envoyer un message aux utilisateurs une fois la tâche terminée ou en cas de problème. Il se dit donc qu'il n'aurait qu'à adapter son programme à la nouvelle imprimante.

Fig. 1 : Richard Stallman en Saint IGNUcius (CC By-Sa 3.0 par Johnny McCullagh)

Pour pouvoir dialoguer avec celle-ci, il avait besoin d'accéder à son code source et là, ce fut la première surprise : alors que jusqu'à présent les éditeurs fournissaient ce code, cette fois-ci, seule la version compilée était disponible. Bien sûr, il était possible de désassembler le code, mais cela représentait un travail considérable pour une modification somme toute mineure. L'imprimante étant un cadeau, et l'absence de code source étant de toute évidence calculée, Richard Stallman n'essaya même pas de contacter Xerox pour l'obtenir. Il n'y avait donc pas de solution et il fallait se résoudre à surveiller de près chaque travail d'impression.

Quelques temps plus tard, Richard Stallman apprit qu'un collègue de l'université de Carnegie Mellon avait en sa possession le précieux code source de l'imprimante. C'est donc plein d'optimisme et en pleine confiance qu'il lui rendit visite. La rencontre fut brève et donna lieu à une seconde surprise, puisque le chercheur refusa de lui communiquer le code en lui indiquant qu'il s'était engagé à n'en divulguer aucune copie. Richard Stallman en ressortit furieux : pour la première fois de sa vie, il venait de se confronter à une clause de confidentialité. Les idées

qui germaient en lui sur le partage des logiciels trouvèrent là un catalyseur et c'est ainsi que d'un événement mineur allait naître en 1983 le projet GNU.

GNU, acronyme de « GNU is Not UNIX », est le projet d'un système d'exploitation compatible UNIX et librement partagé. Pour assurer la protection légale du projet, la *Free Software Foundation* (FSF) est créée en 1985. C'est ainsi qu'apparaîtront la notion de *copyleft* et la fameuse licence GNU GPL (*GNU General Public License*). L'objectif de cette association va un peu plus loin avec la promotion du logiciel libre et la défense des utilisateurs. GNU est donc un projet de système d'exploitation et dans un système d'exploitation, il y a de nombreux composants (bibliothèques système, compilateur, etc.), mais l'un d'entre eux est essentiel : le noyau, qui gère les tâches, la mémoire, etc. Le noyau du projet GNU s'appelle GNU Hurd et à l'heure actuelle, il n'en existe toujours pas de version stable. Sans noyau, il n'y a pas de système d'exploitation... Mais réciproquement, un noyau qui n'aurait pas accès à tous les composants GNU ne peut pas non plus former un système d'exploitation. En 1991, Linus Torvalds a fourni un noyau qu'il a appelé Linux. Les systèmes que nous utilisons ne sont donc pas des systèmes Linux, mais GNU/Linux, et vous ne lisez pas *Linux Magazine*, mais bien *GNU/Linux Magazine*.

Depuis ces épisodes, Richard Stallman arpente le monde pour porter la bonne parole. Il s'est construit un personnage volontairement provocateur, mais qui pointe avec raison de nombreux problèmes de liberté liés aux systèmes informatiques. En fin de conférence, il incarne ainsi souvent Saint IGNUcius de l'église d'Emacs, revêtu d'une aube et d'une « auréole-disque-dur » comme le montre la figure 1. En tant que développeur d'Emacs, il fustige les utilisateurs de Vi (« VI VI VI, le chiffre de la bête ! »).

Richard Stallman, personnage emblématique du logiciel libre, et même le mouvement du logiciel libre, sont donc nés d'une rencontre avec une imprimante qui faisait des bourrages papier et dont les pilotes n'étaient pas accessibles librement... ■

Références

[1] Stallman R. M., Williams S., Masutti C., « Richard Stallman et la révolution du logiciel libre – Une biographie autorisée », éd. Eyrolles, 2010

[2] Fin 1970, Du bourrage papier au logiciel libre – Arte (entretien avec Richard Stallman) : http://www.youtube.com/watch?v=zxbvt6wpmP8

DE LA BIOLOGIE DANS DU CODE : LES ALGORITHMES GÉNÉTIQUES

par ***Tristan Colombo***

Existe-t-il un lien entre un être vivant, aussi simple soit-il, et un programme ? On serait tenté de répondre immédiatement non... Mais en fait, la réponse ne peut pas être aussi catégorique, tout dépend de la façon dont a été codé le programme !

De tout temps, l'Homme s'est inspiré de la nature pour ses inventions : la fermeture Velcro est née de l'observation des fruits secs de la bardane, qui comportent de minuscules crochets très recourbés et qui s'accrochent à n'importe quel tissu ; le gilet pare-balles emprunte sa structure à la soie des fils d'araignée, qui sont dix fois plus résistants que l'acier (et un peu plus légers aussi), etc. C'est de la bionique : l'utilisation de modèles vivants en vue de réalisations techniques (comme pour Steve Austin, l'homme qui valait trois milliards). Pourquoi donc ne pas puiser dans la nature l'inspiration pour créer des méthodes de résolution de problèmes ? C'est ce qui a été fait avec notamment les algorithmes génétiques, qui sont une sous-classe des algorithmes évolutionnaires **[1]**, également appelés algorithmes évolutionnistes ou AE pour les intimes. Les algorithmes génétiques (GA) sont apparus en 1975 **[2]** et ils sont les précurseurs des algorithmes évolutionnistes, bien qu'en tant que sous-classe on puisse penser le contraire : la généralisation est intervenue postérieurement.

À quoi servent ces algorithmes ? Ils sont issus d'un domaine très particulier de l'informatique théorique : l'optimisation calculatoire, qui consiste à trouver les meilleures solutions pour un problème donné. Tout le monde a déjà entendu parler du problème du voyageur de commerce (*Travelling Salesman Problem* ou TSP en anglais), exposé pour la première fois par W. Hamilton en 1859. La résolution de ce problème où un marchand doit parcourir toutes les villes d'un pays en effectuant le plus court chemin peut trouver une solution grâce à l'optimisation. Ainsi, les algorithmes génétiques permettent d'identifier au moins une solution à un problème d'optimisation, sans garantir que la solution soit optimale (c'est la définition de ce que l'on nomme des heuristiques). En fait, on peut même aller plus loin en disant que les algorithmes génétiques constituent une stratégie générale qui peut être appliquée à un grand nombre de problèmes (à condition de les représenter correctement). On parle alors de méta-heuristiques. Attention toutefois, car les algorithmes génétiques sont lents et leur usage ne se justifie que si l'application d'algorithmes plus efficaces n'est pas possible.

Nous avons défini le côté « algorithme » des algorithmes génétiques, mais qu'en est-il de la « génétique » ? On retrouve ici l'inspiration de la nature, mais d'un point de vue microscopique. La théorie de l'évolution, introduite par Charles Darwin en 1859 **[3]**, nous a apporté la sélection naturelle : un être vivant adapté à son milieu à plus de chance de survivre et donc, de se reproduire et de propager ses caractéristiques. Si l'on ajoute à cela la théorie de l'hérédité de Gregor Mendel **[4]** et la génétique des populations **[5]**, on aboutit à la théorie synthétique de

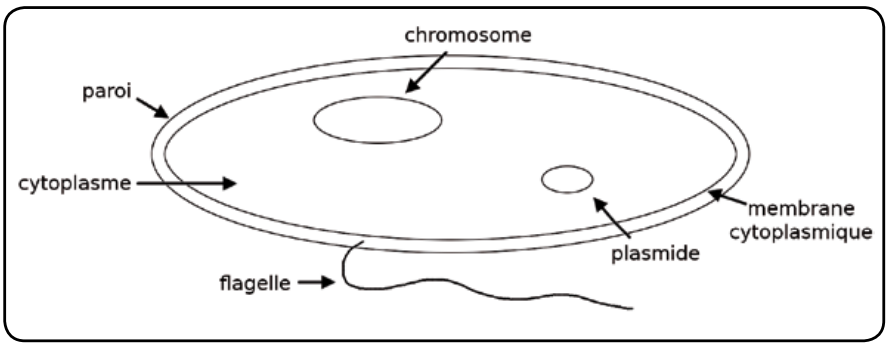


Fig. 1 : Schéma simplifié de la structure bactérienne

l'évolution **[6]**, base des algorithmes génétiques. Nous aurons donc besoin d'un minimum de connaissances en biologie et plus précisément en génétique pour comprendre ces algorithmes. Nous commencerons par une petite parenthèse expliquant les mécanismes de l'évolution avant d'aborder les algorithmes génétiques et de les mettre en pratique sur un exemple simple.

1 Un petit peu de biologie

Dans cette partie, nous aborderons les principes biologiques nécessaires à la compréhension des GA. Il y a deux façons d'envisager cet exposé : soit on se limite aux définitions des éléments utilisés, soit on s'intéresse un peu plus à la biologie en essayant de vraiment comprendre ce qui se passe. J'ai choisi la deuxième solution. Comme nous sommes sur un domaine très restreint, je tiens à vous rassurer, ça ne sera vraiment pas compliqué... Mais différent de ce que vous avez l'habitude de lire.

1.1 Les bactéries

Les algorithmes génétiques s'appuient sur un modèle particulier d'êtres vivants : les bactéries. De l'ordre du micromètre, les bactéries sont les plus petits organismes vivants (les virus, qui sont plus petits que les bactéries, ne peuvent pas être considérés comme vivants de par leur incapacité à se reproduire seuls). Une bactérie est une cellule entourée d'une membrane et contenant tous les éléments nécessaires à sa propre reproduction. Les populations de bactéries peuvent s'adapter aux variations de leur environnement, ou adopter en quelques générations seulement de tout nouveaux caractères.

D'un point de vue physique, l'architecture d'une bactérie est relativement simple comme le montre la figure 1. La cellule est tout d'abord entourée d'une paroi qui lui donne sa forme, sa résistance, et qui entoure une seconde enveloppe plus ou moins épaisse, la membrane cytoplasmique. Le (ou les) chromosome(s) et éventuellement le (ou les) plasmide(s) qui sont des sortes de petits chromosomes circulaires, porteurs de l'information génétique, se trouvent dans l'environnement délimité par la membrane cytoplasmique : le cytoplasme. Dans le cadre des GA, nous allons simplifier ce modèle en considérant que l'information génétique n'est portée que par un unique **chromosome**.

Ce chromosome est composé de molécules d'ADN, dont la fameuse structure en « double hélice » fut découverte en 1953 **[7]**. Elle est formée par une succession de petites unités appelées **nucléotides**, constituées de trois éléments : une molécule d'acide phosphorique, un sucre et une base organique. Dans le cas de l'ADN, le sucre est du désoxyribose, d'où son nom : Acide(phosphorique) Désoxyribo(se) Nucléique. Les **bases**, quant à elles, sont au nombre de quatre : l'adénine (**A**), la thymine (**T**), la cytosine (**C**) et la guanine (**G**). Ces chaînes ont trois propriétés essentielles :

- Elles sont hélicoïdales : elles s'enroulent autour d'un axe en formant une double hélice droite ;
- Elles sont antiparallèles : les deux brins de nucléotides sont parallèles, mais se lisent dans des directions opposées ;
- Elles sont complémentaires : les bases des deux brins ne pouvant

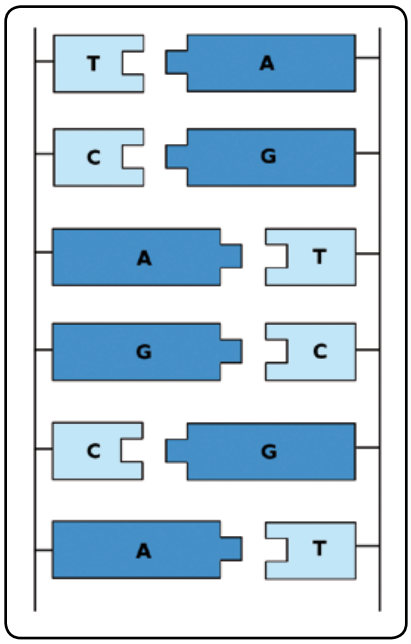


Fig. 2 : Les deux chaînes de nucléotides d'une molécule d'ADN. Les paires de bases constituant chaque barreau ont la même taille.

pas s'apparier n'importe comment, un peu comme pour les pièces d'un jeu de Lego, les deux bases C et T forment de petites briques, alors que les bases A et G forment de grandes briques. On ne peut associer qu'une petite brique avec une grande brique. Les seuls couples autorisés sont A-T et C-G (voir figure 2 page précédente).

L'ensemble des informations génétiques est contenu dans ce que l'on appelle le **génome**. La séquence **ACCCGAT** est un exemple d'information qui pourra ensuite être traduite sous forme de protéine pour constituer un être vivant et détermine ainsi son apparence ou **phénotype**.

1.2 Évolution

Intéressons-nous maintenant à l'évolution de cette information. La reproduction des bactéries prend la forme d'un procédé asexué où chacune grandit en taille, duplique son chromosome, puis se divise en deux bactéries identiques ou clones. Malgré ce type de reproduction clonal, qui laisserait supposer qu'une bactérie fille soit identique à sa « mère », l'information génétique est modifiée d'une génération à l'autre. On peut par exemple s'apercevoir qu'une bactérie pathogène devient résistante aux antibiotiques qui la combattent. Mais quel est ce miracle ? Plusieurs cas pouvant apparaître, nous n'aborderons ici que ceux qui seront utilisés dans les algorithmes génétiques.

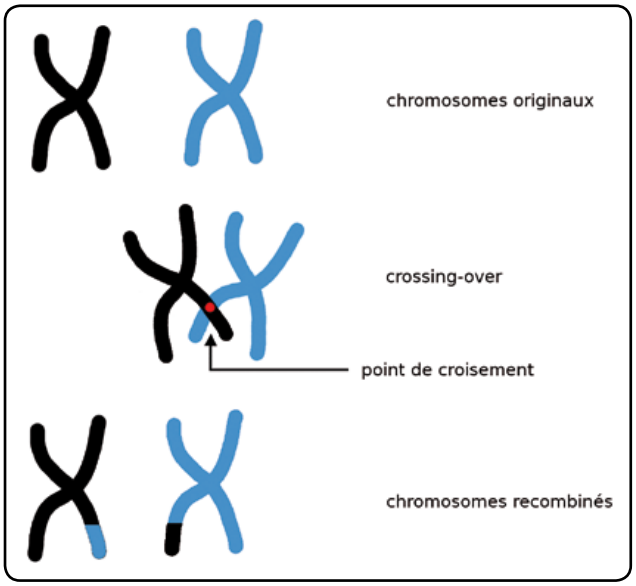


Fig. 3 : Recombinaison génétique par crossing-over

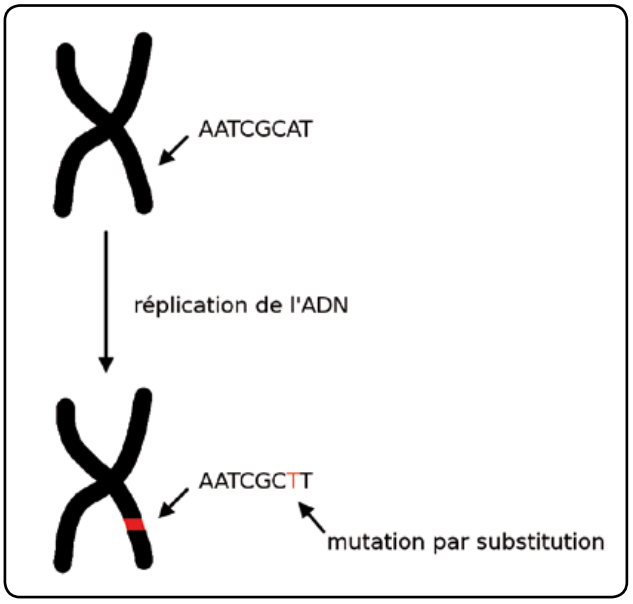


Fig. 4 : Recombinaison génétique par mutation (substitution)

1.2.1 Crossing-over

Le *crossing-over*, ou entrecroisement en bon français, a lieu lors de la duplication du chromosome : deux chromosomes parents se touchent en un point et échangent leurs informations à partir de ce point, comme le montre la figure 3.

1.2.2 Mutations

Lors de la réplication de l'ADN, il peut y avoir des ratés à cause d'accidents de copie des bases. On parle alors de mutations et on en distingue trois types :

- la **substitution** : une base est mal copiée ;
- la **délétion** : une base est oubliée ;
- l'**insertion** : une base est ajoutée.

Ces accidents de copie sont tout à fait analogues à une faute de frappe qui se produirait lors de la copie d'un texte. Prenons par exemple deux phrases : d'un côté « La **r**eine s'en est allée » et de l'autre « La **p**eine s'en est allée ». Ces deux phrases, bien que sémantiquement et syntaxiquement correctes, n'ont pas le même sens. Une seule lettre a été modifiée entre les deux et dans un cas le chagrin s'est évanoui, alors que dans l'autre la femme du roi est partie. Au niveau de l'ADN, les mutations peuvent avoir une influence sur la protéine qui sera produite et engendrer un phénotype différent et donc, un comportement différent en fonction de

ABONNEZ-VOUS !

CONSULTEZ L'ENSEMBLE DE NOS OFFRES SUR : boutique.ed-diamond.com !

11 Numéros de GNU/Linux Magazine

60€*

au lieu de 86,90 €* en kiosque

Économie 26,90€

Économisez plus de 30%*

* Sur le prix de vente unitaire France Métropolitaine

Nouveauté 2014
TOUS LES HORS-SÉRIES PASSENT EN GUIDE !

Un nouveau format avec une reliure de luxe pour ces Guides de référence de 128 pages à conserver dans votre bibliothèque !

Découvrez tous les Guides déjà parus sur

boutique.ed-diamond.com

*OFFRE VALABLE UNIQUEMENT EN FRANCE MÉTROPOLITAINE. Pour les tarifs hors France Métropolitaine, consultez notre site : boutique.ed-diamond.com

NOUVEAU ! Abonnez-vous (réabonnez-vous) en ligne sur :

boutique.ed-diamond.com

Vous pouvez ainsi :
- ➧ Avoir accès à votre suivi personnalisé d'abonnement
- ➧ Profiter des promos réservées à nos abonnés
- ➧ Vous réabonner facilement sans interruption d'abonnement

Pour plus d'informations, veuillez nous contacter via e-mail : cial@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous

Tournez SVP pour découvrir toutes les offres d'abonnement >>

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
E-mail :	

❑ Je souhaite recevoir les offres promotionnelles et newsletters des Éditions Diamond.

❑ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

GNU LINUX MAGAZINE / FRANCE

Édité par Les Éditions Diamond
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

Tournez SVP pour découvrir toutes les offres d'abonnement

ABONNEMENT GLMF

➡ **Tous les abonnements incluant GNU/Linux Magazine :**

offre LM

60€* au lieu de **86,90€**** en kiosque

Économie 26,90€

INCLUS : GNU/Linux Magazine (11nos)

offre LM+

115€* au lieu de **164,30€**** en kiosque

Économie 49,30€

INCLUS : GNU/Linux Magazine (11nos) + ses 6 Guides

NOUVEAUTÉ 2014
TOUS LES HORS-SÉRIES PASSENT EN **GUIDES !**

offre T

198€* au lieu de **277,10€**** en kiosque

Économie 79,10€

INCLUS : GNU/Linux Magazine (11nos), Open Silicium (4nos), MISC (6nos), Linux Pratique (6nos) et Linux Essentiel (6nos)

offre T+

299€* au lieu de **411,20€**** en kiosque

Économie 112,20€

INCLUS : GNU/Linux Magazine (11nos) + ses 6 Guides, Linux Pratique (6nos) + ses 3 Guides, MISC (6nos) + ses 2 Hors-Séries, Open Silicium (4nos) et Linux Essentiel (6nos)

* Tarifs France Métro. (F) ** Base tarifs kiosque zone France Métro (F)

NOUVEAU ! Abonnez-vous (réabonnez-vous) en ligne sur :

boutique.ed-diamond.com

Vous pouvez ainsi : ➡ Avoir accès à votre suivi personnalisé d'abonnement ➡ Profiter des promos réservées à nos abonnés ➡ Vous réabonner facilement sans interruption d'abonnement

Pour plus d'informations, veuillez nous contacter via e-mail : cial@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20

➡ Nos Tarifs s'entendent TTC et en euros

		F	OM1	OM2	E	RM
		France Métro.	Outre-Mer		Europe	Reste du Monde
LM	Abonnement GLMF	60 €	75 €	96 €	83 €	90 €
LM+	Abonnement GLMF + GLMF HS (6 Guides)	115 €	147 €	190 €	160 €	173 €
T	Abonnement GLMF + MISC + OS + LP + LE	198 €	253 €	325 €	276 €	300 €
T+	Abonnement GLMF + GLMF HS (6 Guides) + MISC + MISC HS + OS + LP + LP HS (3 Guides) + LE	299 €	382 €	491 €	415 €	448 €

• OM1 : Guadeloupe, Guyane française, Martinique, Réunion, St-Pierre-et-Miquelon, Mayotte

• OM2 : Nouvelle Calédonie, Polynésie française, Wallis et Futuna, Terres Australes et Antarctiques françaises

MA FORMULE D'ABONNEMENT :

Offre	Zone	Tarif
☐ LM		
☐ LM+		
☐ T		
☐ T+		

J'indique la somme due : (Total) ..... €

Exemple :
Je souhaite m'abonner à l'ensemble des magazines + tous les Hors-séries/Guides et je vis en Belgique.
Je coche donc l'offre **T+** (la totale avec tous les Hors-Séries/Guides), puis ma zone (E), le montant sera donc de 415 euros.

Je choisis de régler par :

☐ Chèque bancaire ou postal à l'ordre des Éditions Diamond (uniquement France et DOM TOM)

☐ Pour les règlements par virements, veuillez nous contacter via e-mail : cial@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20

Date et signature obligatoires

l'environnement. La sélection naturelle fera ensuite le reste pour déterminer si cette mutation a été profitable à l'organisme ou non. La figure 4 montre une mutation par substitution sur un chromosome.

Voilà, ce sont les seules connaissances biologiques dont nous aurons besoin. Nous pouvons maintenant passer à la partie plus théorique.

2 Un petit peu de théorie

Commençons par le principe général :

1. On crée une population au hasard : c'est la **population initiale**. Chaque individu possède un génome propre qui représente une solution possible au problème de départ. Par « solution possible », on entend plutôt proposition de solution, il ne s'agit pas d'une solution qui résout le problème (sinon il serait inutile de créer un algorithme...).
2. Jusqu'à ce qu'un critère d'arrêt soit vérifié (nombre de générations ou temps d'exécution maximum par exemple) :
   2.1. On sélectionne une partie de la population en fonction d'une note qu'on lui attribue. Les nouveaux individus sont donc globalement mieux adaptés à leur environnement.
   2.2. Les individus sélectionnés se reproduisent et donnent naissance à une nouvelle génération qui remplace la population initiale. C'est ici qu'interviennent les recombinaisons génétiques amenant de la diversité dans les solutions.

Plus formellement, soit S l'espace de recherche des solutions et une fonction d'adaptation `f`, souvent appelée *fonction de fitness*, qui fournit une évaluation de la performance des individus et participe donc à la sélection et à la reproduction des individus, `score_max` le meilleur score attribué par la fonction `f`, `max_it` le nombre maximal d'itérations, un algorithme génétique consiste à effectuer les étapes suivantes :

```
Construire la population initiale P0
i <- 0
Tant que f(P0) <> score_max et i < max_it Faire
        Mi <- (f(Pi), Pi)
        Mi <- selection(Mi)
        Pi+1 <- crossingover(Mi)
        Pi+1 <- mutation(Pi+1)
        i <- i +1
```

Il est évident ici qu'au plus il y aura de générations, au plus nous aurons des individus (solutions) adaptés à leur environnement (problème de départ). Mais comme l'écrivait Montesquieu, « le mieux est le mortel ennemi du bien »... Il faudra parfois être raisonnable et arrêter l'algorithme avec une bonne solution qui ne sera pas forcément la meilleure.

Les problèmes pour implémenter cet algorithme seront les suivants :

- Comment représenter l'information génétique ?
- Comment déterminer la population initiale ?
- Comment faire intervenir le mécanisme de sélection pour obtenir les parents de la génération suivante ?
- Comment effectuer la reproduction des individus en y intégrant les recombinaisons génétiques ?

Nous allons voir chacun de ces points.

2.1 Représentation de l'information génétique

Même si ce n'est pas obligatoire, le génotype peut utiliser un codage binaire et il est donc représenté par une chaîne de bits. Il faudra alors disposer d'un fonction capable d'exprimer le phénotype d'un individu comme le montre le tableau suivant :

Génotype	Phénotype
00	**
01	******
10	**********
11	**********

2.2. Calcul de la population initiale

Le choix de la population initiale est très important, car c'est de sa composition que dépendra la convergence plus ou moins rapide vers l'ensemble de solutions. Il faut choisir une population d'individus non homogène, qui soit répartie sur tout l'espace S. En utilisant un codage binaire, le plus simple consiste à générer de manière aléatoire uniforme cette population de départ. On assure ainsi une uniformité de répartition des génotypes... Mais pas forcément des phénotypes, si plusieurs gènes représentent la même donnée (`10` et `01` pourraient avoir pour même phénotype une barre composée de quatre étoiles).

2.3 Sélection

La sélection s'effectue sur la base de la fonction de fitness appliquée à la population courante (notée `Pi` dans l'algorithme). Il existe de nombreuses méthodes de sélection des individus, dont voici les principales :

- La sélection **par élitisme** : on ne conserve que les meilleurs individus au sens de la fonction de fitness ;
- La sélection **par roulette [8]** : on donne à chaque individu une probabilité d'être sélectionné proportionnelle à sa performance. En d'autres termes, on applique la fonction de fitness `f` à tous les individus, ce qui nous permet d'établir une hiérarchie ; puis, on calcule la probabilité de choisir un individu `I` en divisant `f(I)` par la somme des valeurs de `f` calculées sur la population courante. On sélectionne ensuite les individus à partir des probabilités calculées.
- La sélection **par rang** : méthode similaire à la sélection par roulette, mais la probabilité est calculée sur le rang de l'individu.
- La sélection **par tournoi [9]** : on crée arbitrairement des groupes de `n` individus (`n >= 2`) et on applique une autre méthode de sélection (par exemple l'une des trois précédentes) pour sélectionner les individus les plus performants.

2.4 Reproduction

La reproduction est assez simple. En fonction d'un pourcentage qu'il faudra fixer arbitrairement, on va appliquer des recombinaisons génétiques par crossing-over et mutation. Si l'on considère que la représentation des génomes se fait de manière binaire, un crossing-over s'obtiendra de la façon suivante :

Parent 1	1101 0011
Parent 2	0100 1010
Enfant 1	1101 1010
Enfant 2	0100 0011

La mutation, elle, se fera de manière encore plus rapide en sélectionnant une base de manière aléatoire :

Parent	1101 0011
Enfant	1111 0011

2.5 Paramétrage simple

Vous vous en êtes rendu compte : certains paramètres devront être fixés de manière totalement arbitraire. Malheureusement, il n'existe pas de paramétrage « universel » qui donnerait de bons résultats avec n'importe quel problème. Voici des valeurs qui peuvent être utilisées comme point de départ avant de procéder à un éventuel réajustement :

Population	entre 50 et 100 individus
Taux de crossing-over	entre 70% et 95%
Taux de mutations	entre 0,001% et 0,1%

Nous avons globalement vu toute la théorie des algorithmes génétiques. Il nous reste à les appliquer à un exemple simple pour bien en comprendre le fonctionnement.

3 Un algorithme génétique simple

Dans cette partie, le code résolvant le problème sera donné en Python 3. Pourquoi encore ce langage ? La réponse tient dans le fait qu'il est très simple à comprendre et que, même ceux d'entre vous qui ne le connaissent pas pourront aisément adapter le code dans leur langage favori. J'ai choisi cette solution plutôt que de ne présenter qu'un algorithme, de manière à ce que vous puissiez vraiment tester directement le code et voir évoluer les résultats.

3.1 Le problème

Nous allons essayer de résoudre un problème simple : étant donnés les chiffres de 0 à 9 et les opérateurs numériques `+`, `-`, `/` et `%` (reste de la division entière), trouver une expression qui représentera une valeur cible. La multiplication n'est pas utilisée pour éviter d'obtenir des nombres trop grands, mais dans l'absolu, si vous le souhaitez, vous pouvez remplacer le reste de la division entière par la multiplication.

Cherchons par exemple des expressions qui valent 1234. Pour réduire l'espace de recherche, nous limiterons les expressions à 12 symboles maximum. Voici quelques solutions possibles :

- `1234`
- `1200 + 30 + 4`
- `2460 / 2 + 4`
- `...`

Le problème est posé, voyons comment le représenter.

3.2 Le codage

Nous avons vu que le codage binaire était ce qu'il y a de plus simple pour représenter le génotype. Dans le cas de notre problème, nous devons pouvoir représenter 14 symboles (les dix chiffres et les quatre opérateurs). Si nous comptons le fait que nous partirons de 0, il faut que nous soyons capables de conserver en binaire des valeurs allant de 0 à 13 (ce qui fait bien 14 symboles). Or, en binaire, 13 vaut **1101** (1x23 + 1x22 + 0x21 + 1x20). Il nous faut donc 4 bits pour représenter chacun de nos symboles :

```
0000 0
0001 1
0010 2
0011 3
0100 4
0101 5
0110 6
0111 7
1000 8
1001 9
1010 +
1011 -
1100 /
1101 %
1110 vide
1111 vide
```

Nous avons fixé le nombre de symboles d'une équation à 12. Ce choix n'était pas dû au hasard : chaque symbole est codé sur 4 bits, donc 12 symboles tiennent sur 12x4 = 48 bits, soit 6 octets. Les chromosomes (qui représentent une solution au problème) seront donc codés sur 6 octets (chacun d'eux contenant deux gènes). Pour la lisibilité du code et par souci de simplification, nous utiliserons ici des chaînes de caractères pour coder les gènes. J'attire votre attention sur la consommation mémoire que cela va engendrer : un caractère est codé sur un ou plusieurs octet(s) suivant les langages et donc, pour chaque information nous consommons au minimum 50% de mémoire en plus. De même, ce type d'algorithme se prête particulièrement bien à la programmation orientée objet (Gene, Chromosome, Population), mais la résolution sera présentée sous forme de programmation impérative, de manière à ne pas polluer la compréhension de l'algorithme par des considérations techniques.

Pour créer notre population initiale **P0**, nous allons devoir générer de manière aléatoire des chromosomes, puis les enregistrer dans une liste (tableau).

L'algorithme est très simple ; sachant que **nombre_genes** contient le nombre de gènes, **taille_gene** représente la taille d'un gène en bits, et la fonction **random()** renvoie **0** ou **1** de manière aléatoire :

```
chromosome <- ""
Pour i variant de 1 à nombre_genes Faire
    Pour j variant de 1 à taille_gene Faire
        chromosome <- chromosome + random()
```

Le code suivant réalise cette opération et affiche les chromosomes de la population générée.

```
01: # -*- coding:utf-8 -*-
02:
03: import random
04:
05: def chromosome_initial(nb_genes=12, taille_gene=4):
06:     chromosome = ""
07:     for i in range(nb_genes):
08:     for j in range(taille_gene):
09:         chromosome += str(random.randint(0, 1))
10:     return chromosome
11:
12:
13: def population_initiale(individus=100):
14:     population = []
15:     for i in range(individus):
16:         population.append(chromosome_initial())
17:     return population
18:
19:
20: if __name__ == "__main__":
21:     P0 = population_initiale()
22:     for chromosome in P0:
23:         print(chromosome)
```

Le paramétrage par défaut des fonctions (lignes 5 et 13) permet de traiter notre problème, mais permet également d'apporter des modifications par la suite (affinage des paramètres ou récupération du code). Nous obtenons des lignes semblables à la suivante :

```
01111101001100101011010100101010001001110000000
```

Il y a bien 48 caractères, donc visiblement, notre programme fonctionne correctement... Mais ce n'est pas très lisible ! Je vous rappelle que ce chromosome est censé représenter une solution potentielle à notre problème. Difficile de dire ici si c'est le cas ou non. Il nous faut donc une fonction de conversion capable de lire gène par gène un chromosome et de nous renvoyer sa représentation symbolique. Considérons que nous stockons les symboles dans un tableau dont les indices correspondent au codage binaire :

```
codage = ("0", "1", ..., "9", "+", "-", "/", "%", "_", "_")
```

Si nous voulons savoir à quoi correspond le gène **1101**, il suffit de convertir la valeur en décimal (13) et de regarder la valeur stockée dans le tableau à cet indice (**codage[13]** contient **%**). Pour que ce système fonctionne, il ne faut surtout pas oublier que les valeurs **1110** et **1111** ne correspondent à rien, mais que des valeurs doivent leur être associées de manière à ne pas obtenir d'erreur en recherchant un élément inexistant. Ici, j'ai choisi d'indiquer par un **_** qu'il n'y avait pas de symbole associé (on aurait aussi pu choisir une chaîne vide pour ne pas générer d'erreur et autoriser des solutions plus petites en termes de nombre de symboles).

Voyons maintenant comment coder cela :

```
...
20: def gene_to_symbole(gene, codage):
21:     return codage[int(gene, 2)]
22:
23:
24: def chromosome_to_expression(chromosome, codage, taille_gene=4):
25:     expression = ""
26:     for num_gene in range(int(len(chromosome) / taille_gene - 1)):
27:         expression += \
28:             gene_to_symbole(chromosome[num_gene * 4:num_gene * 4 + 4], codage)
29:     return expression
30:
31:
32: if __name__ == "__main__":
33:     codage = tuple(map(str, range(10))) + ("+", "-", "/", "%", "_", "_")
34:     P0 = population_initiale()
35:     for chromosome in P0:
36:         print(chromosome)
37:         print(chromosome_to_expression(chromosome, codage))
```

La fonction **gene_to_symbole()** des lignes 20 et 21 effectue la conversion de binaire en décimal avec **int(gene, 2)**, puis nous retourne le symbole associé au gène dans le tableau de codage. La fonction **chromosome_to_expression()** des lignes 24 à 29 parcourt un chromosome qui lui est passé en paramètre, lit des blocs de quatre caractères (qui représentent les bits) et les transforme en symbole. Au fur et à mesure, elle construit l'expression complète portée par le chromosome. Dans le programme principal, j'ai utilisé une astuce en ligne 33 pour ne pas avoir à écrire à la main tous les éléments du tuple contenant la traduction des gènes : **map(str, range(10))** applique la fonction **str()** à tous les éléments de la liste **range(10)**, soit

[0, 1, ..., 9]. Partant d'une liste d'entiers, on obtient une liste de chaînes de caractères que l'on convertit ensuite en tuple (en Python, les tuples sont des tableaux non modifiables).

On voit maintenant immédiatement la correspondance entre notre chromosome et l'expression qu'il représente :

```
01111101001100101011010100101010001001110000000
7%32-52+270
```

Nous savons coder et décoder les données contenues dans les chromosomes. Attaquons-nous maintenant à la sélection des individus et à leur reproduction.

3.3 La résolution

3.3.1 Évaluation

Ici, nous allons devoir concevoir la fonction de fitness. Qu'est-ce qui peut refléter le fait que nous soyons plus proches ou éloignés de la solution (obtenir 1234) ? Soit **Sol** la solution proposée par un chromosome. Son évaluation va nous donner un résultat : si c'est 1234, nous avons trouvé le résultat et plus la valeur est éloignée de 1234, plus la solution est mauvaise. En analysant le résultat de **|1234 - Sol|**, plus la valeur est proche de **0**, meilleure est la solution. Certaines solutions vont générer des erreurs (présence de **_** ou division par **0**). Nous fixerons alors arbitrairement une valeur suffisamment grande pour indiquer que la solution n'est pas bonne.

En Python, tout cela donne :

```
33: def fitness(chromosome, codage, taille_gene=4, resultat_
final=1234):
34:     expression = chromosome_to_expression(chromosome, codage,
taille_gene)
35:     try:
36:         resultat = abs(resultat_final - eval(expression))
37:     except:
38:         resultat = sys.maxsize
39:     return resultat
```

En ligne 36, nous utilisons la fonction **eval()** pour calculer le résultat de l'expression contenue dans le chromosome. En cas d'erreur, nous récupérons l'exception et affectons à **resultat** la plus grande valeur qu'un entier puisse contenir (ligne 38). Bien sûr, il ne faut pas oublier d'importer le module **sys** en début de programme.

Si nous parcourons la population, nous pouvons associer à chaque individu un score :

```
42: def evaluer_population(population, codage, taille_gene=4, resultat_
final=1234):
43:     for indice, chromosome in enumerate(population):
44:         population[indice] = (fitness(chromosome, codage, taille_gene,
45:                                       resultat_final), chromosome)
46:     population.sort()
```

Comme nous travaillons sur une liste, il est inutile de renvoyer une valeur, car elle est directement modifiée en mémoire. Nous modifions chaque élément de la liste **population** en associant à chaque chromosome le résultat de la fonction **fitness()**. La liste de population est alors du type **[(fitness(c_1), c_1), ... (fitness(c_n), c_n)]** où **c_i** représente le ième chromosome (**i** compris entre **1** et **n**). La dernière instruction de la ligne 46 permet de trier les individus de la population de manière à ce que ceux possédant le score de fitness le plus faible soient les premiers. Cette nouvelle représentation va nous amener à modifier la génération de la population initiale et du coup, la fonction d'évaluation :

```
...
14: def population_initiale(individus=100):
15:     population = []
16:     for i in range(individus):
17:         population.append((None, chromosome_initial()))
18:     return population
...
42: def evaluer_population(population, codage, taille_gene=4, resultat_
final=1234):
43:     for indice, (value, chromosome) in enumerate(population):
44:         population[indice] = (fitness(chromosome, codage, taille_gene,
45:                                       resultat_final), chromosome)
46:     population.sort()
```

3.3.2 Sélection

Pour la sélection, nous adopterons la technique la plus simple avec une sélection élitiste. Par défaut, nous conserverons la moitié de la population contentant les meilleurs candidats à une solution.

```
49: def selection(population, pourcentage=0.5):
50:     return population[0:int(len(population) * pourcentage)]
```

Pour voir les individus sélectionnés dans la population initiale :

```
52: if __name__ == "__main__":
...
54:     P0 = population_initiale()
...
58:     evaluer_population(P0, codage)
59:     P0 = selection(P0)
60:     print(P0)
```

3.3.3 Reproduction

Les individus étant sélectionnés, il faut qu'ils puissent se reproduire. Pour cela, nous devons créer des fonctions qui permettent de mettre en place les recombinaisons génétiques.

Commençons par le crossing-over : il nous faut deux individus, les parents, qui vont donner naissance à deux autres individus, les enfants.

```
54: def crossingover(parent_1, parent_2):
55:     chromosome_1 = parent_1[1]
56:     chromosome_2 = parent_2[1]
57:     point_croisement = random.randint(1, len(chromosome_1) - 2)
58:     enfant_1 = (None, chromosome_1[0:point_croisement] +
59:                 chromosome_2[point_croisement:])
60:     enfant_2 = (None, chromosome_2[0:point_croisement] +
61:                 chromosome_1[point_croisement:])
62:     return [enfant_1, enfant_2]
```

On détermine de manière aléatoire un point de croisement en ligne 54. Ensuite, il faut découper le chromosome de chaque parent suivant ce point et les coller en les mélangeant (lignes 55 à 58). Ici, le crossing-over est volontairement simple, tel qu'expliqué dans la partie traitant de la biologie... Mais dans la nature, il existe des crossing-over avec points de croisement multiples, que nous aurions pu également implémenter ici, de manière à brasser encore plus l'information génétique. Cela entraîne l'apparition d'un paramétrage supplémentaire avec le taux de crossing-over à un point de croisement, deux ou trois, sachant bien sûr que plus le nombre de points de croisement augmente, plus la probabilité d'apparition est faible. Nous restons donc ici sur le cas le plus simple.

Pour les mutations, nous appliquerons des substitutions : de manière aléatoire, nous sélectionnons une base (un bit du chromosome) et nous l'inversons (s'il vaut **1** il passe à **0** et inversement).

```
65: def mutation(individu):
66:     chromosome = individu[1]
67:     base = random.randint(0, len(chromosome) - 1)
68:     mute = "1" if chromosome[base] == "0" else "1"
69:     chromosome = chromosome[0:base] + mute + chromosome[base + 1:]
70:     return (None, chromosome)
```

Nous pouvons créer de la « nouveauté » à partir d'une population existante. Il faut maintenant produire les nouvelles générations.

3.3.4 Production des nouvelles générations

Pour produire une nouvelle génération, nous allons partir d'une population à laquelle nous allons appliquer notre fonction de sélection, puis reproduire les « meilleurs » individus avec crossing-over et mutations suivant certains taux. Il s'agit clairement d'eugénisme et un tel concept sorti de son contexte d'algorithme informatique est assez effrayant.

Voici la fonction de production d'une génération **Pi+1** à partir d'une génération **Pi**.

```
073: def nouvelle_generation(population, codage, taille_gene=4, resultat_final=1234,
074:                         taux_selection=0.5, taux_crossingover=0.9,
075:                         taux_mutation=0.001):
076:     evaluer_population(population, codage, taille_gene, resultat_final)
077:     population = selection(population, taux_selection)
078:     meilleur = population[0][0]
079:     meilleur_eval = chromosome_to_expression(population[0][1], codage)
080:     # Mélange de la population sélectionnée
081:     random.shuffle(population)
082:     nouvelle_population = []
083:     # Crossing-over
084:     indice = 0
085:     taille_population = len(population)
086:     while (indice + 1) < taille_population:
087:         parent_1 = population[indice]
088:         parent_2 = population[indice + 1]
089:         indice += 2
090:         if random.random() <= taux_crossingover:
091:             for i in range(int(1 / taux_selection)):
092:                 nouvelle_population += crossingover(parent_1, parent_2)
093:         else:
094:             for i in range(int(1 / taux_selection)):
095:                 nouvelle_population.append(parent_1)
096:                 nouvelle_population.append(parent_2)
097:     # Mutation
098:     for individu in nouvelle_population:
099:         if random.random() <= taux_mutation:
100:             nouvelle_population[indice] = mutation(individu)
101:     return (meilleur, meilleur_eval, nouvelle_population)
```

Nous utilisons ici toutes les fonctions définies précédemment. Ce qu'il faut surtout noter, ce sont les boucles des lignes 91 et 94 qui permettent de s'assurer que la population compte toujours le même nombre d'individus : une diminution de ce nombre entraînant une chute de la diversité génétique, notre algorithme s'arrêterait rapidement sans solution. Le crossing-over et la

mutation sont soumis à des taux d'apparition (lignes 90 et 99), ce qui permet d'effectuer des réglages en fonction des résultats observés.

3.3.5 Recherche d'une solution

Le programme principal va consister à créer de nouvelles générations jusqu'à obtenir un résultat à notre problème. Comme nous savons qu'il est possible que l'on ne trouve qu'une solution approchée, nous arrêterons notre programme pour le relancer avec une nouvelle population initiale à partir de la 3000ème génération.

```
104: if __name__ == "__main__":
105:     codage = tuple(map(str, range(10))) + ("+", "-", "/", "%", "_", "_")
106:     n_generation = 0
107:     P0 = population_initiale()
108:     while True:
109:         meilleur_score, meilleur_evaluation, P1 = nouvelle_generation(P0,
110:                                                                       codage)
111:         P0 = P1
112:         n_generation += 1
113:         print("Génération: {} Meilleur score: {}".format(
114:             n_generation, meilleur_score))
115:         print("    {}".format(meilleur_evaluation))
116:         if meilleur_score == 0:
117:             break
118:         if n_generation == 3000:
119:             print("Il vaut mieux repartir sur une nouvelle population...")
120:             evaluer_population(P1, codage)
121:             print(P1)
122:             P0 = population_initiale()
123:             n_generation = 0
124:             print("Appuyez sur <Return> pour continuer")
125:             input()
```

Lors du lancement du programme, si vous êtes chanceux, vous devriez obtenir rapidement une solution :

```
Génération: 1 Meilleur score: 1164
    73+4-6-1%80
...
Génération: 26 Meilleur score: 0
    374-37++897
```

Il arrive que le programme n'arrive plus à générer de nouvelles solutions. C'est là que nous arrêtons les calculs pour recommencer avec une nouvelle population initiale. Si vous observez la population à cet instant, vous vous apercevrez qu'il n'y a plus de diversité génétique. Une solution pourrait consister à en réintroduire en augmentant le taux de mutations. Vous pouvez également essayer d'ajuster les paramètres de l'algorithme, mais attention : un taux de mutations trop élevé ferait perdre le bénéfice de la sélection !

Conclusion

Nous avons pu voir une catégorie d'algorithmes bien particuliers, puisque recherchant une solution à tâtons et s'appuyant sur un modèle biologique. Lors de leur utilisation, les problèmes principaux viendront du choix de la fonction de fitness, de la méthode de sélection et du paramétrage employé : chacun de ces éléments peut avoir un impact très important sur la rapidité de l'algorithme à converger vers une solution. De plus, ces réglages ne sont pas généraux et devront donc être effectués pour chaque problème. Dans la dernière partie, j'ai tenu à vous présenter un code mettant en œuvre les algorithmes génétiques sur un problème simple. Ce code n'est pas optimal, mais il a, je l'espère, le mérite d'être clair. Si vous souhaitez vous entraîner sur un autre problème relativement simple, vous pouvez tester la découverte d'un chemin dans un labyrinthe. Sinon, il ne vous reste plus qu'à trouver un problème sur lequel appliquer les algorithmes génétiques... ■

Références

[1] Eiben A. et J. Smith, « Introduction to evolutionnary computing », éd. Springer, 2003.

[2] Holland J., « Adaptation in Natural and Artificial Systems », University of Michigan Press, 1975.

[3] Darwin C., « Origin of species », 1859.

[4] Mendel G., « Experiments in plant hybridization », 1865.

[5] Haldane J., « The causes of evolution », 1932.

[6] Mayr E., « Systematics and the origin of species, from the viewpoint of a zoologist », Harvard University Press, 1942.

[7] Watson J. et Crick F., « Molecular structure of nucleic acids. A structure for desoxyribose nucleic acid », Nature, vol. 171, 1953, p. 737 et 738.

[8] Backer J., « Reducing bias and inefficiency in the selection algorithm », Actes de Second International Conference on Genetic Algorithms and their applications, 1987, p. 14 à 21.

[9] Miller B. et Goldberg D., « Genetic Algorithms, Tournament Selection, and the effects of noise », Evolutionary Computation, vol. 4, p. 113 à 131.

COLLECTD ET PERFWATCHER : CONFIGURATION AVANCÉE

*par **Yves Mettier** [Auteur de C en action (2ième édition) paru chez ENI auteur du guide de survie Langage C paru chez Pearson]*

Le mois dernier, nous avons installé Collectd : vous visualisez les données avec PerfWatcher et vos agents remplissent les fichiers RRD du serveur. Passons maintenant à la configuration avancée de votre installation en étudiant quelques problématiques.

1 Intervalle de collecte : 10 ou 60 secondes ?

L'intervalle de collecte est de 60 secondes par défaut. Si je le mets à 10 secondes, je vais multiplier par 6 l'espace disque nécessaire au stockage des informations, vrai ? Réponse : partiellement vrai. Si vous pensez aux notifications issues du module `top`, elles seront 6 fois plus nombreuses. Mais les fichiers `top process` sont compressés, et compresser six fois des choses similaires donne un résultat plus petit que prévu. Par exemple, 6 fichiers de ce genre de 12 Ko ont été décompressés, concaténés, puis recompressés. Le résultat n'est pas de 72 Ko, mais de 53 Ko. Cela fait x 4,41, et non x 6. Pour les données de `sysconfig`, comme elles ne sont envoyées qu'une seule fois par jour, elle ne sont pas sensibles à l'intervalle.

Intéressons-nous maintenant au plus spectaculaire : les fichiers RRD. Ceux-ci sont optimisés pour un stockage par jour, semaine, mois et année. En d'autres termes, nous avons un certain nombre de « rangées » pour un jour, un autre nombre pour une semaine, et ainsi de suite. Par défaut, avec Collectd, ce nombre est approximativement de 1200 (c'est le résultat d'un calcul). L'intervalle est donc non pas 60 secondes (ou 10), mais la période (jour, semaine, mois, année) divisée par le nombre de rangées. Sur la durée, cet intervalle perd de son importance, car sur le long terme, on ne souhaite plus voir le détail d'un pic, mais plutôt des tendances. L'intervalle n'a donc aucun impact ici. L'intervalle a un impact sur la petite durée qui nous sépare de maintenant à il y a quelques heures. En effet, il existe un certain nombre de rangées dont la durée est calculée en fonction de l'intervalle et non le contraire. C'est ici que vous allez indiquer si ces rangées contiennent des valeurs correspondant à 10 secondes ou à 1 minute. Si vous choisissez 10 secondes, vous aurez approximativement 3h20 échantillonnées à 10 secondes. Si vous préférez 60 secondes, vous aurez plutôt presque 20h (6 fois plus, soit 6 fois 3h20). Une fois cette durée écoulée, vous passez à l'échantillonnage jour, où chaque rangée représente 60 secondes. Quatre choses sont donc à savoir si vous choisissez un intervalle de 10 secondes :

- L'intérêt de l'intervalle à 10 secondes ne dure qu'environ 3h20 dans un fichier RRD. Ensuite, peu importe l'intervalle que vous avez choisi, RRDTool passe à l'échantillonnage prédéfini ;
- L'intervalle de 10 ou 60 secondes impacte la durée d'un certain nombre de rangées, ce nombre étant à peu près fixe, la taille du fichier RRD est sensiblement la même quel que soit votre échantillonnage ;

- Une fois l'intervalle choisi, il impacte le stockage dans les fichiers RRD par la durée de certaines rangées, puis par effet boule de neige toutes les autres, les unes se déversant dans les autres en passant d'une durée prédéfinie à une autre (jour, semaine, mois, année). Modifier l'intervalle corrompt les données en changeant la date à laquelle elles doivent représenter l'état du système. Il est donc impossible de changer l'intervalle dans un fichier RRD, sauf en supprimant ce fichier RRD et en perdant donc son historique ;
- Il est possible de ne configurer qu'un seul plugin à 10 secondes et garder les autres à 60 secondes. C'est sûrement le plus important à savoir.

Pour paramétrer un seul plugin avec un intervalle différent des autres, vous pouvez charger le plugin ainsi :

```
<LoadPlugin cpu
    Interval 10>

</LoadPlugin>
```

Au lieu d'un simple **LoadPlugin cpu**, ceci chargera votre plugin en paramétrant un intervalle spécifique de 10 secondes et en conservant l'intervalle par défaut pour tous les autres plugins.

Que choisir ? 10 ou 60 ? Il faut en tout cas effectuer ce choix à l'installation de l'agent, avant son premier lancement. Après, il est trop tard. Au niveau des fichiers RRD, 10 semble plus intéressant vu le non-prix à payer. Mais il faudra compter avec un trafic réseau 6 fois plus élevé, des collectes 6 fois plus fréquentes sur le serveur. Et des fichiers **top process** environ 4 fois plus volumineux si vous n'avez pas pensé à configurer le plugin **top** à 60 secondes.

2 Configuration avancée des fichiers RRD

2.1 RRD, RRA et calculs

10 secondes, voire 1 seconde, c'est ce que vous avez choisi. Mais au bout de quelques heures, la granularité passe à 60 secondes. Pourquoi ? Quant aux données d'il y a quelques mois, elle est devenue de 7 minutes !

Pour comprendre ce qu'il se passe, nous devons d'abord approfondir notre connaissance des fichiers RRD. Ceux-ci sont des bases de données tournantes (*Round Robin Databases*). En fait, un fichier RRD peut contenir plusieurs de ces bases. Chacune d'elle s'appelle RRA, ou *Round Robin Archive*. Voyons l'une d'elle :

```
$ rrdtool info ixion/load/load.rrd
filename = "load.rrd"
rrd_version = "0003"
step = 10
[...]
rra[0].cf = "AVERAGE"
rra[0].rows = 1200
rra[0].cur_row = 112
rra[0].pdp_per_row = 1
[...]
rra[1].cf = "MIN"
rra[1].rows = 1200
rra[1].cur_row = 817
rra[1].pdp_per_row = 1
[...]
rra[2].cf = "MAX"
rra[2].rows = 1200
rra[2].cur_row = 924
rra[2].pdp_per_row = 1
[...]
rra[3].cf = "AVERAGE"
rra[3].rows = 1235
rra[3].cur_row = 180
rra[3].pdp_per_row = 7
[...]
```

Notez le premier paramètre intéressant, **step**, qui vaut ici 60 ; c'est l'intervalle.

Puis, vous notez une série de RRA. Nous avons gardé ici les caractéristiques que nous allons discuter.

Reprenons : un RRA est une base de données tournante, contenant un certain nombre d'emplacements pour stocker les données. Il s'agit du paramètre **row**. Dans le premier RRA, il est donc possible de stocker 1200 données. Une donnée est constituée d'un agrégat de **pdp_per_row** métriques. Cela représente les données reçues pendant **pdp_per_row** fois l'intervalle **step**. Ainsi, le RRA 0 stocke ici 1 donnée pour 60 secondes et le RRA 3 stocke 1 donnée pour 420 secondes. On peut donc calculer la durée d'enregistrement qu'un RRA peut stocker :

durée = **row** x **pdp_per_row** x **step**

Le RRA 0 peut donc stocker 1200 x 1 x 60 = 1200 minutes = 20 heures.

Le RRA 3 peut stocker 1235 x 7 x 60 = 518700 secondes = approximativement 6 jours.

Le paramètre **cf** indique la façon d'effectuer l'agrégat. Il peut s'agir de prendre soit la valeur minimale des données reçues (**MIN**), maximale (**MAX**) ou moyenne (**AVERAGE**).

Par défaut, Collectd crée plusieurs RRA en calculant **pdp_per_row** pour que **row** soit approximativement de 1200 et que la durée (**RRATimespan**) soit de :

- 3600 (1 heure) ;
- 86400 (1 jour) ;
- 604800 (1 semaine) ;
- 2678400 (1 mois) ;
- 31622400 (1 an).

Pour chaque durée, 3 RRA sont créés : **MIN**, **MAX** et **AVERAGE**.

Cela nous fait donc 15 RRA dans un fichier RRD par défaut.

Le calcul est effectué ainsi :

pdp_per_row = **durée** / (**1200** x **step**)

Le résultat est un entier. Puis **row**, valant ci-dessus 1200, est recalculé en tenant compte du nouveau **pdp_per_row** :

row = **durée** / (**pdp_per_row** x **step**)

Le résultat **row** est également un entier. C'est pour cela qu'il vaut approximativement 1200, mais n'y est pas toujours égal.

Il existe cependant un cas particulier : pour les premiers RRA de durée d'1 heure, **pdp_per_row** est forcé à 1 et **row** à 1200. C'est donc la durée qui est calculée et elle ne vaut donc non pas 1 heure, mais 1200 x 1 x 60, soit 20 heures comme nous l'avions calculé ci-dessus.

Que retenir de ces calculs ? Que si vous modifiez l'intervalle, vous impactez la durée des premiers RRA et la granularité des autres RRA.

Deux derniers calculs :

- Avec le RRA d'un an : **pdp_per_row** = 31622400 / (1200 x 60) = 439 secondes = environ 7 minutes. Il s'agit de la granularité à partir du 2e mois.
- Avec le RRA dit d'une heure, si votre intervalle est de 1 seconde : **durée** = 1200 x 1 x 1 = 1200 secondes = 20 minutes. Il s'agit de la durée pendant laquelle vous allez profiter de la granularité de 1 seconde ! Après ces 20 minutes, vous changez de RRA et revenez à une granularité de 1 minute pour le reste des 24h (nous vous laissons effectuer le calcul).

Remarque : vu les calculs précédents, vous comprenez maintenant pourquoi modifier l'intervalle **step** corrompt votre fichier RRD. La durée représentée par un **row** n'est pas stockée dans le RRA, mais est calculée en fonction de **step**. Modifiez le **step** et vous modifiez l'échelle du temps de vos RRA.

2.2 Personnalisation : choisir les paramètres des RRA

À quoi bon une granularité de 1 seconde si vous ne pouvez pas en profiter plus de 20 minutes ? Il existe bien quelques paramètres pour le plugin **rrdtool** (et son homologue **rrdcached**) comme **RRARows** et **RRATimespan**, mais vous n'arriverez pas à définir une série de RRA acceptables.

Collectd-pw apporte un patch qui vous autorise à aller plus loin. Un premier paramètre, historique, mais conservé pour sa simplicité (et pour la compatibilité avec les anciennes versions) est **RRA**. Il vous permet d'indiquer les méthodes d'agrégation (paramètre **cf**) que vous souhaitez. Ainsi, si vous ne voulez que la moyenne, indiquez ceci :

```
<Plugin rrdcached>
[...]
    RRA AVERAGE
[...]
</Plugin>
```

Ce patch a été ensuite amélioré pour définir plus précisément chaque RRA. Sa syntaxe est :

```
RRADef <durée> <pdp_per_row> <granularité> <cf> <xff>
```

Nous n'aborderons pas le paramètre **xff** ici. Sachez seulement que vous pouvez également le configurer si vous en avez besoin.

Le premier paramètre, **durée**, est le même que celui que vous indiquez si vous utilisez **RRATimespan**. Il s'agit de la durée en secondes du RRA.

Le second paramètre, **pdp_per_row**, a été discuté ci-dessus. Si vous mettez 0, la valeur par défaut est utilisée.

Le troisième paramètre, la **granularité**, qui correspond à **durée** / **rows**, permet de calculer **rows** (**rows** = **durée** / **granularité**), mais aussi **pdp_per_row** si vous avez laissé la valeur par défaut. Vous pouvez déduire la formule : **pdp_per_row** = max(**granularité** / **step**, 1). Notez que si **pdp_per_row** n'est pas nul, **rows** est calculé sans tenir compte du paramètre **granularité**.

Enfin, **cf** indique la méthode d'agrégation. Vous pouvez en mettre plusieurs sur la même ligne.

Appliquons **RRADef** à un cas pratique : pouvoir disposer des métriques pendant 3 jours à la meilleure précision (donc min = max = average : nous n'en gardons qu'un), puis pendant 1 semaine avec un intervalle d'une minute, avec min, max et moyenne, et enfin pendant 1 an avec un intervalle d'une minute également, mais seulement la moyenne pour économiser l'espace disque.

```
RRADef 259200    1   0 AVERAGE
RRADef 604800    0  60 AVERAGE MIN MAX
RRADef 31622400  0  60 AVERAGE
```

Parrot® | Un monde d'innovation au cœur de Paris

GREAT PLACE TO WORK Best Workplaces 2013 France

Venez postuler sur
recrute.parrot.com

Avec ce paramétrage, nous obtenons 5 RRA (au lieu des 15 précédents), mais avec beaucoup plus de **rows**. La taille n'en est donc pas forcément réduite.

3 Configuration d'un Collectd proxy

La configuration d'un Collectd proxy est très simple. Vous devez éditer le fichier de configuration de l'agent Collectd ayant vocation à devenir proxy, trouver le module **network**, et y ajouter les lignes suivantes (celles d'un serveur) :

```
LoadPlugin network
<Plugin network>
[...]
    Listen "0.0.0.0" "25826"
    Forward true
</Plugin>
```

Ainsi, votre agent devient également un serveur, et sait transmettre ce qu'il reçoit aux serveurs à qui il transmet déjà en tant qu'agent.

Un danger avec les proxies est le risque de créer des boucles. Collectd est normalement protégé contre les boucles. En compensation, il ne transmet pas les notifications pour le moment. Aussi, un agent proxy est forcément un Collectd-pw qui transmet également les notifications. Le prix à payer était de retirer cette protection contre les boucles. C'est maintenant à vous d'y veiller. Sinon, ... bouM !

4 Renommer un serveur

Renommer un serveur sur Collectd et PerfWatcher est possible, tant que cela est fait rapidement après le renommage du serveur lui-même. Sur PerfWatcher, clic droit, renommer... ou modifier la liste de remplissage du folder (et dans ce cas, **peuplator** se chargera du renommage). Au niveau de Collectd, vous devrez aller sur les 3 répertoires de stockage de données :

- **/var/lib/collectd/rrd** ;
- **/var/lib/collectd/top** ;
- **/var/lib/collectd/notifications**.

Vous risquez néanmoins de rencontrer un problème : le nouveau serveur existe déjà. Vous allez donc devoir fusionner les répertoires. Pour les notifications, il suffira de recopier les fichiers. Pour les fichiers du répertoire **top**, il faut s'attendre à des fichiers qui portent le même nom. Si tel est le cas, il vaudra mieux garder les nouveaux en s'assurant qu'ils ne se chevauchent pas avec les anciens restants (voir les timestamps à l'intérieur des fichiers). Sinon, la fusion s'effectue en copiant simplement les anciens fichiers et répertoires dans les nouveaux. Enfin, si l'agent a été lancé sur le serveur avec le nouveau nom, des fichiers RRD auront été créés. Il n'est pas possible de fusionner facilement deux fichiers RRD. Aussi, pour chaque conflit, vous devrez choisir entre garder l'ancien ou le nouveau. Généralement vous préférerez l'ancien, car le nouveau n'aura que quelques minutes, voire heures (jours ?) d'historique, contre des mois sur l'ancien. Cela occasionnera un trou dans l'historique, ce qui est le prix à payer.

Notez malgré tout que si un serveur change de nom, il change en général aussi de fonction. Il est souvent préférable de garder un historique qui ne s'alimente plus, voire de le supprimer s'il est devenu inutile, et de repartir sur un historique neuf sur le serveur renommé. La manipulation de renommage est donc finalement généralement inutile et très peu effectuée dans la vraie vie.

5 Séparer les instances du serveur Collectd

Dans certaines configurations, il peut être intéressant de lancer le serveur Collectd sous forme de plusieurs processus. L'un pourra être dédié à l'écriture des fichiers RRD pendant que l'autre se chargera des notifications, **top process** et du module **jsonrpc**. Cela permet de rendre plus souple la gestion du serveur. Mais dans ce cas, nous vous conseillons d'avoir un processus dont le rôle sera uniquement de recevoir les paquets venant du réseau, et de les dispatcher aux autres processus Collectd. Le *dispatcheur* sera alors une sorte de proxy.

Ce mode de fonctionnement a un coût. Il vous oblige à utiliser deux, trois ou quatre fois l'espace mémoire nécessaire à un processus (selon le nombre de processus que vous aurez choisis). Cela est non négligeable, car l'état du parc est en temps réel en mémoire de chacun des processus Collectd que vous allez lancer. Mais il vous permet de relancer un processus et ses modules sans toucher aux autres. Cela est vraiment peu utile avec un petit parc de machines. Un seul processus Collectd

devrait vous suffire. Avec un plus gros parc, l'expérience montre qu'il arrive parfois de modifier les chaînes de filtrage du dispatcheur. Dans ce cas, il est pratique de ne relancer que le dispatcheur et de ne perdre quasiment aucune donnée, puisque le dispatcheur est un proxy et non un agent de stockage.

Dans un premier temps, nous vous déconseillons donc de le faire. Par la suite, lorsque le besoin s'en fera sentir, répartir les tâches entre plusieurs Collectd sera le bienvenu.

Enfin, il est de bon ton d'avoir un serveur de production et un serveur de test et développement. Vous pouvez très bien profiter des données de production en faisant en sorte que le serveur de production qui reçoit les données (directive **Listen** du module **network**) les transmette également au serveur de test et développement (directive **Server**). Si ce dernier est plus petit, vous pourrez définir une chaîne de filtrage pour ne retenir qu'une donnée sur deux, ou comme dans l'exemple ci-dessous, une donnée sur 5 avec le module **hash** :

```
LoadPlugin match_hashed
LoadPlugin match_regex

<Chain "PreCache">
    <Rule>
        <Match "regex">
            Host "^testdev$"
        </Match>
        Target "return"
    </Rule>
    <Rule>
        <Match "hashed">
            Match 0 5
        </Match>
        Target "return"
    </Rule>
    Target "stop"
</Chain>
```

Ce filtrage est à mettre en place sur le serveur de développement (s'appelant *testdev* ici). Il laisse passer les paquets qu'il génère lui-même grâce à la règle testant son nom (premier bloc **<rules>**). Pour les autres paquets, il calcule un hachage pouvant prendre 5 valeurs (de 0 à 4) et ne garde que ceux dont la valeur est 0, soit 1 sur 5.

6 Un PerfWatcher connecté à plusieurs serveurs Collectd

... ou *One PerfWatcher to rule them (Collectd) all.*

Si vous commencez à manquer d'espace disque ou de performances en écriture sur votre disque... Si vous souhaitez simplement mettre Collectd sur un serveur et l'interface utilisateur PerfWatcher sur un autre... Depuis PerfWatcher 2.0, cela est possible.

Abordons d'abord la notion de source Collectd. Il s'agit d'un ensemble de paramètres, qui définissent un serveur Collectd et son plugin **jsonrpc**. Dans le fichier de configuration **config.php** de PerfWatcher, vous pouvez lire ces lignes :

```
/**
 *
 * Collectd sources definitions
 * Add/set your collectd sources
 * Note : localhost as a server needs a Unix socket for rrdcached
 *
 */
$collectd_source_default = "localhost";
$collectd_sources = array(
    "localhost" => array( 'hostname' => "localhost", 'jsonrpc' => "http://127.0.0.1:8090/",
'proxy' => null ),
);
```

Dans ces paramètres est définie une seule source, nommée **localhost**. Elle est d'ailleurs également définie comme source par défaut.

Une source est un ensemble de 3 paramètres :

- **hostname** : le nom du serveur Collectd ;
- **jsonrpc** : l'URL du module **jsonrpc** du Collectd ;
- **proxy** : des paramètres pour un proxy web éventuel pour joindre le module **jsonrpc**.

En indiquant une autre URL que celle définie ci-dessus (et par défaut), vous pouvez séparer PerfWatcher et Collectd en les hébergeant sur deux serveurs différents.

Le paramètre **$collectd_sources** est, comme vous pouvez le remarquer, un tableau. Vous pouvez donc multiplier les lignes de définition de sources Collectd et ainsi disposer de plusieurs serveurs Collectd.

Sur l'interface PerfWatcher, le choix de la source s'effectue sur les « folders » (voir Fig. 1 page suivante).

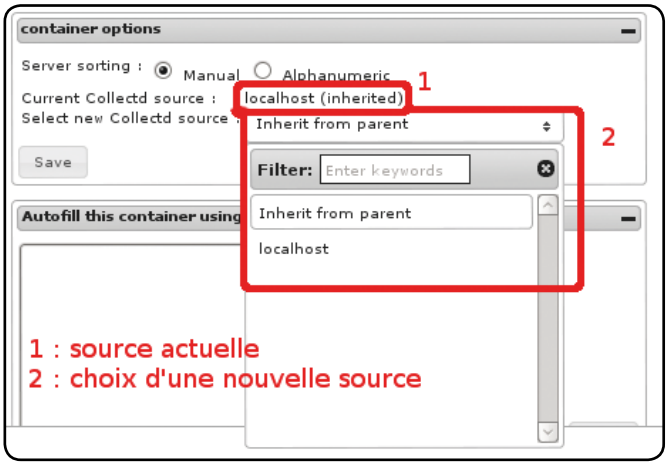


Figure 1

Ainsi, pour chaque serveur contenu dans un folder, PerfWatcher interrogera le serveur Collectd de celui-ci. Par défaut, la source est héritée du folder parent.

Rappelez-vous également que dans les onglets utilisateurs, éditables avec une syntaxe Markdown, vous pouvez indiquer un graphe avec la syntaxe suivante :

```
rrdgraph(<source>, <hostname>, <plugin>, <plugin instance>, <type>, <type instance>)
```

Le premier paramètre est la source. En d'autres termes, sur une même page, vous pouvez faire cohabiter plusieurs graphes de sources distinctes.

Faites attention : une des principales limitations d'avoir plusieurs sources différentes est l'utilisation des agrégateurs. En effet, comme l'agrégation s'effectue sur un serveur Collectd, vous ne pouvez pas agréger les données de serveurs remontant à des serveurs Collectd distincts. Une astuce consisterait à connecter les divers serveurs Collectd vers un serveur Collectd dédié à l'agrégation. Il ne stockerait donc rien d'autre que les données d'agrégation. Aidez-vous de la section précédente pour cela (« Séparer les instances du serveur Collectd »).

7 Programmation d'un module pour Collectd

7.1 Définition de l'identifiant

Pour programmer un module, vous devez savoir précisément ce qu'est un identifiant, comme ceux que nous avons vus en guise de titre pour chaque graphe. Un identifiant s'écrit ainsi :

```
serveur/plugin[-instance]/type[-instance]
```

Le nom du serveur tombe sous le sens, quoiqu'il vous est tout à fait possible d'y indiquer n'importe quel nom, y compris le prénom de votre sonde. Puis vient le nom du module. L'instance du module est facultative, mais peut servir à déterminer sur quoi a été appliqué le module. C'est ici, par exemple, que vous indiquerez le nom d'un disque pour le module **disk**. Ensuite vient le type. Celui-ci doit impérativement être défini au préalable dans la liste des types connus, c'est-à-dire dans le fichier **TypesDB**. Reprenez votre fichier de configuration **/etc/collectd/collectd.conf** et vous y trouverez cette directive au début. Si le type adapté à votre valeur n'existe pas, souvenez-vous que nous vous avions conseillé d'ajouter une ligne **TypesDB "/etc/collectd/custom_types.db"** pour y placer les vôtres. Enfin vient l'instance du type. Cela sert à placer plusieurs valeurs d'un même type dans le même graphe. Ce sera par exemple **input**, **output** et **forward**, ou **read** et **write**...

Il existe quatre sortes de types :

- **GAUGE** : la jauge est facile à comprendre. Elle indique un nombre représentatif d'un état, tel qu'une température ou le nombre de carottes présentes dans votre réfrigérateur ;
- **COUNTER** : le compteur est également facile à comprendre. Il mesure... un compteur, qui a pour caractéristique de ne jamais diminuer. Vous pouvez ainsi compter le nombre de pièces jaunes trouvées dans la rue. Le compteur prend en compte le *buffer overflow*. Donc si vous videz votre caisse à pièces jaunes dans une caisse plus grande, cela ne remet pas le compteur à zéro ;
- **DERIVE** : la dérive est un peu moins évidente, car elle s'intéresse à la différence entre une valeur et la précédente reçue. Si vous indiquez le nombre de pièces de votre porte-monnaie, vous verrez non pas le nombre de pièces, mais le nombre qui y rentre et le nombre qui en sort ;
- **ABSOLUTE** : moins utile, le compteur absolu est un compteur qui est remis à zéro (au contraire de son homologue **COUNTER**) à chaque fois que vous le lisez. Cela peut être pratique pour savoir combien de choses se sont empilées sur votre liste de choses à faire depuis la dernière fois que vous l'avez regardée.

Nous vous invitons à lire le contenu des fichiers **TypesDB** de Collectd et PerfWatcher, et nous vous suggérons de compléter votre lecture par la page de manuel de **rrdcreate** du projet RRDTool et par http://oss.oetiker.ch/rrdtool/tut/rrd-beginners.en.html pour en comprendre les détails.

7.2 PUTVAL et la socket

Avant de vous lancer dans l'écriture d'un module Collectd, demandez-vous s'il n'est pas plus simple (et surtout plus facile à maintenir) d'utiliser un exécutable externe capable de dialoguer avec Collectd à l'aide du *Plain Text Protocol*. Nous avons vu dans l'article traitant de l'installation de Collectd et PerfWatcher les instructions **LISTVAL** et **GETVAL**. Voici l'instruction **PUTVAL** :

```
PUTVAL Identifier [OptionList] Valuelist
```

Parce qu'un exemple vaut mieux que toutes les explications du monde, voici comment intégrer la valeur **23** comme température pour notre nouveau module d'une fois :

```
PUTVAL sedna/temperature-dehors/gauge interval=60 1375022417:23
```

Nous avons indiqué comme option **interval=60**. Cela est facultatif, notre thermomètre vient de passer à 24, et nous allons ajouter la température intérieure :

```
PUTVAL sedna/temperature-dehors/gauge 1375022477:24
PUTVAL sedna/temperature-dedans/gauge 1375022477:21
```

La liste de valeurs s'écrit avec un timestamp suivi d'autant de valeurs que vous avez d'instances de types, séparés par des deux-points. Dans notre cas, nous n'avons qu'une seule instance pour notre type *gauge*, que nous n'avons pas indiquée et qui s'appellera par défaut *value*. Notez que *gauge* se trouve dans le fichiers **TypesDB** de Collectd et a pour type... **GAUGE** !

7.3 Le module exec

Mais voilà, vous voulez que votre script ou exécutable se lance régulièrement. Crontab, avez-vous pensé... Mais pourquoi ne pas utiliser le module **exec** de Collectd ?

Voici sa syntaxe :

```
LoadPlugin exec
<Plugin exec>
    Exec "myuser:mygroup" "myprog"
    Exec "otheruser" "/path/to/another/binary" "arg0" "arg1"
</Plugin>
```

En indiquant la première ligne, Collectd lancera l'exécutable **myprog** (qui doit pouvoir être trouvé avec la variable d'environnement **PATH**) avec le compte **myuser** et le groupe **mygroup**. Avec la deuxième ligne, seul le compte est indiqué, puis vient l'exécutable avec son chemin absolu, suivi de deux arguments. Sachez qu'il est également possible de générer des notifications avec **NotificationExec** au lieu de **Exec**.

Un exécutable aura ce genre de forme (ici en script bash) :

```
HOSTNAME="${COLLECTD_HOSTNAME:-localhost}"
INTERVAL="${COLLECTD_INTERVAL:-60}"

while sleep "$INTERVAL"; do
# exécution du code déterminant la valeur à retourner
    VALUE=xx
# valeur à retourner
    echo "PUTVAL \"$HOSTNAME/plugin-instance/type-instance\"
interval=$INTERVAL N:$VALUE"
done
```

Il s'agit donc d'afficher une ligne en Plain Text Protocol en boucle, puis d'attendre l'intervalle de temps du module. Deux variables d'environnement sont fournies par Collectd, **COLLECTD_HOSTNAME** et **COLLECTD_INTERVAL**.

Vous notez que, bien que ce soit le module **exec** qui lance ce script, la valeur peut avoir n'importe quel identifiant, y compris pour le module. Tout aussi important, faites attention, si votre langage de programmation implique un cache de la sortie standard, à bien vider ce cache (*flush*) avant de s'endormir avant l'occurrence suivante.

Note

Si vous écrivez votre script en Perl, vous pouvez activer le mode auto-flush ainsi :

```
$| = 1 ;
```

Enfin, il faut savoir que si votre script s'arrête, Collectd le relancera dès qu'il se rendra compte de son absence. Aussi, un script pourrait se contenter du calcul de la valeur suivi de la ligne **PUTVAL**. Il serait lancé à chaque intervalle de temps par Collectd. Mais cette pratique est à éviter, car elle multiplie la création de processus, coûteuse en ressources, surtout à chaque intervalle.

Note

Astuce de programmation : si votre script tourne en boucle, vérifiez régulièrement que le processus père, **collectd**, est en cours d'exécution (retrouvez son PID avec une fonction comme **getppid()**). Votre script peut prendre fin si le processus père a disparu sans le signaler à son fils (votre script).

7.4 Le module tail

Vous pouvez encore utiliser le module **tail**. Celui-ci analyse un fichier journal (*log*) à la recherche de motifs que vous y auriez définis. Ce module est un peu plus délicat à configurer, parce qu'il vous revient de définir comment analyser chaque ligne et comment réagir lorsque l'une d'elles correspond. Si vous avez des besoins assez communs, il se peut que vous trouviez des configurations toutes prêtes sur Internet. Par exemple, l'analyse des journaux de Postfix peut vous renseigner sur le nombre de courriers électroniques reçus ou envoyés.

7.5 Votre propre module

Si vous pensez que malgré toutes ces solutions il vous est préférable de coder un module, vous pouvez vous lancer. La porte d'entrée se trouve sur https://collectd.org/wiki/index.php/Plugin_architecture.

Le principe consiste à écrire un fichier en C (la licence GPLv2 est imposée par souci de compatibilité avec le code existant). Vous y créerez une fonction de prototype **void module_register (void);**. Toutes les autres fonctions devront être déclarées comme **static** pour vous assurer qu'elles sont bien locales à votre module. Cette fonction **module_register()** vous servira à définir des fonctions de rappel pour Collectd :

- votre fonction de configuration avec **plugin_register_config()** ou **plugin_register_complex_config** ;
- votre fonction d'initialisation avec **plugin_register_init()** ;

- votre fonction appelée par un thread *read*, avec `plugin_register_read()` ou `plugin_register_complex_read()` ;
- votre fonction appelée par un thread *write*, avec `plugin_register_write()` ;
- votre fonction pour forcer à vider vos caches, avec `plugin_register_flush()` ;
- votre fonction d'arrêt, avec `plugin_register_shutdown()`.

L'enregistrement de toutes ces fonctions de rappel est facultatif. Si vous n'enregistrez pas de fonction de rappel, elle ne sera pas appelée. Ainsi, un module sans fonction de lecture ou d'écriture (ou de journalisation, non indiquée ci-dessus) ne présente aucun intérêt. Par contre, vous pouvez avoir des modules sans configuration, lorsque les informations à obtenir du système sont à des emplacements triviaux ou normés.

Collectd, lorsqu'il aura connaissance de votre module, appellera dans l'ordre votre fonction de configuration au démarrage, puis d'initialisation. Ensuite, à chaque fois qu'il sera temps et qu'un thread sera disponible, votre fonction de rappel de lecture ou d'écriture sera appelée en fonction de la nature du thread. De temps en temps, ainsi qu'à son arrêt, Collectd peut demander à vider le cache. À l'extinction de Collectd, une fois les caches vidés, la fonction d'arrêt est appelée. Et puis c'est fini pour ce cycle. La roue tourne.

Maintenant que vous connaissez les grandes lignes, il ne vous reste plus qu'à vous plonger dans la documentation de Collectd et à lire le code de certains modules en commençant par les plus simples.

8 Participation au projet PerfWatcher

PerfWatcher, bien qu'ayant déjà quelques années derrière lui, est un projet encore jeune au sens collaboratif. Par exemple, au moment où nous écrivons cet article, il n'existe pas de liste de diffusion. Vous devrez contacter directement l'auteur, Cyril Feraudet, son adresse électronique se trouvant sur le site web de PerfWatcher. Mais c'est parce que le projet est jeune, qu'il y a beaucoup à faire, que vous pouvez sûrement y contribuer d'une façon ou d'une autre. Le projet est hébergé sur GitHub pour faciliter son développement collaboratif.

Collectd-pw, en tant que collection de patchs, fait également partie du projet PerfWatcher. Par contre, si vous souhaitez soumettre un bug ou contribuer, vous devrez d'abord déterminer si cela concerne le projet Collectd ou les patchs Collectd-pw. Dans le premier cas, vous pouvez en faire part aux développeurs de PerfWatcher/Collectd-pw, mais ils vous redirigeront probablement vers le projet Collectd, pour que cela puisse bénéficier à tout le monde, y compris à PerfWatcher/Collectd-pw qui se base dessus. Dans le second cas, ils se feront un plaisir de vous aider.

Conclusion

Nous voici au terme de cette série d'articles sur le couple Collectd-PerfWatcher. Et pourtant, de nombreux sujets n'ont pas été abordés. Si vous utilisez Collectd et PerfWatcher dans un contexte professionnel ou amateur avancé, vous les rencontrerez probablement. Du fait de la modularité de Collectd et des fonctionnalités de PerfWatcher, les problèmes rencontrent souvent plusieurs solutions. On prend alors un certain plaisir à discuter et choisir laquelle mette en œuvre. On s'amuse enfin lors de sa réalisation. ■

Remerciements

L'auteur tient à remercier tous ceux sans qui Collectd, Collectd-pw et PerfWatcher ne seraient pas. Florian Foster et les nombreux contributeurs de Collectd. Cyril Feraudet pour avoir lancé le projet PerfWatcher qu'il maintient toujours, et pour avoir commencé à écrire des patchs pour Collectd, ce qui allait devenir Collectd-pw. En tant que contributeur de nombreux patchs de Collectd-pw, l'auteur remercie également ceux qui le soutiennent, ils se reconnaîtront.

Dernière minute

PerfWatcher-2.1 vient de sortir fin mars. Outre quelques corrections de bugs et nouvelles fonctionnalités mineures, cette version est la première à proposer le support PostgreSQL. Le détail se trouve comme d'habitude dans le fichier `ChangeLog` et dans les notes de versions (`doc/ReleaseNotes-2.1.txt`).

MISE EN PLACE D'UN SERVEUR DE MESSAGERIE POUR MODOBOA

par ***Antoine Nguyen***

Modoboa permet de gérer plus simplement votre plateforme de messagerie, mais nécessite pour cela... une plateforme de messagerie ! Découvrez comment déployer la vôtre en lisant cet article !

D'une manière ou d'une autre, nous sommes tous en recherche d'indépendance. Dans notre quotidien ou via les outils que nous utilisons, cette notion importante prend de plus en plus de place, certains événements récents (cf. affaire PRISM) ne nous le rappellent que trop. La messagerie ne fait pas exception à la règle. Héberger sa messagerie est longtemps resté une activité réservée aux initiés, d'autant que les services gratuits et simples d'utilisation sont monnaie courante de nos jours. À quoi bon se fatiguer me direz-vous ?

Chacun étant libre (et responsable) de ses choix, aucune réponse à cette question ne sera apportée. Sachez cependant que Modoboa tente de fournir une solution. En simplifiant la gestion au quotidien d'une plateforme de messagerie, l'auto-hébergement n'est plus synonyme d'expertise et de maux de tête. Seule la mise en place initiale reste légèrement fastidieuse, cet article est justement fait pour vous aider !

Dans ce premier article, nous allons préparer le système en vue d'utiliser Modoboa. Pour des raisons de simplicité (et par choix personnel), la distribution Debian Wheezy est utilisée pour cet article. Son installation ne sera cependant pas traitée, je pars du principe que vous disposez d'un système fonctionnel.

> **Note**
>
> Certains exemples fournis dans cet article font référence à un domaine fictif `domain.tld` qu'il faudra remplacer par le vôtre.

1 Bases de données

La communication entre les différents composants de la plateforme est assurée par une base de données SQL. PostgreSQL (simplicité, choix personnel, tout ça) est utilisé pour cet article.

Celui-ci n'étant pas installé de base sur le système, nous allons y remédier :

```
$ apt-get install postgresql
```

Deux bases différentes vont être nécessaires dans le cadre de cet article :

1. Une pour héberger les tables de Modoboa ;
2. Une pour héberger les tables d'Amavis.

Pour Modoboa, créez un utilisateur, ainsi qu'une base de la manière suivante :

```
$ su - postgres
$ psql
postgres=# CREATE USER modoboauser WITH PASSWORD
'password';
postgres=# CREATE DATABASE modoboa OWNER modoboauser;
```

Répétez l'opération pour Amavis :

```
postgres=# CREATE USER amavisuser WITH PASSWORD
'password';
postgres=# CREATE DATABASE amavis OWNER amavisuser;
postgres=# \q
```

Vous êtes bien évidemment libre de modifier les noms et mots de passe utilisés dans ces exemples (c'est d'ailleurs fortement recommandé !).

2 Amavis

Amavis est un inspecteur de contenu (*content filter* pour les puristes) générique permettant la détection du spam et des virus en travaillant de pair avec différents moteurs (spamassassin, clamav, etc.). De plus, il offre (parmi bien d'autres fonctionnalités) : la mise en quarantaine dans une base SQL, un support DKIM (*DomainKeys Identified Mail*) pour signer les messages sortants et vérifier les messages entrants, etc.

L'installation d'Amavis sous Debian est rapide, car un paquet est disponible nativement :

```
$ apt-get install amavisd-new libdbd-pg-perl spamassassin
```

Côté configuration, cet article se concentre sur le paramétrage nécessaire au fonctionnement avec Modoboa (anti-spam uniquement).

2.1 Création de la base de données

Pourquoi faire simple quand on peut faire compliqué ?

C'est certainement ce que s'est dit le créateur d'Amavis lorsqu'il a décidé de ne pas fournir de moyen rapide pour installer le schéma de la base attendu par Amavis. Ce schéma est disponible dans un fichier **README**, au milieu d'autres informations (**/usr/share/doc/amavisd-new/README.sql-pg.gz**).

Comme je suis de nature généreuse, j'ai pris cinq minutes pour extraire ce schéma et pour le mettre à disposition sur le site de Modoboa ;-)

Pour le récupérer puis l'installer, exécutez les commandes suivantes :

```
$ su - postgres
$ wget http://modoboa.org/resources/amavis_pgsql_schema_2.7.0.sql
$ psql amavis < amavis_pgsql_schema_2.7.0.sql
```

2.2 Mise en quarantaine

Le filtrage du spam est par défaut désactivé sous Debian. Pour l'activer, ouvrez le fichier **/etc/amavis/conf.d/15-content_filter_mode** et dé-commentez les lignes suivantes :

```
@bypass_spam_checks_maps = (
   \%bypass_spam_checks, \@bypass_spam_checks_acl, \$bypass_spam_checks_re);
```

Ouvrez ensuite le fichier **/etc/amavis/conf.d/50-user** et ajoutez les lignes suivantes :

```
# SQL backend (mail preferences, quarantine)
#
$sql_allow_8bit_address = 1;
@lookup_sql_dsn = ( [ 'DBI:Pg:database=amavis;host=127.0.0.1',
'amavisuser', 'password' ] );
@storage_sql_dsn = ( [ 'DBI:Pg:database=amavis;host=127.0.0.1',
'amavisuser', 'password' ] );
$spam_quarantine_method = 'sql:';
$banned_files_quarantine_method = 'sql:';
```

2.3 Libération des messages en quarantaine

Il arrive que certains messages soient considérés comme du spam alors qu'ils ne le sont pas, on parle alors de « faux positifs ». Heureusement, Amavis offre la possibilité de libérer des messages de la quarantaine. Pour activer cette fonctionnalité, éditez le fichier **/etc/amavis/conf.d/50-user** et ajoutez le contenu suivant :

```
$inet_socket_port = [9998, 10024] ;

# POLICY BANKS: AM.PDP
#
$interface_policy{'9998'} = 'AM.PDP-INET';

$policy_bank{'AM.PDP-INET'} = {
  protocol => 'AM.PDP',
  inet_acl => [qw( 127.0.0.1 )],
  auth_required_release => 0,
};
```

Il ne reste plus qu'à redémarrer Amavis :

```
$ /etc/init.d/amavis restart
```

3 Modoboa

3.1 Installation

Certains d'entre vous seront déçus : il n'existe pas de paquet Modoboa pour Debian. L'installation se fait via PyPI, le dépôt officiel de modules additionnels pour Python. Rassurez-vous, son utilisation est extrêmement simple. L'utilitaire **pip**, que l'on peut considérer comme un client PyPI, est fourni nativement par la distribution :

```
$ apt-get install python-pip
```

Un des avantages à utiliser PyPI est de pouvoir disposer de versions plus à jour que celles embarquées dans une distribution comme Debian. Le gros inconvénient est que seules les sources sont fournies, ce qui nécessite de compiler les dépendances mixant du C et du Python.

Pour éviter cela, nous allons tout simplement utiliser APT pour récupérer les librairies impactées :

```
$ apt-get install python-rrdtool python-lxml python-crypto python-psycopg2
```

Puis utiliser **pip** pour le reste :

```
$ pip install modoboa
```

3.2 Déploiement d'une instance

Modoboa étant basé sur Django, vous ne pouvez pas l'utiliser tel quel. Il se présente sous forme d'une collection de modules que vous avez la possibilité d'utiliser pour votre site (ou instance). Partant du principe que vous (les utilisateurs) ne connaissez, ni ne maîtrisez Django, une commande est fournie afin de simplifier la création d'une instance.

Déterminez l'emplacement où vous allez héberger les fichiers de cette instance (par exemple : **/srv/www/modoboa**) et exécutez la commande suivante :

```
$ mkdir /srv/www
$ cd /srv/www
$ modoboa-admin.py deploy -syncdb -collectstatic -with-amavis \
    --dburl postgres://modoboauser:password@localhost/modoboa \
    --amavis_dburl postgres://amavisuser:password@localhost/amavis \
    --domain modoboa.example.tld \
    modoboa_instance
```

3.3 Permissions

Certaines actions effectuées depuis l'interface entraînent une action sur le système (comme par exemple la suppression d'une boîte aux lettres). Pour cela, certaines permissions doivent être accordées à Modoboa et plus précisément à l'utilisateur système en charge de son exécution. En général, il s'agit de l'utilisateur en charge des processus web (Nginx et autres) : **www-data** sur un système Debian.

Commencez par installer l'utilitaire **sudo** :

```
$ apt-get install sudo
```

Puis, ouvrez le fichier **/etc/sudoers.d/modoboa** et copiez-y le contenu suivant :

```
www-data ALL=(vmail) NOPASSWD: ALL
```

Cette ligne indique à **sudo** que l'utilisateur **www-data** peut se faire passer pour l'utilisateur **vmail** afin d'exécuter sans renseigner de mot de passe toutes les commandes auxquelles ce dernier a accès.

Note

À ce moment de la procédure, ces utilisateurs ne sont pas encore disponibles. Cela ne devrait pas poser de problème puisqu'ils seront créés par la suite.

3.4 Tâches automatiques

Afin d'assurer un fonctionnement optimal dans le temps, Modoboa fournit un ensemble de scripts **cron** (ou tâches automatiques) chargés d'effectuer des actions bien précises à des intervalles réguliers.

Créez le fichier **/etc/cron.d/modoboa** avec le contenu suivant :

```
# Operations on mailboxes
* * * * * vmail python /srv/www/modoboa_instance/manage.py handle_mailbox_
operations

# Sessions table cleanup
0 0 * * * root python /srv/www/modoboa_instance/manage.py cleanup

# Logs table cleanup
0 0 * * * root python /srv/www/modoboa_instance/manage.py cleanlogs

# Quarantine cleanup
0 0 * * * root python /srv/www/modoboa_instance/manage.py qcleanup

# Notifications about pending release requests
0 12 * * * root python /srv/www/modoboa_instance/manage.py amnotify
--baseurl='http://modoboa.example.tld'

# Logs parsing
*/5 * * * * root python /srv/www/modoboa_instance/manage.py logparser &>
/dev/null
```

4 Dovecot

Dovecot est un serveur IMAP et POP3 libre. Il est l'un des plus utilisés dans le monde à l'heure actuelle.

Son installation est simple, puisque des paquets sont disponibles en standard sous Debian. Dans le cadre de

Complétez votre collection d'anciens numéros !

boutique.ed-diamond.com

numerique.ed-diamond.com

cet article, nous nous contenterons d'installer les serveurs IMAP et Sieve fournis par Dovecot :

```
$ apt-get install dovecot-imapd dovecot-sieve dovecot-managesieved
dovecot-lmtpd dovecot-pgsql
```

Côté configuration, nous resterons volontairement basiques. Dovecot est un logiciel complexe et sa configuration suit la même logique. L'ensemble de la configuration est située dans le répertoire **/etc/dovecot** et elle est découpée en plusieurs fichiers qui se trouvent dans le répertoire **/etc/dovecot/conf.d**.

4.1 Boîtes aux lettres

Les boîtes aux lettres de chaque utilisateur sont stockées sur un système de fichiers et sont gérées par Dovecot. Ce dernier doit connaître :

1. L'emplacement du stockage ;
2. Le format de stockage ;
3. La structure interne à utiliser lors d'une création.

La définition de l'emplacement est décrite un peu plus loin (cf. Authentification). Cependant, un propriétaire système unique est nécessaire pour le stockage sur système de fichiers :

```
$ useradd -d /srv/vmail -m vmail
```

Le format se configure dans le fichier **/etc/dovecot/conf.d/10-mail.conf** :

```
# maildir
mail_location = maildir:~/maildir
```

Comme vous l'aurez compris, nous indiquons à Dovecot d'utiliser le format Maildir (un message par fichier).

Dans le même fichier, nous allons indiquer quelle est la structure (composition) souhaitée des boîtes nouvellement créées. Localisez la définition de l'espace de noms **inbox** et modifiez-le comme suit :

```
namespace inbox {
  inbox = yes
  mailbox Drafts {
    auto = subscribe
   special_use = \Drafts
 }
  mailbox Junk {
    auto = subscribe
    special_use = \Junk
  }
  mailbox Sent {
    auto = subscribe
    special_use = \Sent
  }
  mailbox Trash {
    auto = subscribe
    special_use = \Trash
  }
}
```

Chaque nouvelle boîte aux lettres sera donc dotée par défaut de quatre sous-boîtes :

- *Drafts* : stockage des brouillons ;
- *Junk* : stockage du pourriel ;
- *Sent* : stockage des messages envoyés ;
- *Trash* : poubelle.

4.2 Authentification

La configuration de l'authentification se fait en plusieurs étapes. Tout d'abord, il faut indiquer à Dovecot de chercher les informations dans la base SQL. Ouvrez le fichier **/etc/dovecot/conf.d/10-auth.conf** et dé-commentez la ligne correspondant à l'interface SQL (fin du fichier) :

```
#!include auth-system.conf.ext
!include auth-sql.conf.ext
#!include auth-ldap.conf.ext
#!include auth-passwdfile.conf.ext
#!include auth-checkpassword.conf.ext
#!include auth-vpopmail.conf.ext
#!include auth-static.conf.ext
```

Ensuite, ouvrez le fichier **/etc/dovecot/dovecot-sql.conf.ext** et modifiez les paramètres suivants :

```
driver = pgsql

connect = host=127.0.0.1 dbname=modoboa user=modoboauser password=password

password_query = SELECT email AS user, password FROM core_user WHERE email='%u'
and is_active

user_query = SELECT '/srv/vmail/%d/%n' AS home, <uid> as uid, <gid> as gid,
'*:bytes=' || mb.quota || 'M' AS quota_rule FROM admin_mailbox mb INNER JOIN
admin_domain dom ON mb.domain_id=dom.id WHERE mb.address='%n' AND dom.name='%d'

iterate_query = SELECT email AS username FROM core_user
```

Remplacez **<uid>** et **<gid>** par les valeurs correspondant à l'utilisateur **vmail** créé un peu plus tôt.

Vous remarquerez qu'un chemin absolu est utilisé dans cet exemple : **/srv/vmail**. Il s'agit du répertoire de stockage racine de toutes les boîtes aux lettres.

Pour terminer, indiquons à Dovecot de laisser une porte ouverte sur le système d'authentification, afin que Postfix puisse en profiter (section 5). Ouvrez le fichier **/etc/dovecot/conf.d/10-master.conf** et localisez la définition du service **auth**. Modifiez-la comme suit :

```
service auth {
  # ...
  # Postfix smtp-auth
  unix_listener /var/spool/postfix/private/auth {
    mode = 0666
    user = postfix
    group = postfix
  }
}
```

4.3 Livraison des messages

La livraison des messages se fait via l'utilisation du protocole LMTP (*Local Mail Transport Protocol*).

Ouvrez le fichier **/etc/dovecot/conf.d/10-master.conf**, localisez la définition du service **lmtp** et modifiez-la comme suit :

```
service lmtp {
    #...
    unix_listener /var/spool/postfix/private/
dovecot-lmtp {
      mode = 0600
      user = postfix
      group = postfix
    }
    #...
}
```

Ouvrez le fichier **/etc/dovecot/conf.d/20-lmtp.conf**, localisez encore la définition du protocole **lmtp** et modifiez-la comme suit :

```
protocol lmtp {
  postmaster_address = postmaster@example.tld
  mail_plugins = $mail_plugins
}
```

4.4 Quotas

La configuration des quotas est certainement la partie la plus complexe. Un quota est une limite physique appliquée à une boîte aux lettres. Elle peut concerner sa taille ou le nombre de messages dans une boîte aux lettres.

Commencez par ouvrir le fichier **/etc/dovecot/conf.d/10-mail.conf** et activez le greffon permettant l'activation des quotas :

```
mail_plugins = quota
```

Il faut ensuite effectuer une série de modifications.

Dans le fichier **/etc/dovecot/conf.d/20-lmtp.conf**, modifiez le paramètre **mail_plugins** :

```
mail_plugins = $mail_plugins quota
```

Dans le fichier **/etc/dovecot/conf.d/10-master.cf**, localisez la définition du service **dict** et modifiez-la en :

```
service dict {
  # If dict proxy is used, mail processes
  # should have access to its socket.
  # For example: mode=0660, group=vmail
  # and global mail_access_groups=vmail
  unix_listener dict {
    mode = 0600
    user = vmail
    #group =
  }
}
```

Dans le fichier **/etc/dovecot/conf.d/20-imap.conf**, localisez la définition du protocole **imap** et modifiez-la comme suit :

```
protocol imap {
  # ...

  mail_plugins = $mail_plugins imap_quota

  # ...
}
```

Dans le fichier **/etc/dovecot/conf.d/90-quota.conf**, localisez le bloc **plugin** en fin de fichier et transformez-le en :

```
plugin {
  quota = dict:User quota::proxy::quota
}
```

Dans le fichier **/etc/dovecot/dovecot.conf**, activez le stockage des quotas en base SQL :

```
dict {
quota = pgsql:/etc/dovecot/dovecot-dict-sql.conf.ext
}
```

Ouvrez finalement le fichier **/etc/dovecot/dovecot-dict-sql.conf.ext** et modifiez-le :

```
connect = host=127.0.0.1 dbname=modoboa
user=modoboauser password=password
map {
  pattern = priv/quota/storage
  table = admin_quota
  username_field = username
  value_field = bytes
}
map {
  pattern = priv/quota/messages
  table = admin_quota
  username_field = username
  value_field = messages
}
```

La configuration (dans le sens modification de fichiers) est terminée, mais il reste encore deux opérations à effectuer.

Premièrement, Django (le framework web utilisé pour développer Modoboa) ne crée pas les tables de manière optimale dans PostgreSQL. En effet, il ne définit pas les contraintes d'utilisation de la table **admin_quota**. Pour les ajouter :

```
$ su - postgres
$ psql modoboa
modoboa=# ALTER TABLE admin_quota ALTER COLUMN bytes SET DEFAULT 0;
modoboa=# ALTER TABLE admin_quota ALTER COLUMN messages SET DEFAULT
0;
```

Enfin, un *trigger* (réaction suite à un événement particulier) est nécessaire afin de mettre à jour l'état actuel des quotas (la consommation).

Finalement (cette fois-ci c'est vrai), installez le trigger en exécutant les commandes suivantes :

```
$ su - postgres
$ wget http://modoboa.org/resources/modoboa_postgres_trigger.sql
$ psql modoboa < modoboa_postgres_trigger.sql
```

4.5 Filtrage des messages entrants

Le filtrage des e-mails se fait via l'utilisation du langage Sieve et du protocole ManageSieve. Sieve permet de définir des règles de filtrage, tandis que ManageSieve permet de les manipuler à distance. Ces deux composants sont fournis par Dovecot, mais leur activation nécessite un léger paramétrage... Vous n'en êtes plus à quelques lignes près !

Ouvrez le fichier **/etc/dovecot/conf.d/20-managesieve.conf** et dé-commentez la déclaration des services suivants :

```
service managesieve-login {
  # ...
}

service managesieve {
  # ...
}

protocol sieve {
  # ...
}
```

Dans le fichier **/etc/dovecot/conf.d/20-lmtp.conf**, localisez la définition du protocole **lmtp** et activez le greffon **sieve** comme suit :

```
protocol lmtp {
  # ...
  mail_plugins = $mail_plugins quota sieve
}
```

Pour finir, ouvrez le fichier **/etc/dovecot/conf.d/90-sieve.conf** et modifiez la définition du greffon :

```
plugin {
  # Location of the active script. When ManageSieve is used this is actually
  # a symlink pointing to the active script in the sieve storage directory.
  sieve = ~/.dovecot.sieve

  #
  # The path to the directory where the personal Sieve scripts are stored. For
  # ManageSieve this is where the uploaded scripts are stored.
  sieve_dir = ~/sieve
}
```

Enfin, il ne vous reste plus qu'à redémarrer Dovecot pour que toutes les modifications soient activées :

```
$ /etc/init.d/dovecot restart
```

5 Postfix

Postfix est un MTA (*Mail Transport Agent*). Il s'agit du composant en charge de l'acheminement des e-mails vers leur destination via l'utilisation du protocole SMTP (*Simple Mail Transport Protocol*). Postfix étant une référence dans le domaine, la question de son utilisation ne se pose pas vraiment... (sans troll aucun, c'est promis).

Tout comme pour Dovecot, des paquets Debian sont disponibles en standard :

```
$ apt-get install postfix postfix-pgsql
```

Choisissez l'option « Pas de configuration » comme réponse à la question posée.

Là encore, nous nous contenterons d'une configuration simple. Les possibilités de Postfix sont telles qu'un magazine complet ne serait pas suffisant pour les décrire.

5.1 Enregistrement DNS

Le protocole SMTP peut être comparé à un protocole de routage : il prend en charge l'acheminement d'un message entre une source et une destination. L'identification de la destination se fait via le protocole DNS (*Domain Name Server*) qui va permettre au MTA source,

à partir d'un nom de domaine (ex : ngyn.org) de déterminer l'adresse IP du MTA destinataire.

Cette association est rendue possible grâce à la présence d'un enregistrement MX (*Mail eXchanger*) dans la zone du domaine destinataire. Ce type d'enregistrement est **obligatoire** au fonctionnement de votre messagerie.

Voici un exemple de définition d'enregistrement MX (syntaxe Bind) :

```
example.tld.       MX      10 modoboa.example.tld.
```

Votre *registrar* (hébergeur de nom de domaine) vous proposera certainement un outil plus simple pour le déclarer.

5.2 Requêtes d'accès à la base Modoboa

La communication entre Postfix et Modoboa se fait principalement via la base de données. Plusieurs requêtes, utilisées pour récupérer différents types d'informations, sont nécessaires. Heureusement, leur création est simplifiée par l'utilisation de la commande suivante :

```
$ modoboa-admin.py postfix_maps --categories std autoreply relaydomains \
    --dbtype postgres /etc/postfix
```

Répondez ensuite aux quelques questions posées.

Note

Si votre base de données est hébergée localement, utilisez **127.0.0.1** plutôt que **localhost** pour indiquer son adresse.

5.3 Configuration de base

Créez le fichier **/etc/postfix/main.cf** et copiez le contenu suivant à l'intérieur :

```
inet_protocols = ipv4
smtpd_banner = $myhostname ESMTP
biff = no

# appending .domain is the MUA's job.
append_dot_mydomain = no

myhostname =  modoboa.example.tld
mydomain = example.tld
myorigin = $mydomain
mydestination = $myhostname, localhost.$mydomain, localhost
mynetworks = 127.0.0.0/8
mailbox_size_limit = 0
owner_request_special = no
recipient_delimiter = +
empty_address_recipient = MAILER-DAEMON
relay_domains =
```

5.4 Support TLS

L'activation du chiffrement via TLS est utile pour sécuriser les communications entre les clients et le MTA. Dans le fichier **/etc/postfix/main.cf**, ajoutez les lignes suivantes :

```
smtpd_use_tls = yes
smtpd_tls_auth_only = no
smtpd_tls_key_file = /etc/ssl/private/ssl-cert-snakeoil.key
smtpd_tls_cert_file = /etc/ssl/certs/ssl-cert-snakeoil.pem
smtpd_tls_loglevel = 1
smtpd_tls_received_header = yes
smtpd_tls_session_cache_timeout = 3600s
smtpd_tls_ask_ccert = yes
tls_random_source = dev:/dev/urandom
smtp_tls_note_starttls_offer = yes
```

Vous avez sans doute remarqué que nous n'avons pas créé le certificat utilisé (*snakeoil*). Il s'agit en effet d'un certificat factice généré lors de l'installation d'OpenSSL sur le serveur.

Note

Il est fortement conseillé d'utiliser un véritable certificat !

5.5 Domaines virtuels

Historiquement, un MTA était chargé de gérer les messages d'un unique domaine et seuls les destinataires disposant d'un compte de type UNIX (système) étaient considérés comme valides. Avec le temps, ce fonctionnement trop restrictif a été amélioré afin de permettre :

- la gestion de plusieurs domaines par un seul MTA ;
- le support d'utilisateurs non-UNIX ou « virtuels ».

Dans cette configuration, un seul utilisateur système est utilisé pour stocker les messages sur le système de fichiers.

Le support des domaines virtuels s'active en quelques lignes de configuration. Éditez le fichier **/etc/postfix/main.cf** et ajoutez-y les instructions suivantes :

```
virtual_transport = lmtp:unix:private/dovecot-lmtp
virtual_mailbox_domains = pgsql:/etc/postfix/sql-domains.cf
virtual_alias_domains = pgsql:/etc/postfix/sql-domain-aliases.cf
virtual_alias_maps = pgsql:/etc/postfix/sql-aliases.cf,
      pgsql:/etc/postfix/sql-domain-aliases-mailboxes.cf,
      pgsql:/etc/postfix/sql-catchall-aliases.cf

smtpd_recipient_restrictions =
      permit_mynetworks
      check_recipient_access pgsql:/etc/postfix/sql-maintain.cf
      reject_unverified_recipient
      reject_unauth_destination
```

5.6 Relayage de domaines

Un MTA n'est pas toujours la destination finale d'un domaine, il peut aussi faire office de relais. Un cas classique d'utilisation d'un relais est la mise en place d'une passerelle dédiée au filtrage (sorte de *reverse proxy*). La passerelle filtre les messages en entrée et transfère le contenu valide au véritable MTA qu'il protège. Ce type de fonctionnement nécessite un paramétrage spécifique.

Ouvrez le fichier **/etc/postfix/main.cf** et ajoutez ou modifiez les paramètres suivants :

```
relay_domains = pgsql:/etc/postfix/sql-relaydomains.cf

transport_maps =
    pgsql:/etc/postfix/sql-relaydomains-transport.cf
    pgsql:/etc/postfix/sql-relaydomain-aliases-transport.cf

smtpd_recipient_restrictions =
    permit_mynetworks
    check_recipient_access pgsql:/etc/postfix/sql-maintain.cf
    reject_unverified_recipient
    reject_unauth_destination
    check_recipient_access
        pgsql:/etc/postfix/sql-relay-recipient-verification.cf
```

5.7 Authentification

Activer l'authentification au niveau d'un MTA permet de distinguer les utilisateurs « internes » des utilisateurs « externes ». Les « internes » (tiers de confiance) seront par exemple autorisés à envoyer des messages à n'importe quelle destination, tandis que les « externes » seront limités au(x) domaine(s) géré(s) par ce MTA.

Les utilisateurs (au sens humain) ne sont pas censés utiliser le même port de communication que les serveurs. Le port 25 est destiné aux communications entre serveurs, tandis que le port 587 (*submission*) est réservé aux utilisateurs.

L'activation du port 587 se fait dans le fichier **/etc/postfix/master.cf**. Ouvrez-le, puis dé-commentez et adaptez les lignes suivantes :

```
submission inet n       -       -       -       -       smtpd
  -o syslog_name=postfix/submission
  -o smtpd_tls_security_level=encrypt
  -o smtpd_sasl_auth_enable=yes
  -o smtpd_client_restrictions=permit_sasl_authenticated,reject
  -o milter_macro_daemon_name=ORIGINATING
```

Pour simplifier l'architecture globale, Postfix va s'appuyer sur Dovecot pour l'opération d'authentification en tant que telle. Ce dernier laisse ouverte une porte de communication dédiée, il faut juste indiquer à Postfix son emplacement :

```
smtpd_sasl_auth_enable = yes
smtpd_sasl_type = dovecot
smtpd_sasl_path = private/auth
broken_sasl_auth_clients = yes
smtpd_sasl_security_options = noanonymous
```

Ajustez finalement le contenu du paramètre **smtpd_recipient_restrictions** afin d'autoriser les messages provenant des utilisateurs authentifiés :

```
smtpd_recipient_restrictions =
      permit_mynetworks
      permit_sasl_authenticated
      check_recipient_access pgsql:/etc/postfix/sql-maintain.cf
      reject_unverified_recipient
      reject_unauth_destination
      check_recipient_access
          pgsql:/etc/postfix/sql-relay-recipient-verification.cf
```

5.8 Réponses automatiques

La configuration des réponses automatiques se fait en deux étapes. Premièrement, ouvrez le fichier **/etc/postfix/main.cf** et modifiez son contenu de la manière suivante :

```
transport_maps =
        pgsql:/etc/postfix/sql-relaydomains-transport.cf
        pgsql:/etc/postfix/sql-relaydomain-aliases-transport.cf
        pgsql:/etc/postfix/sql-autoreplies-transport.cf
virtual_alias_maps = pgsql:/etc/postfix/sql-aliases.cf
        pgsql:/etc/postfix/sql-domain-aliases-mailboxes.cf,
        pgsql:/etc/postfix/sql-autoreplies.cf,
        pgsql:/etc/postfix/sql-catchall-aliases.cf
```

Ensuite, ouvrez le fichier **/etc/postfix/master.cf** et déclarez le service **autoreply** :

```
autoreply unix -       n       n       -       -       pipe
         flags= user=vmail:vmail argv=python /srv/www/modoboa_
instance/manage.py autoreply $sender $mailbox
```

5.9 Intégration avec Amavis

Lors du paramétrage d'Amavis (cf. chapitre correspondant), nous n'avons pas pu finaliser la configuration, car Postfix n'était pas encore installé. Cette contrainte étant levée, ouvrez le fichier **/etc/postfix/master.cf**, localisez la déclaration du service **smtp** (généralement en début de fichier) et ajoutez les options suivantes :

```
smtp      inet  n       -       -       -       -       smtpd
        -o smtpd_proxy_filter=inet:[127.0.0.1]:10024
        -o smtpd_proxy_options=speed_adjust
```

Nous indiquons à Postfix d'envoyer tout le trafic reçu vers le port 10024 (**amavisd**) pour analyse. Une fois traité, ce trafic doit être en mesure de retourner vers Postfix pour être acheminé. Pour ce faire, déclarez un nouveau service :

```
127.0.0.1:10025 inet n   -       n       -       -       smtpd
        -o content_filter=
        -o smtpd_authorized_xforward_hosts=127.0.0.0/8
        -o smtpd_delay_reject=no
        -o smtpd_client_restrictions=permit_mynetworks,reject
        -o smtpd_helo_restrictions=
        -o smtpd_sender_restrictions=
        -o smtpd_recipient_restrictions=permit_mynetworks,reject
        -o smtpd_data_restrictions=reject_unauth_pipelining
        -o smtpd_end_of_data_restrictions=
        -o smtpd_restriction_classes=
        -o mynetworks=127.0.0.0/8
        -o smtpd_error_sleep_time=0
        -o smtpd_soft_error_limit=1001
        -o smtpd_hard_error_limit=1000
        -o smtpd_client_connection_count_limit=0
        -o smtpd_client_connection_rate_limit=0
        -o receive_override_options=no_header_body_checks,no_unknown_recipient_checks
        -o local_header_rewrite_clients=
```

Il ne reste plus qu'à redémarrer Postfix :

```
$ /etc/init.d/postfix restart
```

6 Nginx et uWSGI

Un serveur HTTP est nécessaire pour permettre l'accès à l'interface de Modoboa. Cet article illustre l'utilisation de Nginx (question de goût personnel), mais Apache est un candidat tout aussi valable.

Nginx seul n'est pas suffisant pour interpréter les pages « Python » de Modoboa (je prends volontairement ce raccourci explicite, même s'il n'est pas tout à fait exact), il doit être couplé à un serveur WSGI (*Web Server Gateway Interface*). WSGI est une interface propre à Python et définissant les méthodes de communication entre un serveur HTTP et une application web.

Tous les composants utilisés dans cette section sont fournis en standard par la distribution Debian :

```
$ apt-get install nginx uwsgi uwsgi-plugin-python
```

6.1 Configuration de uWSGI

Créez le fichier **/etc/uwsgi/apps-available/modoboa_instance.ini** et copiez-y le contenu suivant :

```
[uwsgi]
uid = www-data
gid = www-data
plugins = python
chdir = /srv/www/modoboa_instance
module = modoboa_instance.wsgi:application
master = true
harakiri = 30
sharedarea = 4
processes = 4
vhost = true
```

Activez cette configuration en créant un lien symbolique spécifique dans le répertoire **/etc/uwsgi/apps-enabled** comme suit :

```
$ ln -s /etc/uwsgi/apps-available/modoboa_instance.ini /etc/uwsgi/apps-enabled
```

Pour finir, redémarrez uWSGI :

```
$ /etc/init.d/uwsgi restart
```

6.2 Configuration de Nginx

Nous allons configurer un hôte virtuel dédié à Modoboa. La configuration Nginx nécessaire pour ce mode de fonctionnement est très simple. Créez le fichier **/etc/nginx/site-available/modoboa_instance** et copiez-y le contenu suivant :

```
server {
    listen 80;
    server_name modoboa.example.tld;
    root /srv/www/modoboa_instance/modoboa_instance;
    access_log /var/log/nginx/modoboa.example.tld-access.log;
    error_log /var/log/nginx/modoboa.example.tld-error.log;
    location /sitestatic/ {
            autoindex on;
            alias /srv/www/modoboa_instance/modoboa_instance/
sitestatic/;
    }
    location /media/ {
            autoindex on;
            alias /srv/www/modoboa_instance/modoboa_instance/media/;
   }
```

```
    location / {
        include uwsgi_params;
        uwsgi_pass unix:/var/run/uwsgi/app/modoboa_instance/socket;
    }
}
```

Activez cette configuration en créant un lien symbolique dans le répertoire **/etc/nginx/sites-enabled/** :

```
$ ln -s /etc/nginx/sites-available/modoboa_instance /etc/nginx/
sites-enabled
```

Finalement, redémarrez Nginx :

```
$ /etc/init.d/nginx restart
```

7 Automx

Automx est un utilitaire très pratique qui permet d'automatiser la configuration des clients de messagerie (Thunderbird, Outlook, etc.). Aucun paquet Debian n'étant disponible, son installation est un peu manuelle :

```
$ wget https://github.com/sys4/automx/archive/v0.10.1.zip
$ apt-get install memcached unzip python-sqlalchemy python-
dateutil python-memcache python-ipaddr python-m2crypto
$ unzip v0.10.1.zip
$ cd automx-0.10.1
```

Copiez ensuite les fichiers nécessaires sur le système :

```
$ mkdir -p /usr/local/lib/automx
$ cp src/automx_wsgi.py /usr/local/lib/automx
$ cp -r src/automx /usr/local/lib/python2.7/dist-packages/
$ cp src/conf/automx.conf /etc/
```

7.1 Configuration

Toute la configuration d'automx se situe dans le fichier **/etc/automx.conf**. Ouvrez-le et adaptez le contenu des paramètres suivants :

```
[automx]
provider = example.tld
domains = *

[global]
backend = sql
action = settings
host = postgres://modoboauser:password@localhost/modoboa
query = SELECT first_name || ' ' || last_name AS display_name,
email FROM core_user WHERE email='%s'
result_attrs = display_name, email

smtp = yes
smtp_server = modoboa.example.tld
smtp_port = 587
smtp_encryption = starttls
smtp_auth = plaintext
smtp_refresh_ttl = 6
smtp_default = yes

imap = yes
imap_server = modoboa.example.tld
imap_port = 143
imap_encryption = starttls
imap_auth = plaintext
imap_refresh_ttl = 6
```

Nous nous contentons d'indiquer à automx la liste des domaines qu'il va servir (requête SQL d'accès à la base de Modoboa), ainsi que la configuration SMTP et IMAP qu'il enverra aux clients.

7.2 Configuration de Nginx et uWSGI

Commencez par créer le fichier **/etc/uwsgi/apps-available/automx.ini** avec le contenu suivant :

```
[uwsgi]
plugins = python
chdir = /usr/local/lib/automx
module = automx_wsgi
master = True
uid = www-data
gid = www-data
vhost = True
harakiri = 30
sharedarea = 2
processes = 2
```

Activez cette configuration en créant un lien symbolique spécifique dans le répertoire **/etc/uwsgi/apps-enabled** comme suit :

```
$ ln -s /etc/uwsgi/apps-available/automx.ini /etc/uwsgi/apps-
enabled
```

Pour finir, redémarrez uWSGI :

```
$ /etc/init.d/uwsgi restart
```

Créez ensuite le fichier **/etc/nginx/sites-available/autoconfig.example.tld** avec le contenu suivant :

```
server {
    listen 80;
    server_name autoconfig.example.tld;

    access_log /var/log/nginx/autoconfig.example.tld-access.log;
```

```
    error_log /var/log/nginx/autoconfig.example.tld-error.log;

    location /mail/config-v1.1.xml {
        include uwsgi_params;
        uwsgi_pass unix:/var/run/uwsgi/app/automx/socket ;
    }
}
```

Activez cette configuration en créant un lien symbolique dans le répertoire **/etc/nginx/sites-enabled** :

```
$ ln -s /etc/nginx/sites-available/autoconfig.example.tld /etc/
nginx/sites-enabled
```

Finalement, redémarrez Nginx :

```
$ /etc/init.d/nginx restart
```

7.3 Enregistrements DNS

L'étape finale (oui, on y arrive...) consiste à modifier votre zone DNS afin d'y ajouter les enregistrements spécifiques à l'auto-découverte. Comme d'habitude, les acteurs principaux de l'Internet ne sachant pas accorder leurs violons, une seule entrée n'est pas suffisante :

```
autoconfig     IN CNAME    modoboa
autodiscover   IN CNAME    modoboa
```

autoconfig étant la valeur attendue par Mozilla et **autodiscover** celle de Microsoft...

La syntaxe utilisée est celle de Bind et ne sera pas forcément valide chez votre registrar. Elle correspond à la création de deux alias vers l'enregistrement **modoboa.example.tld**. Vous êtes libre de créer des enregistrements de type A si vous le souhaitez.

Conclusion

Ce premier article vous a montré (rappelé pour certains) comment mettre en place un serveur de messagerie complet, doté d'une interface ergonomique et efficace de gestion. C'est une opération fastidieuse qui demande du temps et des connaissances et malheureusement, Modoboa n'est pas encore d'une grande aide sur cette partie. Cependant, cela pourrait changer dans un futur proche...

Dans mon prochain article, je vous présenterai en détails les fonctionnalités d'administration offertes par Modoboa, en espérant vous faire réaliser que vous ne pouvez plus vous en passer ! ■

LE JEU DE LA VIE ET LES LABYRINTHES

par ***Tristan Colombo***

Le jeu de la vie n'est pas vraiment un jeu, mais un automate qui évolue au cours du temps en fonction de certains paramètres et un labyrinthe... C'est un labyrinthe. Le lien entre les deux ? On peut trouver la sortie du labyrinthe grâce au jeu de la vie...

John Conway, un mathématicien anglais, a inventé le jeu de la vie en 1970. Comme dit précédemment, il ne s'agit pas vraiment d'un jeu, mais d'un automate et même, pour être plus précis, d'un automate cellulaire.

Dans cet article, je vous propose donc de partir des automates cellulaires de manière à définir et implémenter le jeu de la vie. Par la suite, nous utiliserons cet automate pour trouver la sortie d'un labyrinthe.

1 Les automates cellulaires

Un automate cellulaire, aussi noté CA de l'anglais *cellular automaton*, est un système composé de cellules élémentaires ne pouvant se trouver à un instant donné que dans un état défini par un ensemble d'états possibles. Les cellules, qui peuvent être vues comme les cellules d'un tableau, évoluent toutes en même temps, de manière synchrone. Notre automate évolue ainsi d'état en état à l'aide de règles de transition qui, pour chaque cellule, font intervenir leur voisinage.

Cet automate est donc composé de cellules et leur nombre définit sa taille et sa dimension ; on parlera de domaine. Comme vous avez pu le voir, un CA est caractérisé par des termes précis, au nombre de cinq. Après les avoir vus sommairement, je vous propose d'examiner plus précisément ce qu'ils représentent.

1.1 Le domaine

L'ensemble des cellules qui composent un CA constitue le domaine, qui peut être à une ou plusieurs dimension(s). En général, on utilise les dimensions 1 (tableau à une ligne) et 2 (tableau à deux dimensions), qui sont les plus simples à représenter.

1.2 Le voisinage

Chaque cellule de l'automate touche des cellules voisines. C'est ce qui définit le voisinage. On peut définir de nombreux voisinages, car des contraintes peuvent être appliquées pour décider quelles cellules sont vraiment voisines des autres. Dans un domaine à deux dimensions, deux types de voisinages sont particulièrement utilisés :

- le voisinage de Von Neumann, qui ne prend en compte que les cellules voisines se situant à gauche, à droite, en haut et en bas ;

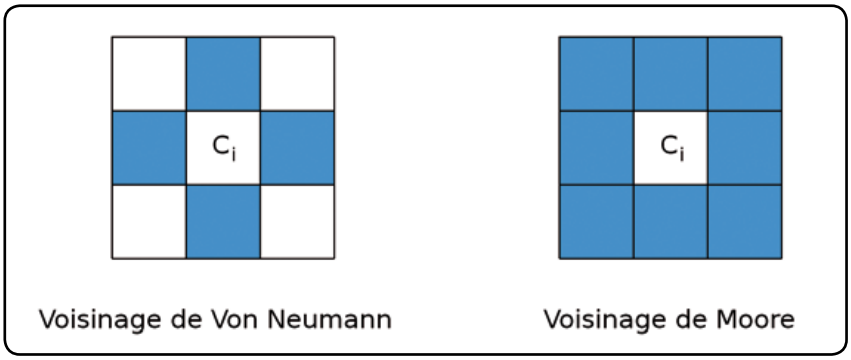


Fig. 1 : Deux types de voisinages dans un domaine à deux dimensions

- le voisinage de Moore, qui prend en compte toutes les cellules voisines.

La figure 1 page précédente montre une représentation de ces voisinages.

1.3 L'ensemble des états

Ensemble, généralement fini, des états que peut prendre chaque cellule. À chaque état de l'automate, les cellules seront dans un état dont la valeur appartient à cet ensemble. Par exemple, si l'ensemble des états vaut `{0, 1}`, alors chaque cellule ne pourra être que dans l'état `0` ou dans l'état `1`.

1.4 Les règles de transition

L'automate cellulaire évolue et ce sont les règles de transition qui dictent cette évolution. Pour calculer l'état n+1 d'un automate, on part de son état n auquel on applique les règles de transition.

1.5 Les conditions aux bords

Le domaine utilisé est souvent supposé infini, mais en pratique, il sera toujours borné. Le problème lorsque l'on a des bords, c'est que pour calculer le voisinage et appliquer les règles de transition on ne disposera pas des informations suffisantes. Il faut donc déterminer arbitrairement comment seront calculés les voisinages des cellules de bordures. Deux stratégies sont particulièrement utilisées :

- On feint un domaine infini en repliant l'espace bord à bord (on parle de configuration torique). Par exemple, si une cellule se trouve en haut du tableau, son voisinage sera calculé à l'aide des cellules du bas sur la même colonne et les colonnes adjacentes.

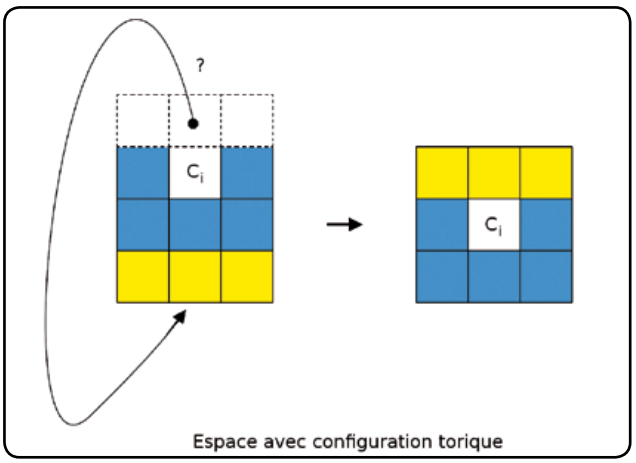


Fig. 2 : Exemple de conditions aux bords avec espace torique

- On fixe arbitrairement un état statique pour les cellules de la bordure externe.

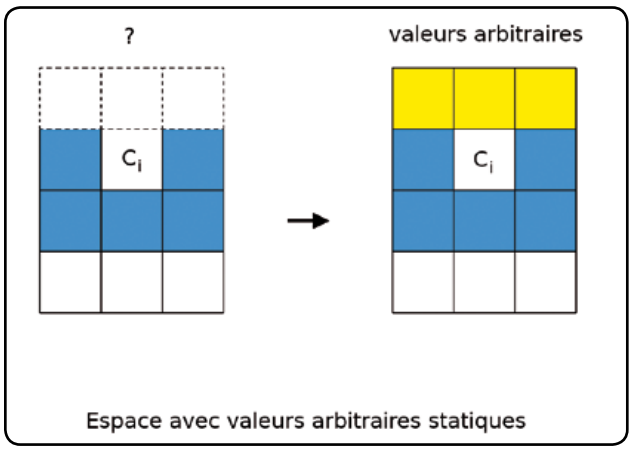


Fig. 3 : Exemple de conditions aux bords avec voisins arbitrairement fixés

Les figures 2 et 3 schématisent ces deux stratégies.

1.6 Étude de l'automate

Une fois que les cinq caractéristiques précédentes sont fixées, on étudie comment se comporte l'automate : quelles sont les suites d'états qu'il traverse. Ces suites d'états sont appelées trajectoires.

Comme le nombre de cellules est fini et que l'ensemble des états de chaque cellule également, la trajectoire des CA est périodique : à partir d'un certain état appelé transitoire de la trajectoire, nous appliquerons toujours la même sous-trajectoire suivant une période donnée.

Prenons un exemple en créant un automate cellulaire extrêmement simplifié : un domaine à une dimension composé de quatre cellules ayant pour ensemble d'états `{0, 1}`. Le tableau suivant montre les états des quatre cellules de l'automate pour quelques transitions (ses trajectoires).

État	Cellule 1	Cellule 2	Cellule 3	Cellule 4
t_0	0	0	0	0
t_1	1	1	1	1
t_2	1	0	1	0
t_3	0	1	0	1
t_4	1	0	1	0
t_5	0	1	0	1

Ici, l'état transitoire de la trajectoire est t_2 : nous pouvons remarquer que l'état t_2 est le même que l'état t_4, puis l'état t_3 est le même que t_5, et ainsi de suite. La période T est égale à 2, puisque tous les deux états nous retournons au début de notre sous-trajectoire. La figure 4 montre une représentation graphique de cet automate.

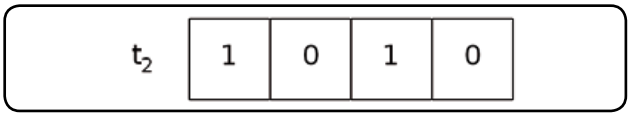


Fig. 4 : Automate à une dimension et quatre cellules

2 Les règles du jeu de la vie

L'automate cellulaire du jeu de la vie respecte des règles très simples. Nous avons vu qu'il fallait cinq éléments pour caractériser un CA, les voici pour le jeu de la vie :

- Domaine : deux dimensions et le plus grand possible ;
- Voisinage : voisinage de Moore ;
- Ensemble d'états : `{0, 1}`. Une cellule est soit morte (état `0`), soit vivante (état `1`).
- Règles de transition :
  1. Condition de naissance : une cellule morte passe à l'état vivant si et seulement si elle possède 3 cellules voisines vivantes (voir figure 5) ;
  2. Condition de survie : une cellule vivante reste dans l'état vivant si et seulement si elle possède 2 ou 3 cellules voisines vivantes (voir figure 6) ;
  3. Condition de mort : dans les autres cas, une cellule vivante passe à l'état mort et une cellule morte reste à l'état mort.
- Conditions aux bords : configuration torique.

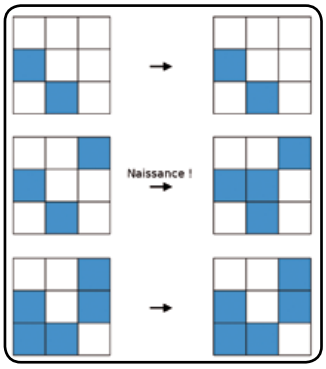


Fig. 5 : Les règles de transition du jeu de la vie : condition de naissance

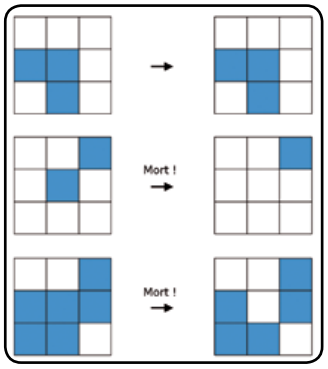


Fig. 6 : Les règles de transition du jeu de la vie : condition de survie

En fonction des conditions initiales (il faudra quand même placer quelques cellules vivantes sur le domaine), l'automate produira des structures particulières.

3 Le jeu de la vie en Python

Pour implémenter le jeu de la vie en Python, nous allons utiliser le module `curses` de manière à afficher de manière lisible l'évolution des cellules dans un terminal et, du coup, sans avoir à passer en mode graphique. Nous allons procéder par étapes.

3.1 Initialisation

Au début, nous aurons besoin d'initialiser notre environnement `curses` et l'espace qui contiendra les cellules :

```
01: import curses
02:
03:
04: def init_curses(width=20, height=20, pos=(0, 0)):
05:     curses.initscr()
06:     curses.noecho()
07:     curses.cbreak()
08:     curses.curs_set(0)
09:
10:     window = curses.newwin(width + 2, height + 2, pos[0], pos[1])
11:     window.border(0)
12:     window.keypad(1)
13:     return window
14:
15:
16: def close_curses():
17:     curses.echo()
18:     curses.nocbreak()
19:     curses.curs_set(1)
20:     curses.endwin()
21:
22:
23: def initialize(config=None, width=20, height=20):
24:     return [[0 for col in range(width)] for row in range(height)]
25:
26:
27: def display(space, win):
28:     for nline, line in enumerate(space):
29:         for ncell, cell in enumerate(line):
30:             if cell == 0:
31:                 win.addstr(nline + 1, ncell + 1, '.')
32:             else:
33:                 win.addstr(nline + 1, ncell + 1, '*')
34:
35:
36: if __name__ == "__main__":
37:     win = init_curses()
38:     space = initialize()
39:     display(space, win)
40:     win.getch()
41:     close_curses()
```

Je ne reviendrai pas sur l'utilisation du module `curses`, qui a été traité récemment dans un hors-série **[1]**. Ici, nous initialisons notre espace grâce à la fonction `initialize`

L'OPEN SOURCE AU SERVICE DES COLLECTIVITÉS AU **COTER CLUB**

17 - 18 juin

au Palais des Congrès de

Caen

avec SoLibre

Objectif libre
Infrastructure & Formations Linux

ametys
Web CMS - Intranet

BlueMind
Messagerie & espaces collaboratifs

Avencall
Communiquez librement

Merethis
Centreon editor

MAKINA CORPUS
Développons de nouveaux horizons

LINUX MAGAZINE / FRANCE
Administration et développement sur systèmes open source et embarqués

SUDOKEYS
libérez votre système d'information

CoTer Club http://www.coter-club.org

SoLibre est membre du CNLL

des lignes 23 et 24, qui renvoie une liste de listes où toutes les cellules sont initialisées à 0. Par défaut, notre espace mesure 20 cellules de long et 20 cellules de haut, mais pourra être modifié à l'aide des paramètres **width** et **height**. La fonction **display** des lignes 27 à 33 est chargée d'afficher l'espace dans notre fenêtre **curses** (appelée **win**). Pour cela, nous parcourons l'espace ligne à ligne et cellule à cellule. Pour rappel, la fonction **enumerate** appliquée à une liste renvoie un tuple contenant l'index et l'élément correspondant à cet index.

Ce programme constitue la toute première étape de notre développement : nous affichons un espace de cellules mortes. Si nous voulons que quelque chose se passe avec notre automate, il faudra commencer avec une configuration donnée de cellules vivantes. C'est à cela que va servir le paramètre **config** de la fonction **initialize** (ligne 23) que nous n'avons pas encore utilisé. Nous supposerons que **config** contient un tuple de tuples indiquant les coordonnées des cellules vivantes :

```
23: def initialize(config=None, width=20, height=20):
24:     space =  [[0 for col in range(width)] for row in range(height)]
25:     for row, col in config:
26:         space[row][col] = 1
27:     return space
```

Nous pouvons alors initialiser notre espace avec un « U » de cellules vivantes :

```
39: if __name__ == "__main__":
40:     u = ((8, 8), (8, 10), (9, 8), (9, 10), (10, 8), (10, 9), (10, 10))
41:
42:     win = init_curses()
43:     space = initialize(u)
44:     display(space, win)
45:     win.getch()
46:     close_curses()
```

Ce code donne l'affichage montré en figure 7.

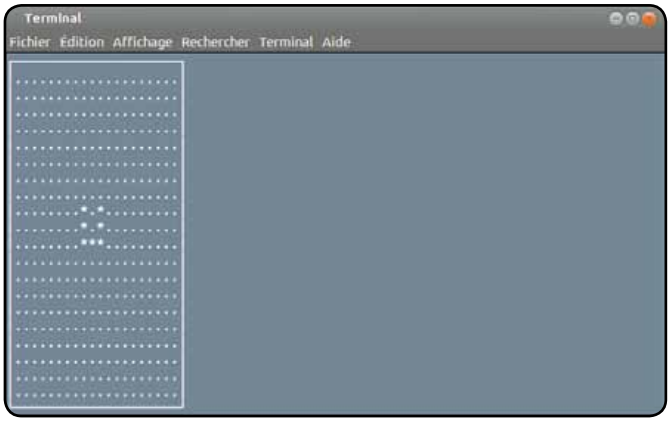


Fig. 7 : Initialisation de l'espace avec un « U » de cellules vivantes

3.2 Calcul du voisinage

Le calcul du voisinage est simple pour toute cellule n'appartenant pas à la bordure, puisqu'il suffit de compter le nombre de cellules adjacentes à l'état 1. Pour les cellules de la bordure, il faudra replier l'espace et donc, aller chercher les valeurs des cellules se trouvant de « l'autre côté » du tableau. Si nous faisons les tests pas à pas, cela sera relativement long à écrire... Nous allons ruser en définissant une sous-fonction qui prendra en paramètres des indices relatifs et les convertira en indices « réels » sur le tableau replié.

Par exemple, si nous disposons d'un tableau de 10 lignes et 10 colonnes et que nous souhaitons connaître la valeur de la case **(-1, 0)**, alors notre fonction traduira cela en **(9, 0)**. De même, pour **(-1, -1)** nous obtiendrons **(9, 9)**.

```
39: def neighbours(space, row, col):
40:     max_border_row = len(space) - 1
41:     max_border_col = len(space[0]) - 1
42:
43:     def neighbour(row, col):
44:         if row == -1:
45:             row = max_border_row
46:         elif row > max_border_row:
47:             row = 0
48:         if col == -1:
49:             col = max_border_col
50:         elif col > max_border_col:
51:             col = 0
52:
53:         return space[row][col]
54:
55:     return neighbour(row, col - 1) + neighbour(row - 1, col -1) + \
56:         neighbour(row - 1, col) + neighbour(row - 1, col + 1) + \
57:         neighbour(row, col + 1) + neighbour(row + 1, col + 1) + \
58:         neighbour(row + 1, col) + neighbour(row + 1, col - 1)
```

Dans les lignes 40 et 41, nous stockons la taille maximale d'une ligne et d'une colonne de manière à ne pas avoir à les recalculer constamment. Dans les lignes 43 à 53, nous définissons notre sous-fonction qui renvoie la valeur d'une cellule en fonction d'indices relatifs. Les lignes 55 à 58 sont alors très simples, puisqu'il suffit d'additionner les valeurs des cellules autour de la cellule cible en tournant dans le sens trigonométrique (sens inverse des aiguilles d'une montre) et en partant de la cellule de gauche.

3.3 Transitions

Nous allons ici parcourir l'ensemble des cellules du tableau et définir en fonction du nombre de cellules voisines vivantes l'évolution des cellules. Pour ne pas biaiser cette évolution en modifiant directement l'état des cellules dans le tableau, nous allons créer un nouveau tableau qui prendra la place du précédent à la fin de l'opération.

```
61: def transition(space, width=20, height=20):
62:     space_next = [[0 for col in range(width)] for row in range(height)]
63:     for row in range(height):
64:         for col in range(width):
65:             n = neighbours(space, row, col)
66:             if space[row][col] == 0 and n == 3:
67:                 space_next[row][col] = 1
68:             elif space[row][col] == 1 and (n == 2 or n == 3):
69:                 space_next[row][col] = 1
70:     return space_next
```

En ligne 62, nous créons un nouveau tableau ne contenant que des cellules mortes (à l'état 0), puis nous parcourons l'ensemble des cellules grâce aux deux boucles imbriquées des lignes 63 et 64 avant d'appliquer les règles de transition définies précédemment : naissance dans les lignes 66 et 67, survie dans les lignes 68 et 69 et mort dans les autres cas (comme nous avons initialisé toutes les cellules à 0, il n'y a rien à faire). À la fin de la fonction, nous renvoyons le tableau mis à jour.

3.4 Lancement du code

Pour lancer notre programme, il va simplement falloir afficher le tableau puis appliquer les transitions et cela en boucle jusqu'à ce que l'utilisateur décide d'arrêter. Dans cette implémentation, c'est l'utilisateur qui décide d'appliquer les règles de transition en appuyant sur une touche quelconque. Pour quitter le programme, il faut appuyer sur la touche **<Esc>** (code ASCII 27).

```
72: if __name__ == "__main__":
73:     u = ((8, 8), (8, 10), (9, 8), (9, 10), (10, 8), (10, 9), (10, 10))
74:
75:     win = init_curses()
76:     space = initialize(u)
77:     while True:
78:         display(space, win)
79:         key = win.getch()
80:         if key == 27:
81:             break
82:         space = transition(space)
83:     close_curses()
```

La figure 8 montre un résultat obtenu avec ce code légèrement modifié de manière à afficher les cellules vivantes en rouge et masquer les cellules mortes.

4 La problématique du labyrinthe

Un labyrinthe est composé de murs et de couloirs, que l'on peut représenter sous la forme d'un tableau où les cellules valant 0 sont accessibles et les cellules valant 1 sont

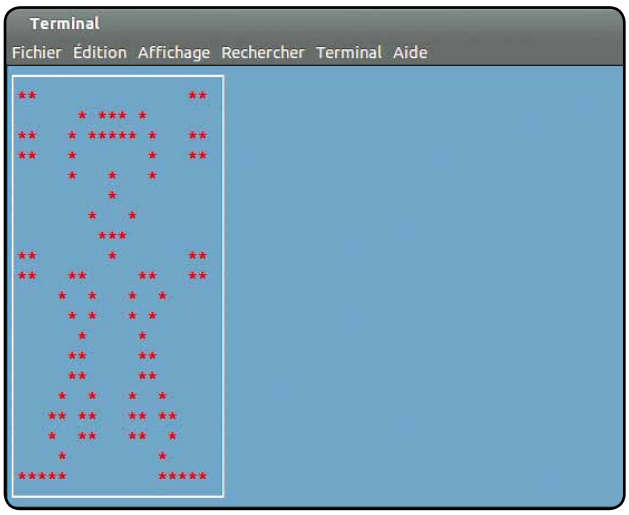


Fig. 8 : Exemple d'évolution de l'automate en partant d'un « U » de cellules vivantes

inaccessibles (ce sont des murs). Un labyrinthe de **row** lignes et **col** colonnes possède également obligatoirement une entrée et une sortie. Nous considérerons que l'entrée se situe en **(0, 0)** (coin supérieur gauche) et la sortie en **(row - 1, col - 1)** (coin inférieur droit). Les bordures du labyrinthe seront infranchissables et nous ajouterons donc un mur tout autour.

Voici un exemple de codage et d'affichage d'un labyrinthe :

```
01: def display(lab):
02:     print("*" * (len(lab) + 2))
03:     for row in lab:
04:         print("*", end="")
05:         for cell in row:
06:             if cell == 1:
07:                 print("*", end="")
08:             else:
09:                 print(" ", end="")
10:         print("*")
11:     print("*" * (len(lab) + 2))
12:
13:
14: if __name__ == "__main__":
15:     lab = [
16:         [0, 0, 0, 1, 0, 0, 1, 0, 0, 0],
17:         [1, 1, 0, 1, 0, 1, 1, 0, 1, 0],
18:         [0, 0, 0, 0, 0, 1, 0, 0, 1, 0],
19:         [0, 0, 0, 1, 1, 1, 1, 0, 1, 0],
20:         [1, 1, 0, 1, 0, 0, 0, 0, 1, 0],
21:         [0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
22:         [0, 0, 0, 0, 0, 1, 1, 1, 1, 0],
23:         [1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
24:         [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
25:         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
26:     ]
27:
28:     display(lab)
```

Au lancement, nous obtiendrons des caractères où les murs seront représentés par des étoiles :

```
************
*  *  *  *
*** * ** * *
*    *  * *
*  **** * *
*** *   * *
*  * *   *
*   **** *
******* *
*       ***
*         *
************
```

5 Un automate cellulaire pour sortir d'un labyrinthe

Pour trouver la sortie de ce labyrinthe, nous allons créer un automate cellulaire dont voici les caractéristiques :

- Domaine : deux dimensions de la taille souhaitée pour le labyrinthe. Nous poursuivrons sur notre exemple et utiliserons donc un tableau de 10 x 10 cases ;
- Voisinage : voisinage de Von Neumann (on ne peut se déplacer qu'horizontalement ou verticalement) ;
- Ensemble d'états : `{-1, 0, 1, 2, 3, 4, 5}`. Il s'agit ici d'un codage que nous allons expliquer :
  - `-1` : case de sortie du labyrinthe (nous avons dit que par défaut il s'agirait de la case du coin inférieur droit, soit (9, 9)) ;
  - `0` : case libre sur laquelle on peut se déplacer ;
  - `1` : case occupée par un mur (déplacement impossible) ;
  - `2`, `3`, `4`, `5` : indication de déplacement pour trouver la sortie :
    - → `2` : aller vers la gauche ;
    - → `3` : aller vers la droite ;
    - → `4` : aller vers le haut ;
    - → `5` : aller vers le bas.
- Règles de transition :
  1. Si une cellule est dans l'état case libre (code `0`) et que l'une de ses voisines est dans l'état sortie (code `-1`), alors la cellule prend pour valeur le code correspondant à la direction à suivre pour atteindre la sortie (voir figure 9) ;
  2. Si une cellule est dans l'état case libre (code `0`) et que l'une de ses voisines est dans l'état déplacement (code `2`, `3`, `4`, ou `5`), alors la cellule prend pour valeur le code correspondant à la direction à suivre pour atteindre la cellule de déplacement (voir figure 10) ;
  3. Dans tous les autres cas, l'état de la cellule reste inchangé.
- Conditions aux bords : les bords sont infranchissables et seront assimilés à un mur.

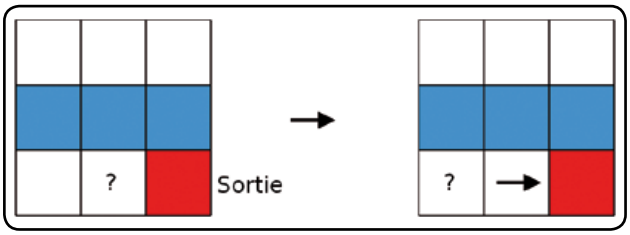


Fig. 9 : Exemple de transition : voisinage de la case de sortie

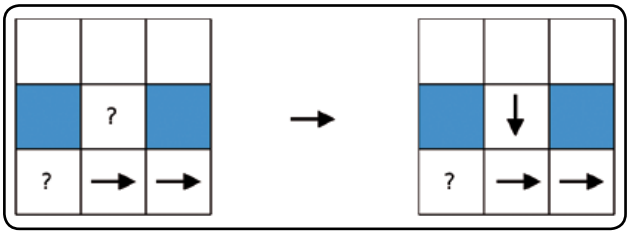
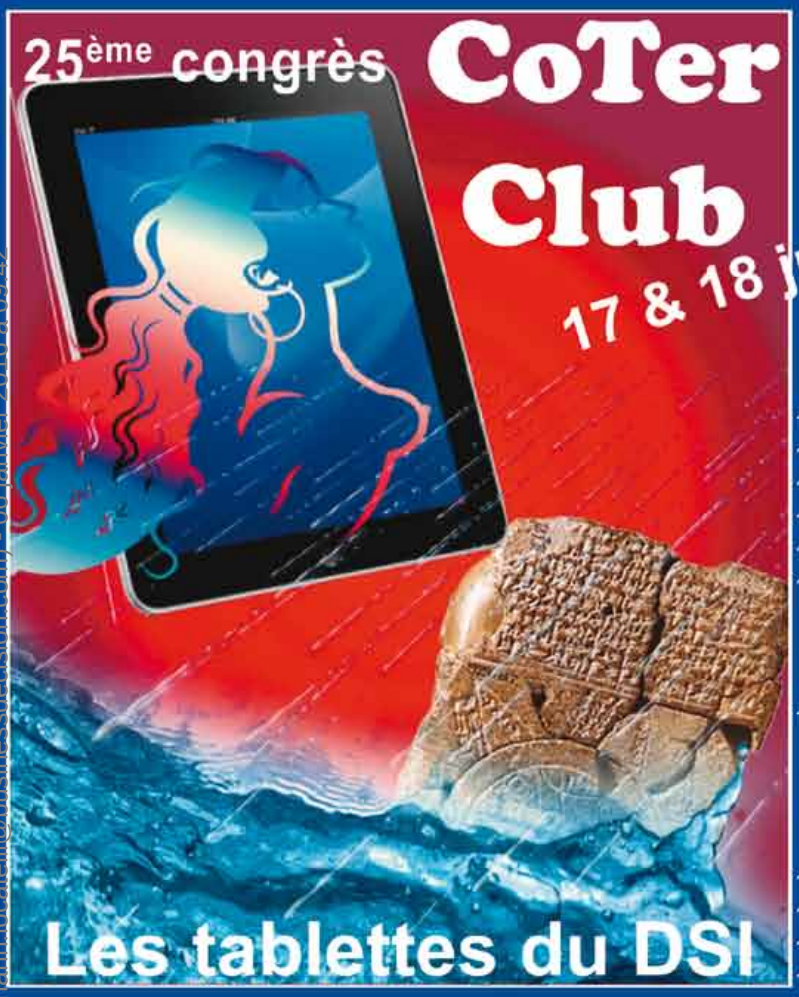


Fig. 10 : Exemple de transition : voisinage d'une case de déplacement

L'automate va remplir progressivement l'ensemble des cases libres du tableau et indiquer la direction à suivre pour atteindre la sortie. Pour implémenter notre algorithme, nous avons deux possibilités : soit utiliser les caractéristiques de l'algorithme sans réfléchir et donc, parcourir à chaque étape toutes les cases du tableau pour appliquer les règles de transition, soit essayer de coder les choses « intelligemment ».

Nous savons que l'état d'une cellule ne peut être modifié que si une de ses voisines est la sortie ou contient une indication de direction... Inutile donc de parcourir tout le tableau : il suffit de maintenir à jour une liste de cellules susceptibles d'être modifiées. Lorsque cette liste permet d'atteindre la case (0, 0) qui est l'entrée du labyrinthe, nous arrêterons nos calculs puisque nous aurons la solution. Nous allons modifier la fonction `display` de manière à ce qu'elle affiche non seulement le labyrinthe, mais également le chemin menant à la sortie et qui sera codé par des points (et un `X` pour la sortie).



CAEN Normandie

Mardi 17 juin

9h00 : Accueil des congressistes
9h30 : Accueil par la Présidente et le Maire de Caen.
10h00 : Atelier 1 : « **la DSI, service stratégique ?** »
11h30 : Visite des stands
12h30 : Repas
14h00 : Atelier 2 : « **Patrimoin'data** »
15h30 : Visite des stands
17h00 : Intervention de **Louis Naugès : La R2I, Révolution Industrielle Informatique, ou comment remettre la DSI au cœur de l'innovation dans les Systèmes d'Information**
19h30 : Soirée de Gala

Mercredi 18 juin

8h30 : Assemblée Générale Ordinaire du CoTer Club
9h00 : Rapid'Démos
10h30 : Visite des stands
12h30 : Cocktail déjeunatoire sur les stands
14h00 : Atelier 3 : « **Retour d'expériences** »
15h30 : Visite des stands
17h00 : Fin du congrès

Le CoTer Club, association de DSI, est une structure indépendante de réflexion au service des Collectivités Territoriales.
Chaque année elle organise un congrès en partenariat avec les acteurs majeurs du secteur des CollectivitésTerritoriales.

Tout au long de l'année, l'association organise mensuellement, entre chaque congrès, des groupes de travail afin d'étudier les nombreuses problématiques auxquelles sont confrontées les DSI.
Il publie aussi régulièrement une newsletter sur l'actualité informatique en collectivités territoriales.
Cette année, le 25ème congrès, qui se déroulera à Caen, les 17 et 18 juin 2014, s'interrogera, entre autres, dans la mouvance technologique actuelle, sur la vision que porte la Direction Générale d'une collectivité sur sa DSI !

Et pour la deuxième année consécutive, le CoTer Club organise la veille du congrès un tournoi de golf, entre partenaires et collectivités.
Tous les détails pour vous inscrire sur notre site : www.coter-club.org
Venez nombreux pour partager aussi un moment de convivialité.



```
01: def display(lab):
02:     print("*" * (len(lab) + 2))
03:     for row in lab:
04:         print("*", end="")
05:         for cell in row:
06:             if cell == 1:
07:                 print("*", end="")
08:             elif cell == "." or cell == "X":
09:                 print(cell, end="")
10:             else:
11:                 print(" ", end="")
12:         print("*")
13:     print("*" * (len(lab) + 2))
```

Le calcul des voisins, basé sur le voisinage de Von Neumann, est simple à réaliser. Nous ajoutons une liste qui contient les coordonnées des cases voisines à l'état vide et qu'il faudra donc explorer.

```
16: def neighbours(lab, row, col):
17:     next = []
18:
19:     if row != 0:
20:         if lab[row - 1][col] == 0:
21:             next.append((row - 1, col))
22:         elif lab[row - 1][col] != 1:
23:             lab[row][col] = 5
24:
25:     if col != 0:
26:         if lab[row][col - 1] == 0:
27:             next.append((row, col - 1))
28:         elif lab[row][col - 1] != 1:
29:             lab[row][col] = 2
30:
31:     if row != len(lab) - 1:
32:         if lab[row + 1][col] == 0:
33:             next.append((row + 1, col))
34:         elif lab[row + 1][col] != 1:
35:             lab[row][col] = 4
36:
37:     if col != len(lab[0]) - 1:
38:         if lab[row][col + 1] == 0:
39:             next.append((row, col + 1))
40:         elif lab[row][col + 1] != 1:
41:             lab[row][col] = 3
42:
43:     if lab[row][col] != 0:
44:         if row == 0 and col == 0:
45:             return False
46:         else:
47:             return next
48:     else:
49:         return []
```

À la fin de la fonction, nous renvoyons la liste des cases à explorer (ligne 47), ou une liste vide si l'état de la case courante n'a pas changé (ligne 49). Si la case courante correspond à l'entrée dans le labyrinthe (case **(0, 0)**), nous renvoyons une valeur spéciale **False** qui interrompra la boucle de traitement.

Nous savons, pour chaque cellule, déterminer quelle direction prendre pour s'approcher de la sortie. Mais cela sera codé sous forme de chiffres dans le tableau représentant le labyrinthe. Il faut décoder le chemin pour pouvoir l'afficher de manière claire. Ce sera le rôle de la fonction **display_path** :

```
52: def display_path(lab):
53:     row = 0
54:     col = 0
55:
56:     while lab[row][col] != -1:
57:         d, lab[row][col] = lab[row][col], "."
58:         if d == 2:
59:             col -= 1
60:         elif d == 3:
61:             col += 1
62:         elif d == 4:
63:             row += 1
64:         else:
65:             row -= 1
66:
67:     lab[row][col] = "X"
68:     display(lab)
```

Ici, nous partons simplement de l'entrée du labyrinthe en **(0, 0)** comme le montre l'initialisation des variables **row** et **col** en lignes 53 et 54, puis à l'aide des indications contenues dans chaque cellule, nous progressons dans le labyrinthe et marquons le chemin par des points (ligne 57). Notez l'emploi de l'opérateur **,** qui permet de réaliser la double affectation :

```
d = lab[row][col]
lab[row][col] = "."
```

À la fin de la fonction, nous faisons appel à la fonction **display** pour afficher effectivement le labyrinthe et le chemin.

Le programme principal consistera à parcourir depuis la sortie la liste des cellules susceptibles d'être modifiées :

```
71: if __name__ == "__main__":
72:     lab = [
73:         [0, 0, 0, 1, 0, 0, 1, 0, 0, 0],
74:         [1, 1, 0, 1, 0, 1, 1, 0, 1, 0],
75:         [0, 0, 0, 0, 0, 1, 0, 0, 1, 0],
76:         [0, 0, 0, 1, 1, 1, 1, 0, 1, 0],
77:         [1, 1, 0, 1, 0, 0, 0, 0, 1, 0],
78:         [0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
79:         [0, 0, 0, 0, 0, 1, 1, 1, 1, 0],
80:         [1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
81:         [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
82:         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```
83:     ]
84:     exit = (9, 9)
85:     lab[exit[0]][exit[1]] = -1
86:
87:     display(lab)
88:
89:     next = [(9, 8)]
90:     while True:
91:         current = next.pop()
92:         cells = neighbours(lab, current[0], current[1])
93:         if cells is False:
94:             break
95:         next = cells + next
96:
97:     display_path(lab)
```

Nous retrouvons la définition du labyrinthe dans les lignes 72 à 83, la case de sortie dans les lignes 84 et 85 et la liste des cellules à explorer en ligne 89 (il s'agit de l'ensemble des cases vides autour de la sortie... Ici, il n'y en a qu'une : la case **(9, 8)**). Ensuite, la boucle des lignes 90 à 95 parcourt la liste des cellules « intéressantes » et calcule leur nouvelle valeur, ainsi que les nouvelles cellules à étudier. En cas d'obtention de la valeur **False** (lignes 93 et 94), on quitte la boucle : le chemin de **(0, 0)** jusqu'à la sortie a été trouvé.

Ce code peut bien sûr être optimisé et il faudrait traiter les cas d'erreur. Par exemple, que se passe-t-il s'il n'est pas possible d'aller de l'entrée jusqu'à la sortie ?

Conclusion

Nous avons vu ici des automates cellulaires simples, mais il est possible de résoudre des problèmes complexes grâce à eux en les appliquant, par exemple, à des problèmes de flux. Nous avons également pu voir que des contraintes locales (les voisinages) entraînent parfois des comportements généraux tout à fait imprévisibles. Testez le jeu de la vie avec des conditions initiales différentes et vous verrez apparaître des comportements étonnants, dont certains ont été baptisés. Vous pourrez ainsi rencontrer une multitude de structures : des canons, des jardins d'Eden, ou encore des vaisseaux spatiaux. Tout est possible, tout est réalisable... ■

Référence

[1] Colombo T., « Interface console améliorée », GNU/Linux Magazine hors série n°71, mars/avril 2014, p. 88 à 101.

INSIDE ANDROID : KIT KAT 4.4

par **Benjamin Zores** *[architecte Android @ Alcatel-Lucent]*

Continuons sur notre lancée avec une plongée dans les entrailles de la dernière version d'Android (4.4), la bien nommée Kit Kat.

Annoncée en grande pompe marketing, en association avec Nestlé, la tant attendue version « K » d'Android, prévue pour être la désormais célèbre version 5.0 sous le nom de *Key Lime Pie*, est finalement disponible depuis le 31 octobre 2013 en version 4.4 également appelée *Kit Kat* (KK). Comme désormais à l'accoutumée, les changements sont nombreux et profonds et le code source s'en ressent. Les sources de l'AOSP voient en effet la balance grimper de **361 à 407 projets** (ou composants Git) et de **8 à 10 Go** (+25%) depuis Jelly Bean 4.3 ! Voyons donc sans plus tarder ce qu'il y a sous le capot.

1 Projet « Svelte »

Rappelez-vous, Jelly Bean a connu le projet « Butter » (ou « beurre »), qui visait à rendre le système Android plus fluide et plus réactif (ce qui a été le cas). Mais en contrepartie, la consommation de ressources matérielles a grimpé, rendant d'autant plus difficile la faculté d'anciens terminaux à migrer vers Jelly Bean et augmentant d'autant la fragmentation des différentes versions de l'OS.

Kit Kat introduit donc le projet « Svelte », qui a pour but d'affiner quelque peu Android afin de le rendre disponible sur une plus grande majorité de terminaux. Le but établi est clairement de pouvoir le faire tourner sur des téléphones disposant de moins de 512 Mo de mémoire (quand même !). Comme le dit si bien Dave Burke, à la tête de l'ingénierie Google : « J'ai conçu le projet Butter pour rendre le système plus réactif. Le fait est, le beurre alourdit. J'ai ainsi conçu le projet Svelte pour perdre du poids. Au final, ma contribution à Android est proche de zéro ;-) ».

Pour arriver à affiner le système, les développeurs ont eu une idée hélas trop rare de nos jours : se mettre à la place de l'utilisateur et travailler sur un cas réel. Google a donc fourni à l'ensemble de ses développeurs un Nexus 4, mais dans une configuration plus modeste que l'original. Ainsi, le téléphone a vu sa mémoire réduite d'1 Go à 512 Mo de mémoire, sa résolution d'écran 1280x768 réduite à 960x540 pixels, deux de ses quatre cœurs ont été coupés et les deux restants ont vu leur fréquence réduite à la baisse. Le système était difficilement utilisable et il a fallu faire en sorte qu'il le devienne. Plusieurs points d'action ont alors été envisagés :

- Réduire l'empreinte mémoire du système ;
- Réduire l'empreinte mémoire des applications « maison » (Google Experience) ;
- Améliorer la réaction des applications en cas de défaut mémoire ;
- Fournir aux développeurs de meilleurs moyens de mesure et d'instrumentation leur permettant d'être conscients de l'empreinte mémoire de leurs applications.

1.1 Meilleure gestion mémoire

Kit Kat améliore sensiblement la gestion mémoire globale du système, empêchant les applications d'être trop gourmandes. Lorsque de nombreux services démarrent, Android les lancent désormais de manière sérialisée, afin d'éviter les pics induits par une exécution parallélisée. Le système met maintenant également fin aux processus devenus trop lourds et qui restent en sommeil.

Au niveau noyau, Android propose désormais la fonctionnalité KSM (*Kernel Same-page Merging*), introduite avec

Linux 2.6.32, et qui permet à des processus de partager des pages mémoire. Le noyau scanne ainsi la mémoire pour y trouver des pages identiques, et essaie de les partager entre processus au moyen d'une opération de type COW (*copy-on-write*). Si cela fait certes gagner en mémoire, cela a un coût en temps CPU (ce qui, du coup, peut également avoir un effet néfaste sur votre batterie). Seules les pages marquées comme **MADV_MERGEABLE** par les applications peuvent cependant être éligibles. Si vous souhaitez activer KSM, votre noyau devra évidemment être configuré de la sorte et les lignes suivantes devront être ajoutées à votre fichier **init.rc** :

```
write /sys/kernel/mm/ksm/pages_to_scan 100
write /sys/kernel/mm/ksm/sleep_millisecs 500
write /sys/kernel/mm/ksm/run 1
```

Une autre fonctionnalité noyau désormais supportée par Android est la capacité à utiliser de la mémoire compressée comme zone de swap (*Swap to ZRAM*). Comme pour tout système Linux un peu juste côté mémoire, l'utilisation de swap permet de lancer davantage de processus. Mais sur un système embarqué, il serait peu judicieux de swapper sur une flash eMMC. Aussi, Android permet désormais de compresser une partie de la mémoire pour gagner d'autant. Mais encore une fois, comme pour KSM, cette fonctionnalité est consommatrice de temps CPU et son usage réduira d'autant la durée de vie de votre batterie. Pour activer cette dernière, vous devrez ajouter la ligne suivante à votre fichier **fstab** :

```
/dev/block/zram0 none swap defaults zramsize=<size in bytes>,swapprio=<swap
partition priority>
```

Ainsi que la ligne suivante à votre fichier **init.rc** :

```
swapon_all /fstab.X
```

Le paramètre **zramsize** détermine quelle taille de votre mémoire doit être allouée à zram. Des taux de compression de l'ordre de 30 à 50% sont généralement observés. Le paramètre **swapprio** n'a d'intérêt que si vous disposez de plus d'un espace de swap.

Si les cgroups mémoire sont activés dans votre configuration noyau, l'ActivityManager d'Android désignera les threads de faible priorité comme étant plus éligibles au swap que les autres.

Kit Kat apporte également une nouvelle API appelée **ActivityManager.isLowRanDevice()**, permettant aux applications de déterminer si elles tournent sur un terminal à faibles performances et éventuellement de désactiver certaines fonctionnalités applicatives. Pour les terminaux disposant de 512 Mo de mémoire ou moins, il est nécessaire pour les constructeurs de le déclarer au sein du fichier de configuration **BoardConfig.mk** par le paramètre suivant :

```
PRODUCT_PROPERTY_OVERRIDES += ro.config.low_ram=true
```

Enfin, Kit Kat intègre une meilleure gestion des seuils du *Low Memory Killer*, de plus petits caches pour les buffers graphiques et, en utilisant le noyau 3.4, permet de réduire la pression sur **kswapd** lorsque le système est en défaut de page.

1.2 Réduction de l'empreinte mémoire système

Kit Kat apporte son lot d'optimisations pour réduire la consommation mémoire globale du système. On notera ainsi les améliorations suivantes :

- Dégraissage des processus **system_server** et **SystemUI** (responsable de l'affichage), permettant de gagner plusieurs Mo de mémoire ;
- Pré-chargement des caches DEX au sein de la machine virtuelle Dalvik (DVM) ;
- Remplacement au sein du framework Java des objets **HashMap** et **HashSet** par **ArrayMap** et **ArraySet**, dont l'implémentation est bien plus légère ;
- Réduction du cache utilisé pour la gestion des fontes ;
- Possibilité de désactiver la capacité de compilation JIT (*Just-In-Time*) de Dalvik. L'usage de JIT consomme en moyenne 100 à 200 Ko de mémoire supplémentaire par application. La taille maximale du cache de code peut ainsi désormais être configurée. Une désactivation complète de ce dernier permet d'économiser entre 3 et 6 Mo de mémoire sur le système. Ceci se fait simplement en déclarant la variable suivante au sein du fichier **BoardConfig.mk** :

```
PRODUCT_PROPERTY_OVERRIDES += dalvik.vm.jit.codecachesize=0
```

1.3 MemTrack

Kit Kat dispose d'un nouvel élément au sein de sa couche d'abstraction matérielle (HAL) permettant de mieux suivre l'utilisation mémoire et l'allocation

de surfaces graphiques. Vous trouverez la description de cette nouvelle HAL au sein du fichier d'entête **hardware/libhardware/include/hardware/memtrack.h**. Cette dernière permet de suivre la mémoire utilisée par une ressource matérielle et requiert donc l'usage de la HAL pour s'interfacer directement avec le matériel. Un exemple simple est celui d'une texture mémoire, qui est allouée par un processus au sein de la mémoire graphique, mais qui n'est pas pour autant mappée dans l'espace d'adressage de ce processus. L'autre intérêt est de pouvoir trier l'utilisation de la mémoire graphique en différentes catégories.

Plusieurs catégories sont supportées :

- **MEMTRACK_TYPE_OTHER**
- **MEMTRACK_TYPE_GL**
- **MEMTRACK_TYPE_GRAPHICS**
- **MEMTRACK_TYPE_MULTIMEDIA**
- **MEMTRACK_TYPE_CAMERA**

Les statistiques d'utilisation mémoire par un processus donné peuvent alors être récupérées par l'appel à la fonction suivante :

```
getMemory(<pid>, MEMTRACK_TYPE_GL)
```

1.4 ProcStats

L'optimisation du système et de ses applications faite, place maintenant aux applications tierces. Il a fallu pour cela doter les développeurs des moyens d'optimiser leurs applications. C'est chose faite avec **procstats**, un utilitaire permettant de tracer l'utilisation mémoire de chaque application. L'utilitaire permet de suivre l'évolution de la courbe mémoire au fur et à mesure du temps, de se rendre compte des temps d'exécution et de suivre l'empreinte mémoire globale des applications et des services en tâches de fond.

procstats peut être lancé en version graphique à travers le menu ***Paramètres > Options pour Développeurs > Statistiques des Processus > Votre Application***. Pour les vrais barbus, il peut (et doit) être lancé simplement via la commande suivante :

```
adb shell dumpsys procstats --details
```

Les informations relatives aux différents processus sont récupérées automatiquement du système sans aucune intervention de l'utilisateur ou du développeur. Aucun hook ou profileur n'est nécessaire.

Considérons maintenant une application disposant de deux services (**PremierService** et **SecondService**), qui sont démarrés au lancement de l'application. Voici un exemple d'informations retournées par **procstats** :

```
* com.test.procstats / u0a51:
  * com.test.procstats / u0a51:
    TOTAL: 100% (4.4MB-5.0MB-6.1MB/3.0MB-3.1MB-3.1MB over 3)
    Top: 1.7% (6.1MB-6.1MB-6.1MB/3.1MB-3.1MB-3.1MB over 1)
    Service: 90% (4.4MB-4.4MB-4.4MB/3.0MB-3.0MB-3.0MB over 2)
    Service Rs: 8.1%
  * com.test.procstats.PremierService:
    Process: com.test.procstats
    Running count 2 / time 0.23%
    Started count 1 / time 0.23%
    Executing count 2 / time 0.01%
  * com.test.procstats.SecondService:
    Process: com.test.procstats
    Running count 1 / time 92%
    Started count 1 / time 92%
    Executing count 1 / time 0.01%
```

Le premier service a consommé 0,23% du temps d'exécution de notre application et le second 92%. Comme vous vous en rendez compte, nos deux services consomment bien plus que le reste de notre application, et en tout cas, bien plus qu'il ne nous semblait. Mais voyons maintenant quelle durée est représentée par ces pourcentages, au moyen de la commande suivante :

```
adb shell dumpsys -a
```

Et son résultat :

```
* com.test.procstats / u0a51:
  Process com.test.procstats (3 entries):
  Screen On  / Norm / Top       : +30s259ms
* com.test.procstats / u0a51:
  Process com.test.procstats (3 entries):
  Screen On  / Norm / Top       : +30s259ms
    Service   : +26m35s118ms (running)
        Service Rs: +2m23s130ms
        TOTAL     : +29m28s507ms
  # [...]
  mActive=true
  mNumActiveServices=1 mNumStartedServices=1
  Service com.test.procstats.PremierService:
  Process: com.test.procstats
  Running op count 2:
  Screen On  / Norm / +4s116ms
    TOTAL: +4s116ms
  # [...]
  Service com.test.procstats.SecondService:
  Process: com.test.procstats
  Running op count 1:
  Screen On  / Norm / +27m4s66ms (running)
    TOTAL: +27m4s66ms
```

Et voilà, **PremierService** a tourné pendant 4s et **SecondService** pendant plus de 27mn !! De plus, **mNumActiveServices** vaut encore 1, alors qu'il devrait être à 0. Notre service tourne donc encore ! Vous l'aurez compris, grâce à ce nouvel outil, les développeurs vont enfin pouvoir comprendre précisément ce que fait réellement leur application (et pourquoi la durée de vie de votre batterie est en chute libre...).

2 Android RunTime (ART)

Outre le projet Svelte, l'autre fonctionnalité majeure apparue en toute discrétion au sein de Kit Kat se prénomme ART (pour *Android RunTime*), une nouvelle machine virtuelle, encore expérimentale (et donc non activée par défaut). Le but de cette dernière est de remplacer la désormais célèbre machine Dalvik. Encore au stade de développement, cette dernière peut d'ores et déjà être essayée par les développeurs applicatifs (et les utilisateurs d'ailleurs). Il suffit de valider son utilisation au sein du menu ***Paramètres > Options pour Développeurs***, comme le montre la figure 1.

Fig. 1 : Menu de sélection de la machine virtuelle (Dalvik ou ART)

Notez cependant que ART n'est pas encore 100% compatible avec Dalvik et l'activer risque d'empêcher certaines applications de fonctionner (personnellement je l'utilise sans le moindre soucis). ART est un projet gardé secret par Google (comme à l'accoutumée) depuis maintenant plus de deux ans et visant à améliorer les performances des terminaux Android. Le projet part du principe que les périphériques d'aujourd'hui sont autrement plus performants en terme de CPU et de mémoire que la première génération de périphériques Android. Les contraintes ont donc évoluées.

Les sources du projet ART sont disponibles dans l'AOSP au sein du répertoire **libcore/libart** et les deux machines virtuelles peuvent coexister sur votre terminal (Dalvik conserve son nom, soit **libdvm.so**, tandis qu'ART est désormais disponible sous le nom de **libart.so**). Il est très facile de choisir à la compilation si l'une, l'autre ou les deux implémentations doivent être installées sur votre terminal. Pour ce faire, modifiez le fichier **build/target/product/core_minimal.mk**

pour y faire figurer la valeur correspondante au sein de la nouvellement introduite variable **PRODUCT_RUNTIMES** :

```
PRODUCT_RUNTIMES := runtime_libdvm_default
PRODUCT_RUNTIMES += runtime_libart
```

Mais voyons pour les différences principales entre ces deux machines virtuelles (VM). ART versus Dalvik se résume principalement à une implémentation *Ahead-of-Time* (AOT) versus *Just-in-Time* (JIT). Les applications Android sont écrites en Java et pré-compilées en byte-code Dalvik, de manière à être portables, contrairement à du code natif. Mais pour s'exécuter sur votre téléphone, ce byte-code doit pourtant être converti en code machine.

L'approche JIT permet de convertir les portions de code prêtes à être exécutées au moment précis où l'application va en avoir besoin. Au fur et à mesure de l'exécution de votre application, davantage de code sera compilé (dans la limite de la taille de cache de code disponible, comme vu précédemment). Comme seule une partie du code se retrouve compilée, l'empreinte mémoire reste faible et l'espace disque utilisé par votre application reste moindre (le code compilé est stocké en mémoire flash).

Par opposition, l'approche AOT consiste à compiler le byte-code Dalvik avant que cela ne soit nécessaire. L'intégralité du byte-code de votre application sera ainsi converti en langage machine dès l'installation de votre application. Les avantages sont certains : le code et donc vos applications s'exécuteront désormais bien plus vite, car déjà compilés. Le lancement des applications devient alors presque immédiat et vous n'aurez plus le sentiment que le système est en train de ramer (pendant que le CPU compile l'application en tâche de fond). De plus, le code ayant été compilé une fois pour toutes, votre processeur ne sera plus jamais occupé à cette tâche. Vous gagnez donc en autonomie sur votre batterie (en toute théorie). L'inconvénient néanmoins concerne l'espace disque. Les applications grossissent et réduisent donc d'autant l'espace libre disponible pour l'utilisateur. Mais comme les prix du stockage baissent... Enfin, la taille de code (compilé) étant désormais supérieure, la consommation mémoire augmente également. Ceci va quelque peu à l'encontre du projet Svelte ;-) Enfin, les développeurs adoptant l'approche du « test & trial » (autrement appelée « je code, je lance l'application et on verra bien ;-) ») détesteront le fait que les temps d'installation s'allongent.

Le site anglophone Android Police a fait quelques essais comparatifs entre Dalvik et ART **[1]**. L'écart de performance semble être compris entre 10 et 20% en faveur de la nouvelle machine virtuelle. Si les travaux d'ART sont prometteurs, il faudra attendre de voir les bénéfices réels sur le long terme (performances, charge mémoire, conséquence sur la batterie...). Aucune date n'est aujourd'hui fixée pour le remplacement de la VM par défaut de Dalvik vers ART, l'implémentation ne permettant pas aujourd'hui de garantir une compatibilité à 100%.

3 Graphisme

Avec les améliorations majeures apportées par Jelly Bean au niveau de la couche graphique, Kit Kat fait plutôt pâle figure, mais les quelques changements restent intéressants. Ainsi, SurfaceFlinger, le composant responsable de la composition graphique, supporte désormais (et utilise) les API OpenGL ES 2.0 en lieu et place de la version 1.0 et le compositeur matériel (hwcomposer) supporte désormais la composition d'écrans virtuels, en plus de l'écran principal.

4 Nouveautés audio

Toujours dans l'optique d'améliorer l'autonomie de votre périphérique, mais également pour améliorer la qualité auditive de l'ensemble, Android supporte désormais (et ce n'est pas trop tôt !) les DSP audio, déléguant l'intégralité du traitement du signal (décodage audio, mais également application d'effets en sortie) à un chipset spécialisé plutôt que de gérer l'ensemble de manière complètement logicielle. Cela se traduit dans la HAL (cf. le fichier **hardware/libhardware/include/hardware/audio.h** des sources AOSP) par un nouveau module **codec_offload**, au même titre qu'existait les périphériques USB, HDMI ou encore A2DP :

```
#define AUDIO_HARDWARE_MODULE_ID_CODEC_
OFFLOAD "codec_offload"
```

Conférer le traitement du signal audio à un DSP a un effet bénéfique sur la batterie. Les tests effectués par Google sur un Nexus 5 montrent une augmentation de 50% de l'autonomie de la batterie lors de la délégation du traitement vers le DSP. Au niveau applicatif, la fonctionnalité est complètement transparente : le système délègue l'audio au DSP, si tant est que ce dernier soit disponible.

Au niveau applicatif, la librairie **audiofx** supporte désormais un nouvel effet audio d'augmentation des basses (*loudness enhancer*), permettant

d'augmenter les graves dans un discours. En outre, le framework audio peut désormais remonter aux applications l'horodatage des paquets audio (*Audio PTS* ou *Audio Presentation Timestamps*) en provenance de la HAL. Ceci permet aux applications une meilleure synchronisation audio/vidéo en accordant (enfin) les PTS des trames audio et des trames vidéo.

5 Capteurs et traitements par lots

Toujours dans une volonté d'économiser la batterie de votre terminal, Android 4.4 offre la possibilité de délivrer les évènements des différents capteurs (accéléromètre, compas...) par lots différés, plutôt qu'en instantané. Ceci permet ainsi au SoC de rester en veille plutôt que d'être continuellement réveillé à chaque interruption levée par un capteur matériel. Ce mécanisme s'avère plutôt judicieux pour les sportifs qui utilisent leur terminal pour enregistrer des informations (e.g. jogging), sans pour autant devoir maintenir leur téléphone (et son écran) allumé. D'un point de vue développeur, trois possibilités vous sont offertes pour récupérer les événements d'un capteur :

- Requête explicite d'événements à un moment arbitraire ;
- Requête différée d'événements à la fin d'un cycle de traitement ;
- Requête différée d'événements par contrôle de la fréquence des cycles de livraison.

La HAL (cf. le fichier **`hardware/libhardware/include/hardware/sensor.h`**) se voit ainsi étendue d'une nouvelle fonction **`batch()`** à implémenter pour chacun des pilotes de capteur à supporter. Dans le cas habituel (traitement immédiat), tous les événements renvoyés par le capteur le sont dès leur détection. Ainsi, un accéléromètre configuré à 50 Hz générera des interruptions 50 fois par seconde. Dans le cas d'un traitement par lots (mode « batch »), les événements reportés sont stockés et renvoyés dès qu'un événement aura été retardé par davantage de nanosecondes que spécifiées en valeur de timeout. Le nombre d'interruptions est ainsi grandement réduit et le SoC n'en restera en veille que plus longtemps. Chaque événement remonté se voit néanmoins adjoindre un horodatage, permettant à l'application de suivre ces derniers, même lorsque traités en batch. Le comportement

du traitement par lots est cependant quelque peu différent en fonction du contexte CPU :

- En usage normal (lorsque le CPU est allumé), les événements doivent être reportés par batch à chaque fin de période (atteinte du timeout). Aucun événement ne peut être perdu ;- En veille (lorsque le CPU est éteint), les capteurs ne doivent pas pouvoir réveiller le CPU et le sortir de cet état. De ce fait, seuls les capteurs restent actifs et ces derniers enregistrent les évènements perçus au sein d'une FIFO circulaire (les capteurs disposant d'une faible capacité mémoire). Des événements sont donc susceptibles d'être perdus. Lorsque le système sortira de veille, un batch sera renvoyé au CPU avec les derniers événements de l'historique.

À noter qu'avec Android 4.4, la famille des capteurs s'agrandit encore un peu plus, avec le support des détecteurs de pas et autres compteurs de pas. Du bonheur pour les coureurs !

6 Émetteurs infrarouges

Kit Kat apporte en outre le support des émetteurs infrarouges (ou *IR Blasters*), au sein d'une nouvelle HAL que vous trouverez via le fichier `hardware/libhardware/include/hardware/consumerir.h`. Cette dernière (et l'API applicative associée) vous permettra de développer des applications permettant le contrôle de TV et autres appareils électroniques. Il vous sera ainsi très facile de scanner les fréquences infrarouges disponibles en implémentant les fonctions suivantes de la HAL :

```
int (*get_num_carrier_freqs)(struct consumerir_device *dev);
int (*get_carrier_freqs)(struct consumerir_device *dev,  size_t len, consumerir_freq_range_t *ranges);
```

Tout comme vous pourrez transmettre des données à la fréquence choisie via la fonction `transmit()` :

```
int (*transmit)(struct consumerir_device *dev, int carrier_freq, const int pattern[], int pattern_len);
```

7 Sécurité avancée

Enfin, finissons ce tour d'horizon des changements apportés à la plateforme Android par Kit Kat, par une revue des nouvelles fonctionnalités de sécurité introduites :

- **SELinux** (mode renforcement). Introduit avec Jelly Bean mais en mode permissif (seule une notification de violation de politique de sécurité était levée), Kit Kat voit désormais passer sa politique de sécurité au niveau supérieur. Toute violation d'un domaine SELinux en mode renforcé sera ainsi bloquée.
- **VPN par utilisateur**. Pour les tablettes supportant de multiples utilisateurs (cf. Jelly Bean), l'activation d'un VPN pour router le trafic Internet se fait désormais de manière cloisonnée à chaque utilisateur, plutôt que d'être étendue à tout le système.
- **Support des algorithmes de chiffrement DSA et ECDSA** (*Elliptic Curve DSA*). Le KeyStore d'Android se voit maintenant adjoindre deux nouveaux algorithmes vous permettant de chiffrer vos données.
- **Notification de certificats SSL néfastes**. L'utilisateur est désormais notifié si un certificat SSL s'est vu ajouté au système et qui pourrait en compromettre l'intégrité en permettant le suivi de trafic du réseau chiffré.
- **Rejet de certificats SSL frauduleux**. Le système détecte désormais et empêche l'utilisation de certificats Google frauduleux lors de communications SSL/TLS.

Conclusion

Voici donc qui met fin à notre présentation des évolutions majeures de la plateforme Android introduites par la version 4.4, dite Kit Kat. Le système continue d'évoluer et remet progressivement en cause des années d'acquis (comme par exemple la future bataille Dalvik vs. ART), les contraintes matérielles évoluant avec le temps. C'est ainsi avec plaisir que nous découvrirons la prochaine version, dès sa sortie des cartons de Google (comme d'habitude, aucune date ou numéro de version n'est annoncé). En attendant, et comme on le dit si bien, « Have a break, have a ... » ;-) ■

Référence

[1] http://www.androidpolice.com/2013/11/12/meet-art-part-2-benchmarks-performance-wont-blow-away-today-will-get-better/

21 & 22 juin 2014 - LE CENTQUATRE-PARIS

Maker Faire® Paris

L'événement Maker Faire arrive en France

Astronomie
Inventions Cuisine
Impression 3D
Robotique Écologie
Chimie Sciences
Do It Yourself Drone
Compétition de robots Geek
Craft
Arduino Origami
Énergie Modèlisme
Génie Musique Internet
Objets connectés Bricolage
Fun Artisanat Ingénierie

Ateliers - Démonstrations - Conférences

Informations et billetterie
makerfaireparis.com

Un événement Make:

Co-produit par leFabShop, 104 paris l'incubateur

AUTOUR DU LANGAGE C

par ***Damien Balima***

Aperçu de quelques logiciels et applications essentiels gravitant autour du langage C.

Introduction

Depuis sa sortie dans les années 70, le langage C et la programmation en général ont bien évolué. De nombreux outils sont venus se greffer pour améliorer la qualité de production, réduire les risques de bugs, faciliter la maintenance et la gestion des projets. Parmi ces outils, on trouve en premier lieu ceux liés au compilateur **gcc**, comme le débogage avec **gdb**, la couverture de code avec **gcov** ou le profilage de code avec **gprof**. Mais l'on trouve également des utilitaires de compilation comme **make**, des analyseurs de code, de fuite de mémoire, des utilitaires de tests unitaires, de tests d'intégration, de modélisation, de vérification formelle, des générateurs de documentation, sans oublier des bibliothèques, des environnements de développement, des interfaces graphiques, des outils de gestion de versions et de management de projets.

Dans cet article, nous nous proposons de faire un rapide survol des outils GNU/Linux rattachés au langage C : une liste non-exhaustive – on ne pourra pas lister toutes les applications et bibliothèques existantes – mais assez représentative, pour mener à bien un projet de développement en C.

1 GCC on the fly

Parmi les options du compilateur GCC disponibles, version GNU libre développée par Richard Stallman du compilateur CC, on trouve notamment des options liées au débogage (**-g**), à la couverture de code (**--coverage**), au profilage de code (**-pg**), et également des options de détection d'erreurs (**man gcc**).

1.1 Détection des erreurs

Une erreur détectée par GCC interrompt la compilation d'un fichier objet ou l'édition des liens du programme. Elles peuvent être dues à des erreurs de syntaxe, de grammaire (utiliser un type *long char* par exemple), de déclarations, de constantes (réassigner une constante), de *cast* entre différents types incompatibles, des erreurs de *linkage* (un symbole externe non résolu, une librairie non renseignée ou introuvable).

En plus de détecter ces erreurs, GCC peut aussi analyser le code plus en détails et alerter sur de possibles problèmes à venir. Ces alertes, ou *warnings,* peuvent être assez importantes. Pour activer la plupart des alertes, ajoutez l'option **-Wall** à la compilation. Dans ce cas, GCC détectera les variables déclarées mais non utilisées, les erreurs d'adresses, de limites de tableau, d'énumérations, de commentaires, de déclarations implicites, de parenthèses et bien d'autres encore. Certaines alertes ne sont pas activées par l'option **-Wall**, comme celles de l'option **-Wextra** (cf. **man gcc**).

1.2 Débogage

La compilation ne garantit pas que le programme va fonctionner dans tous les cas. Aussi, une vérification consiste à passer le débogueur GDB *(The GNU debugger)* pour vérifier l'état des variables, des pointeurs et des sorties de fonctions.

Pour ce faire, on ajoute l'option **-g** à la ligne de commandes GCC, ce qui a pour effet d'ajouter des symboles de débogage au fichier binaire, ce qui augmente également sa taille. Par contre, cette option est à proscrire durant la compilation du programme de production si vous voulez protéger le code source. C'est pourquoi, en principe, on définit deux environnements de compilation : l'un pour la production, l'environnement *Release*, l'autre pour le débogage, l'environnement *Debug*.

Nous utiliserons le code d'exemple suivant (**exemple.c**) :

```
#include <stdlib.h>
#include <stdio.h>

#define TRUE 1
#define FALSE 0

int afficher(const char *message)
{
    char erreur[1] = {0};
    long c = erreur;

    erreur = printf("%s\n", message);
    if (erreur < 0) {
        return FALSE;
    }
    return TRUE;
}

int main(int argc, char *argv[])
{
    const char *p = NULL;

    if(argc > 1){
        p = argv[1];
    }else{
        p = "C is sexy";
    }
    if (!afficher(p))
    {
        perror("afficher");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

Pour ajouter les symboles de débogage :

```
$ gcc exemple.c -g -o Debug/exemple
```

On peut à présent utiliser le débogueur GDB avec ce fichier binaire. Pour vérifier l'état du programme à un endroit précis du code, on doit placer, avec **gdb**, des points d'arrêt (ou *breakpoints*) aux lignes de code souhaitées (*break Ligne* ou *break Fonction*). Une fois les points d'arrêt placés, on lance le programme sous **gdb** (commande **r**), qui se mettra en pause au premier breakpoint. À partir de là, on peut utiliser les commandes de **gdb** pour connaître la valeur d'une variable (commande **display**), de la mémoire, d'une fonction, mais l'on peut également suivre le programme instruction par instruction (commandes **s**, **n**) jusqu'au bogue, si bogue il y a, ou sauter vers le prochain breakpoint.

Exemple d'utilisation en ligne de commandes :

```
$ gdb Debug/exemple
[...]
Reading symbols from /home/dams/Debug/exemple...done.
(gdb) break main
Breakpoint 1 at 0x40058c: file exemple.c, line 20.
(gdb) r
Starting program: /home/dams/Debug/exemple
Breakpoint 1, main (argc=1, argv=0x7fffffffea48) at exemple.c:20
20      const char *p = NULL;
(gdb) s
22      if(argc > 1){
(gdb) s
25          p = "C is sexy";
(gdb) s
27      if (!afficher(p))
(gdb) s
afficher (message=0x400688 "C is sexy") at exemple.c:9
9     int erreur = 0;
(gdb) s
11      erreur = printf("%s\n", message);
(gdb) n
C is sexy
12      if (erreur < 0) {
(gdb) display erreur
1: erreur = 10
(gdb) k
(gdb) q
```

On retrouve souvent les principales commandes **gdb** intégrées aux environnements de développement, ce qui facilite le débogage, mais on trouve également des interfaces pour **gdb**, comme **Ddd** et **Cgdb** **[1]**.

1.3 Couverture de code

La couverture de code permet de lister les instructions exécutées durant le déroulement d'un programme, et *a contrario*, celles qui n'ont jamais été exécutées, i.e. le *code mort*. C'est très utile pour les tests unitaires, pour lesquels un exécutable de test est créé incluant la librairie à tester : on sait ainsi quelles parties du code furent testées, i.e. la *couverture de code*, un indicateur principal de qualité du logiciel.

gcov est une application GNU/Linux de référence pour la couverture de code. Dans le principe, pour utiliser **gcov**, il faut compiler la librairie ou le programme à tester avec GCC et l'option **--coverage**. Ensuite, durant l'exécution, deux fichiers **.gcno** et **.gcda**, contenant les informations de couverture de code et pouvant être lus par **gcov**, sont générés (cf. **man gcc**, **man gcov**).

Par exemple :

```
$ gcc --coverage exemple.c exemple
$ ./exemple
C is sexy
$ gcov exemple.c
File 'exemple.c'
Lines executed:73.33% of 15
Creating 'exemple.c.gcov'
$ less exemple.c.gcov
...
      1:   18:int main(int argc, char *argv[])
       -:   19:{
      1:   20:    const char *p = NULL;
       -:   21:
      1:   22:    if(argc > 1){
  #####:   23:        p = argv[1];
       -:   24:    }else{
      1:   25:        p = "C is sexy";
       -:   26:    }
      1:   27:    if (!afficher(p))
       -:   28:    {
  #####:   29:        perror("afficher");
  #####:   30:        return EXIT_FAILURE;
       -:   31:    }
      1:   32:    return EXIT_SUCCESS;
       -:   33:}
```

On lit ci-dessus que les lignes 23, 29 et 30 du code n'ont pas été exécutées (caractères **#**), et que les lignes 25, 27, 31 ont été exécutées une fois, pour une couverture de code de 73.33 %.

Il existe également l'application **Lcov** qui génère une interface graphique pour afficher ces informations de façon plus conviviale **[2]**.

1.4 Profilage de code

Le profilage de code renseigne sur les ressources (processeur et mémoire) utilisées par un programme, de façon à déterminer les fonctions à optimiser, entre autres. Le premier utilitaire que l'on pourrait citer est l'utilitaire **gprof**, qui nécessite également GCC. Pour utiliser **gprof**, il faut compiler le programme avec l'option **-pg** (cf. **man gcc**). Ensuite, l'exécution du programme générera un fichier **gmon.out**, qui pourra être lu et analysé par **gprof** (on indique à nouveau le programme exécutable en paramètre).

Par exemple :

```
$ gcc -pg exemple.c -o exemple
$ ./exemple
C is sexy
$ gprof --brief exemple
...
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        1     0.00     0.00  afficher

     Call graph

granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       1/1           main [7]
[1]     0.0    0.00    0.00       1         afficher [1]
-----------------------------------------------

Index by function name

   [1] afficher
```

Avec un programme aussi court, l'information n'est pas très pertinente, mais sur un programme plus complexe, on visualiserait ainsi les fonctions requérant le plus de temps processeur et donc, les moins optimisées.

D'autres logiciels de profilage de code existent, en particulier le logiciel **Oprofile**, qui enregistre l'utilisation des ressources système durant une session de profilage et **sprof**, comme **gprof**, mais pour les bibliothèques partagées **[3, 4]**.

2 Les Makefiles

Devenus incontournables pour les projets de développement, les fichiers Makefiles sont des fichiers textes utilisés par le *moteur de production* **GNU Make** pour faciliter la compilation et l'installation d'un programme. Après une première compilation, seuls les fichiers sources modifiés sont recompilés, ce qui apporte un gain de temps non négligeable.

Un fichier nommé **Makefile** doit être édité – soit directement, soit en passant par des outils comme les **autotools** (ou *GNU build system*) – de façon à renseigner les fichiers sources à compiler et leurs commandes de compilation. Chaque commande doit être précédée d'une *tabulation* (sans espaces). Les étiquettes (labels) des opérations sont suivies de deux points. Par défaut, c'est l'opération **all** qui s'exécute.

Exemple de fichier **Makefile** :

```
all: exemple4

exemple4: exemple4.o fonctions.o
<-TAB->gcc exemple4.o fonctions.o -o exemple4
```

```
exemple4.o: exemple4.c
  gcc -o exemple4.o -c exemple4.c

fonctions.o: fonctions.c
  gcc -o fonctions.o -c fonctions.c

clean:
  rm -rf *.o exemple4
```

```
$ make
gcc -o exemple4.o -c exemple4.c
gcc -o fonctions.o -c fonctions.c
gcc exemple4.o fonctions.o -o exemple4
$ make clean
rm -rf *.o exemple4
```

Les projets les plus complexes font appel à des générateurs de fichiers Makefiles, comme les *autotools* (**automake**, **autoconf**), et possèdent en principe une phase de configuration (**configure**), de compilation (**make**), d'installation (**make install**) et de désinstallation (**make uninstall**).

3 Les environnements de développement

Les environnements de développement (EDI, ou IDE, *Integrated Development Environment*), sont des interfaces avec des outils intégrés pour faciliter le développement. Dans cette catégorie, on trouve des éditeurs de textes à coloration syntaxique, mais aussi des systèmes complets de développement avec moult plugins et accessoires.

3.1 Les éditeurs de textes

Les premiers EDI furent des éditeurs de textes optimisés pour la programmation, comme **Vim** (*Vi Improved,* de Bram Moolenaar) et **Emacs** (développé à l'origine par Richard Stallman sur un PDP-10). Ces éditeurs étant légers, rapides à lancer et à utiliser, sont parfaits pour une édition de code ponctuelle.

Vim gère plusieurs modes d'édition (commande, insertion, remplacement, visualisation) et s'utilise en console. Il s'ouvre par défaut en mode commande. Pour se placer en mode d'insertion de texte, il faut appuyer une fois sur la touche **I** ; pour se remettre en mode commande, la touche **ESC** ; pour le mode remplacement de texte, la touche **R** ; et pour le mode visualisation, la touche **V**. Pour quitter, se placer en mode commande, puis saisir

les caractères **:q** (ou **:wq** pour enregistrer et quitter). Ensuite, il existe une multitude de raccourcis clavier pour le traitement de texte et autres opérations, comme par exemple **CTRL-W V** pour diviser la fenêtre verticalement, **CTRL-W F** sur un **#include** pour ouvrir le fichier inclus, ou **CTRL-W I** sur une fonction pour retrouver sa déclaration, **dd** (*delete*) pour couper une ligne, **yy** (*yank*) pour la copier, **p** (*put*) pour la coller, **u** (*undo*) pour revenir en arrière, etc.

De plus, Vim (voir figure 1) est hautement paramétrable et possède de nombreux modules, ce qui en fait un éditeur de premier choix pour ses utilisateurs expérimentés.

Emacs, quant à lui, gère plusieurs tampons de textes (*buffers*), des macros (d'où le nom Emacs, pour *Editing MACroS*), plusieurs modes d'édition, et se lance par défaut avec une interface graphique. Emacs dispose aussi de nombreux raccourcis clavier, dont les suivants : **CTRL-X CTRL-C** pour quitter, **CTRL-Z** pour suspendre la session, **CTRL-S** pour une recherche, **CTRL-/** pour un *undo*, **CTRL-X CTRL-S** pour enregistrer, **CTRL-W** pour couper, **Alt+W** pour copier, et **CTRL-Y** pour coller. Ensuite, Emacs est lui aussi hautement paramétrable et possède de nombreux modules **[6]**.

De plus, il existe aujourd'hui de nombreux éditeurs de textes à coloration syntaxique pour la programmation, comme **Gedit** (l'éditeur de GNOME), **Kate** (l'éditeur de KDE), **Nedit**, **Nano**, **Scribes** ou **Scite**.

Note

Sous Linux, et Unix en général, il est recommandé de connaître les commandes de base de l'éditeur **Vi**, qui peuvent, par exemple, être nécessaires sur un ancien serveur.

Fig. 1: Édition de code sous Vim

3.2 Les environnements graphiques

Il s'agit de logiciels dédiés au développement, contenant en général de nombreuses fenêtres, de l'auto-complétion et divers outils. Plus complets sur certains points, mais aussi plus lourds à charger, ils ont l'avantage de centraliser les ressources (fichiers sources, plugins) nécessaires au développement dans un unique projet de développement. Il en existe un bon nombre sous GNU/Linux.

L'un des plus connus est probablement **Eclipse**, projet open source qui possède de nombreux *forks* commerciaux, davantage utilisé pour le développement Java, mais qui possède cependant un plugin **CDT** pour les langages C et C++. Eclipse (voir figure 2) possède de nombreux plugins et tourne sous une JVM *(Java Virtual Machine)*, aussi requiert-il pas mal de ressources système **[7]**.

Très connu également sous GNU/Linux, **CodeBlocks** est également bien fourni, avec de nombreux plugins à portée de clic, et il gère nativement les projets en langage C. On trouve également **Anjuta**, **Geany**, **CodeLite**, **NetBeans** (concurrent d'Eclipse pour Java, gère également les projets C), **Kdevelop** (l'IDE de KDE), **Qt-Creator** (pour les projets Qt) et bien d'autres **[8, 9]**.

4 Les analyseurs de code

En plus des avertissements et analyses de GCC, il existe des logiciels dédiés à l'analyse de code, en particulier pour du code C. L'un des plus anciens est probablement le programme **Lint**, ancêtre de nombreux analyseurs *Lint-like*, développé dans

LE CLOUD GAULOIS, UNE RÉALITÉ ! VENEZ TESTER SA PUISSANCE

EXPRESS HOSTING

Cloud Public
Serveur Virtuel
Serveur Dédié
Nom de domaine
Hébergement Web

sales@ikoula.com
01 84 01 02 50
express.ikoula.com

ENTERPRISE SERVICES

Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

sales-ies@ikoula.com
01 78 76 35 50
ies.ikoula.com

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

sales@ex10.biz
01 84 01 02 53
www.ex10.biz

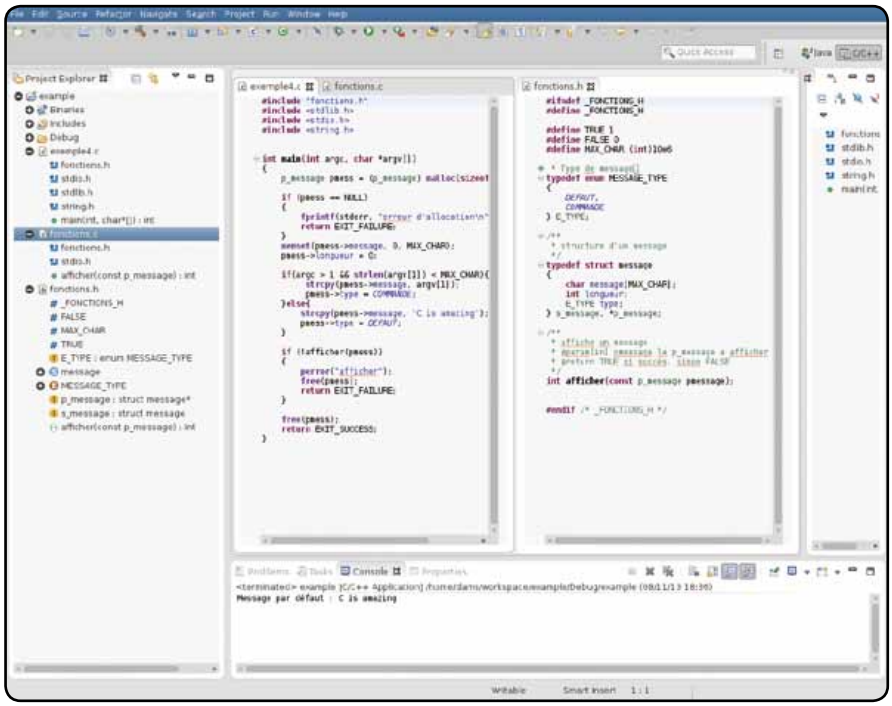


Fig. 2 : Eclipse Kepler avec le plugin CDT

les Laboratoires Bells par Stephen C. Johnson, et dont la plupart des détections ont été portées sous GCC **[10]**.

Il existe aujourd'hui bon nombre d'analyseurs de code statiques, certains fournissant également des indicateurs de métrologie, ou *metrics*, d'autres intégrant également de la vérification formelle, et certains sont même de véritables plateformes de vérification.

Parmi les solutions libres ou open source les plus connues, on peut citer **CppCheck**, intégré dans de nombreux EDI en tant que plugin et qui fonctionne pour le C et le C++ ; **Splint**, une version évoluée de Lint gérant également la vérification formelle ; **Frama-C** et **SonarQube** qui sont tous deux des plateformes de vérification, ainsi que **Sparse**, de Linus Torvalds – le créateur du système Linux –, dédié à l'analyse des noyaux Linux. Mais il en existe de nombreux autres en versions commerciales, complets, certifiés ou récompensés, et indispensables pour les applications industrielles critiques comme **Veracode**, **Polyspace**, **Klockwork Insight**, **QA·C** ou **CodeSonar**. La plupart font référence à la norme MISRA C pour les règles industrielles de codage en C **[11]**.

On trouve également l'excellent et open source **CCCC** *(C and C++ Code Counter)*, davantage spécialisé pour les metrics (lignes de code, commentaires par ligne de code, complexité de code), et qui produit également une interface HTML.

Pour un premier analyseur de code, **CppCheck**, qui s'utilise d'un clic de souris avec CodeBlocks par exemple, indiquera, entre autres, si une variable peut être réduite de portée, si elle est obsolète, ou si elle risque de générer une fuite de mémoire.

5 Les détecteurs de fuite de mémoire

Lorsqu'un programme ne libère pas ses allocations mémoire (avec des appels à la fonction `free`) et qu'en plus, il requiert de plus en plus de mémoire durant son déroulement, le système risque d'être à court de mémoire disponible, générant un ralentissement, voire une panne système. Pour éviter ce phénomène de fuite de mémoire, il existe des détecteurs, qui permettent entre autres de savoir si une allocation mémoire avec un appel à la fonction `malloc` n'a pas été libérée par un `free`.

De façon générale, sous GNU/Linux, on peut vérifier l'état de la mémoire du système via de nombreuses commandes et applications permettant d'afficher la mémoire consommée par les processus, allant de la commande `ps` à la commande `top`, en passant par `vmstat`, ou la lecture d'un fichier `statm` d'un dossier de processus du répertoire virtuel `/proc/`. Il existe également des moniteurs système comme `gnome-system-monitor`, `ksysguard`, `conky` ou `gkrellm`, des serveurs de monitoring comme **Nagios**, **Munin** ou **mon**, et une multitude d'applets de bureau comme **adesklets** ou **gdesklets**.

Cependant, pour obtenir précisément la liste des instructions d'un programme à l'origine d'une fuite de mémoire, il faut passer par des débogueurs de mémoire (*memory debuggers*) comme **Valgrind** ou des outils pour Gcc comme **Mudflap**, **Mcheck**, **Mprobe** et **Mtrace**. On trouve aussi dans le domaine commercial des outils comme **Purify**, **BoundsChecker** et **Insure++ [12, 13, 14]**.

Valgrind possède plusieurs outils intégrés comme **Memcheck**, qui est utilisé par défaut, **Cachegrind** pour la mémoire cache, **Callgrind** pour les graphes d'appels et **Hellgrind** pour les threads. L'outil **Memcheck** renseigne notamment sur la mémoire *heap* allouée, les fuites de mémoire possibles et les variables non initialisées. Par exemple :

```
$ valgrind ./exemple4
==7180== Memcheck, a memory error detector
==7180== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward
et al.
==7180== Using Valgrind-3.7.0 and LibVEX; rerun with -h for
copyright info
==7180== Command: ./exemple4
...
Message par défaut : C is amazing
==7180== HEAP SUMMARY:
==7180==     in use at exit: 0 bytes in 0 blocks
==7180==   total heap usage: 1 allocs, 1 frees, 10,000,008 bytes
allocated
==7180==
==7180== All heap blocks were freed -- no leaks are possible
...
```

Entre caractères '**==**' on trouve l'identifiant du processus, tandis que le résumé nous informe qu'il y a bien eu une allocation et une libération de mémoire d'un espace de 10 Mo, et qu'aucune fuite n'a été détectée (**no leaks are possible**).

6 Les tests unitaires

Les tests unitaires vont permettre de tester les fonctionnalités du code, dans les cas prévus, les cas d'erreurs et les cas limites, pour s'assurer que le programme fonctionne bien selon ses spécifications.

En principe, il faudra fournir une bibliothèque du programme dont les fonctions exportées pourront être testées par un programme de test linké avec celle-ci et une bibliothèque de tests unitaires. Une fois les tests unitaires réalisés, en exécutant ce programme de tests, ils pourront être à nouveau utilisés pour s'assurer de la compatibilité des nouvelles versions (tests de non-régression). Au niveau des EDI, on ajoute généralement un projet spécifique pour les tests unitaires – que l'on trouve parfois dans la liste des types de projets disponibles – qui sera dépendant du projet de développement principal et linké avec une bibliothèque de tests unitaires (comme **CUnit**).

Il existe de nombreux frameworks de tests unitaires pour le C, comme **Check** et **CUnit**, voire des frameworks pour C++ réutilisables sous un projet de tests C++, en ajoutant des balises externes C autour des **#include** de fichiers d'en-têtes de la librairie C à tester **[15]**.

Au niveau du programme de tests, celui-ci repose sur une suite d'assertions qui doivent être vérifiées pour valider les tests. Par exemple, avec Check, pour tester une fonction **afficher** :

```
...
START_TEST (afficher)
{
   fail_if (afficher(NULL) == TRUE, "doit retourner FALSE si
argument NULL");
   fail_if (afficher("test") == FALSE, "doit retourner TRUE si
argument valide");
}
END_TEST
...
```

Les tests unitaires sont souvent utilisés en milieu industriel et sont intégrés dans des normes de qualité de logiciel avec, en plus, une certaine couverture de code à respecter selon la criticité du projet (cf. *IEEE Standard for Software Unit Testing (IEEE 1008-1987*), la norme aéronautique ED-12B, la norme industrielle IEC 61508 ou la norme spatiale européenne ECSS-E-40). Ils font également l'objet d'une certification internationale pour les testeurs professionnels : l'*International Software Testing Qualifications Board (ISTQB)*.

7 Les gestionnaires de versions

Pour faciliter le suivi du développement du projet (modifications, ajouts de fichiers, versions), il existe des gestionnaires de versions (*Version Control System* ou *Revision Control Software*). Ces derniers sont devenus indispensables pour tout projet de développement.

Au niveau open source, l'un des premiers et des plus connus à être utilisé fut le système **CVS** *(Concurrent Versions System)*. CVS peut en effet s'utiliser en local ou en client-serveur, un serveur stockant un dépôt de fichiers sources, tandis qu'un client peut à distance rapatrier une version sur son poste (commande **checkout**), mettre à jour cette version locale (commande **commit**), puis mettre à jour la version du serveur (commande **merge**).

On trouve également le gestionnaire **Subversion** (SVN) du groupe Apache, qui permet de gérer des branches spécifiques de dépôts.

Mais le gestionnaire de versions du moment est certainement **Git**, conçu par Linus Torvalds. Git apporte davantage de souplesse, de robustesse et de contrôle, et on le retrouve dans de nombreux projets open source, comme ceux des sites **GitHub** et **Gitorious**. Au niveau de ses interfaces graphiques, on trouve des interfaces comme **Gitk** (et son module **Git-gui**, inclus avec Git). **Git-cola**, **Giggle**, **Qgit**, **Gitg**, et bien d'autres. Il est également bien utile de disposer d'une interface web pour suivre les dépôts du serveur. Pour cela, on trouve des interfaces web comme **Gitweb** (distribué avec Git), **GitStat**, **Viewgit**, et des interfaces plus commerciales comme **Gitorious**, **GitHub** et **Stash** d'Atlassian **[16, 17]**.

8 Sans oublier...

8.1 Les générateurs de documentation d'API

Des générateurs de documentation comme **Doxygen** ou **Docurium** généreront de la documentation d'API à partir des commentaires des fichiers sources compris entre des balises spécifiques. La figure 3 montre un exemple d'une telle documentation générée avec Doxygen. Les documents peuvent également être centralisés et partagés grâce à des wikis comme **DokuWiki**, **MediaWiki** ou **WikiWikiWeb**, mais aussi des solutions plus commerciales comme **BrainKeeper**, **Confluence** ou **SocialText [18, 19, 20]**.

8.2 Les gestionnaires de projets

Pour les projets nécessitant d'importantes ressources humaines, pour faciliter la gestion des tâches et le planning de chacun, un gestionnaire de projets peut s'avérer indispensable. Parmi ceux-ci, on trouve des gestionnaires spécialisés dans le suivi des tâches, souvent utilisés pour la gestion des incidents, bugs et améliorations, comme **Bugzilla**, **Redmine** ou **Mantis**, mais aussi des outils commerciaux comme **Zendesk** ou **CodeBeamer**. On trouve également des gestionnaires tout-en-un, avec gestion de ressources humaines, planning et gestion des incidents intégrés, comme **Calligra Plan** de KDE, **2-plan**, **FusionForge**, **OpenProject** ou des outils commerciaux comme **MetaTeam** d'Atlova, **10,000ft**, **Jira** d'Atlassian **[21, 22]**.

8.3 Les outils de modélisation UML

Au niveau de la conception, il existe des langages normalisés, dont le plus connu est UML (*Unified Modeling Language*). Bien que les diagrammes UML soient davantage utilisés pour les langages orientés objets (diagrammes de classes), une bonne partie des diagrammes sont réutilisables pour un projet C, comme les diagrammes d'activité, les diagrammes de séquences, les diagrammes de cas d'utilisation. Ce sont des outils libres ou open source comme **Dia**, **Umbrello**, **Modelio** (voir figure 4), **Eclipse UML2 Tools** ou **ArgoUML**, ou des outils commerciaux comme **Enterprise Architect**, **Poseidon** ou **Rhapsody** d'IBM. L'utilisation de schémas normalisés facilite la reprise du projet et rend la documentation compréhensible par d'autres équipes.

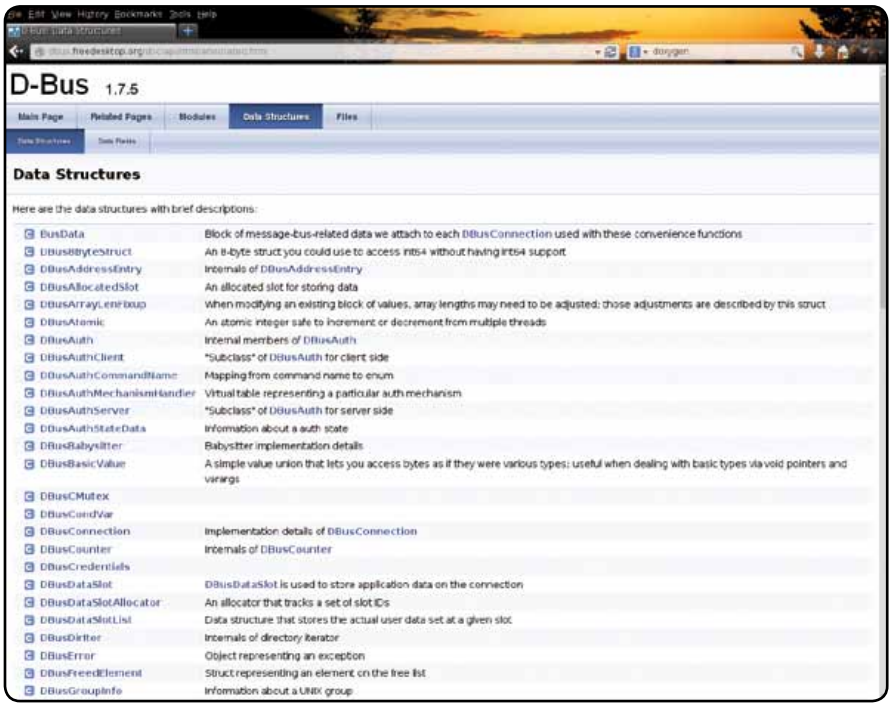


Fig. 3 : Exemple de documentation avec Doxygen, pour l'API de D-Bus

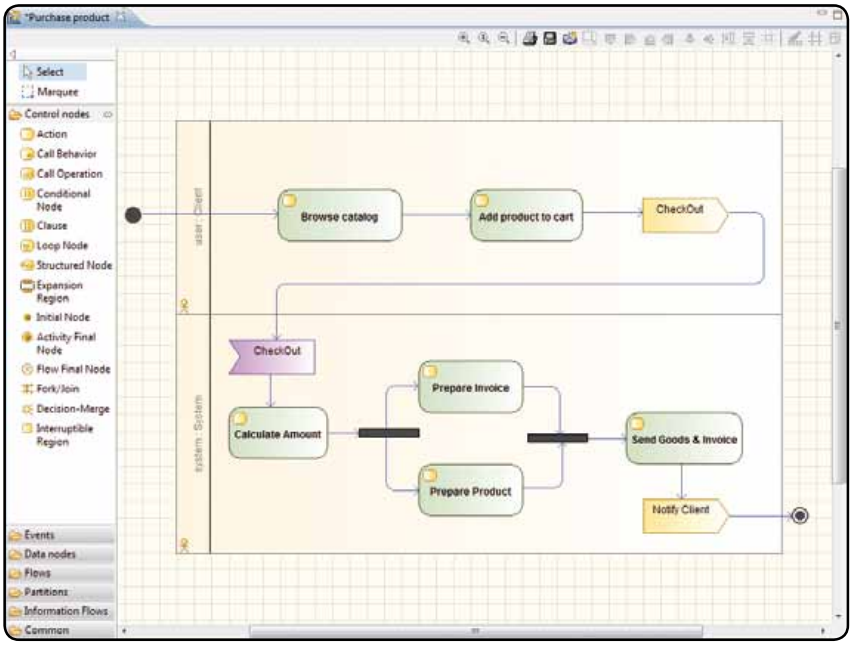


Fig. 4 : Diagramme UML d'activité, avec Modelio

8.4 Les logiciels de tests d'intégration

Les tests d'intégration ne doivent pas être oubliés : ils permettent de s'assurer de la qualité d'un livrable avec des logiciels comme **Jenkins** (open source), **Apache Continuum** (voir figure 5) ou **TeamCity** (commercial). En principe, un test d'intégration va récupérer les sources du projet depuis un dépôt (Git, SVN), lancer la commande de compilation, exécuter les tests unitaires et exécuter la commande de génération du package final. Si tout se passe bien, le test d'intégration est validé, mais à la moindre erreur (une erreur de compilation ou un test unitaire qui ne passe pas) c'est l'ensemble du test qui échoue. L'avantage, c'est qu'il s'agit d'un processus automatisé, et que l'on dispose d'une traçabilité et de la version la plus récente du projet.

8.5 Les validations formelles et model-checkers

Pour les projets les plus complexes, des *model-checkers* comme **Spin** ou **Smv** permettront de valider un modèle du programme. Pour les projets les plus critiques, il existe également des outils de *validation formelle* permettant de valider formellement, avec des annotations spécifiques, le projet, prouvant ainsi sa fiabilité et sa sûreté. On trouve par exemple les modules **Jessie** et **Wp** pour **Frama-C**, les logiciels **Splint**, **BLAST** ou **l'Atelier-B** de ClearSy **[23]**.

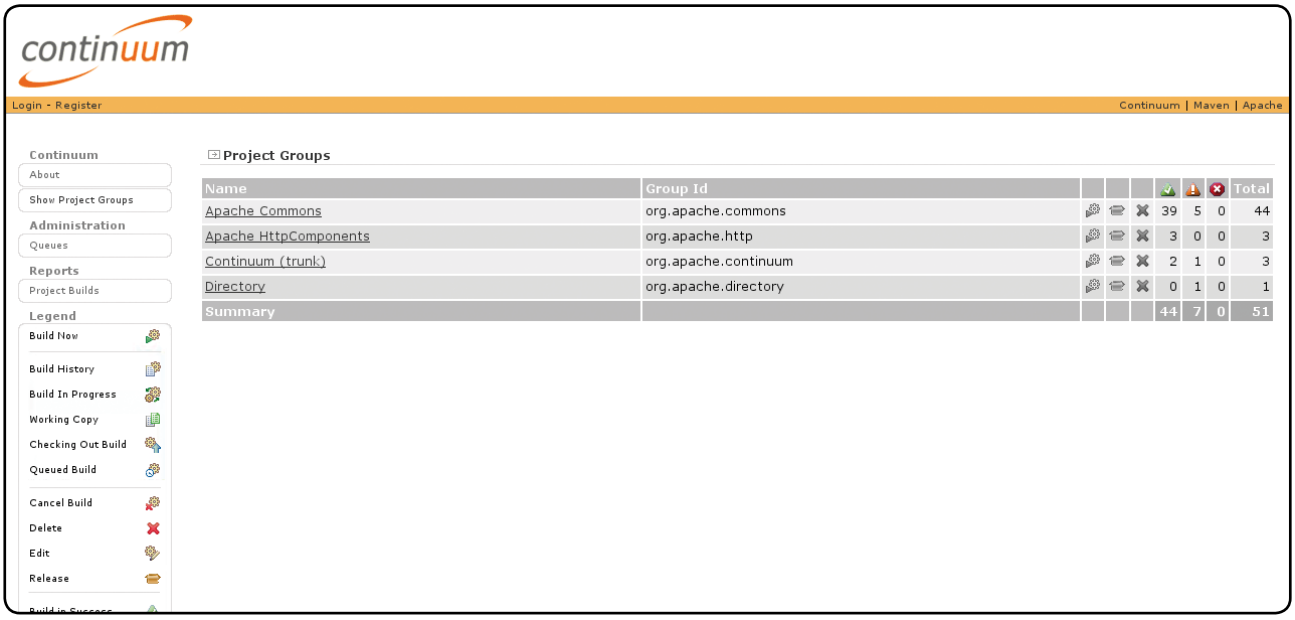


Fig. 5 : Apache Continuum

8.6 Les gestionnaires d'exigences

Un projet bien étudié comporte au préalable une phase d'étude et de rédaction d'un cahier des charges, dans lequel le client et la maîtrise d'œuvre doivent définir certaines exigences pour le produit final (fonctionnalités, normes, sécurité, internalisation, performance, ergonomie, etc.). Le nombre des exigences peut être assez conséquent et pour faciliter leur suivi, leur maintenance et leur gestion, il existe des gestionnaires d'exigences, souvent intégrés dans une solution globale de conception, comme **PTC Integrity** de MKS, **Reqtify** de Dassault Systemes, **Visure Requirement** de Visure Solutions ou **Rational DOORS** d'IBM.

Conclusion

Il existe de nombreux outils gravitant autour d'un projet de développement en C, dont certains sont essentiels, voire indispensables, comme le débogueur Gdb, le moteur de production Make, le débogueur de mémoire Valgrind et le gestionnaire de versions Git.

Cependant, on trouve aujourd'hui des outils pour satisfaire la plupart des besoins les plus exigeants, de la conception au développement, en passant par la gestion de projets. La liste ci-présentée n'est pas exhaustive, mais elle offre un aperçu de l'état actuel du marché. Il en existe encore bien d'autres, autant dans le domaine libre et open source que commercial. ■

Références

[1] Bodor D., « Petit tutoriel du débogueur GDB », GNU Linux Magazine / France, Hors-Série n°55, 2011

[2] The Linux Test Project, LCOV, http://ltp.sourceforge.net/coverage/lcov.php, 2013

[3] Levon J., Elie P., OPROFILE, http://oprofile.sourceforge.net/news/, 2013

[4] Wikipedia, List of performance analysis tools, http://en.wikipedia.org/wiki/List_of_performance_analysis_tools, 2013

[5] GNU.org , The GNU Make Manual, http://www.gnu.org/software/make/manual/make.html, 2013

[6] Wikipedia, Emacs, http://fr.wikipedia.org/wiki/Emacs, 2013

[7] The Eclipse Foundation, CDT Project, http://www.eclipse.org/cdt/, 2013

[8] Oracle Corporation, NetBeans IDE Features (C and C++ Development), https://netbeans.org/features/cpp/, 2013

[9] Digia Plc, Qt Project, http://qt-project.org/, 2013

[10] Wikipedia, Lint (software), http://en.wikipedia.org/wiki/Lint_(software), 2013

[11] Wikipedia, List of tools for static code analysis, http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis, 2013

[12] GNU.org, Heap Consistency Checking, http://www.gnu.org/software/libc/manual/html_node/Heap-Consistency-Checking.html, 2013

[13] Wikipedia, Memory debugger, http://en.wikipedia.org/wiki/Memory_debugger, 2013

[14] The Valgrind Developers, Memcheck : a memory error detector, http://valgrind.org/docs/manual/mc-manual.html, 2013

[15] Wikipedia, List of unit testing frameworks, http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks, 2013

[16] Wikipedia, List of revision control software, http://en.wikipedia.org/wiki/List_of_revision_control_software, 2013

[17] Git, Interfaces, frontends, and tools, https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools, 2013

[18] Docurium, https://github.com/libgit2/docurium, 2013

[19] Wikipedia, Comparison of documentation generators, http://en.wikipedia.org/wiki/Comparison_of_documentation_generators, 2013

[20] Bugzilla, http://www.bugzilla.org, 2013

[21] Wikipedia, Comparison of issue-tracking systems, http://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems, 2013

[22] Wikipedia,Comparison of project management software, http://en.wikipedia.org/wiki/Comparison_of_project-management_software, 2013

[23] The Blast Team, BLAST, http://mtc.epfl.ch/blast, 2013

Quelle interopérabilité entre mes différents fournisseurs Cloud ?

Avec Aruba Cloud,

vous avez l'assurance de ne pas être prisonnier d'un fournisseur. Nos services sont intégrés au **driver DeltaCloud** et compatibles **S3**. De plus, vous pouvez utiliser des formats standards d'images de machines virtuelles, **avec VHD et VMDK**, ainsi que des modèles personnalisés provenant éventuellement d'autres sources.

3 hyperviseurs

6 datacenters en Europe

APIs et connecteurs

70+ templates

Contrôle des coûts

Nous avons choisi Aruba Cloud car nous bénéficions d'un haut niveau de performance, à des coûts contrôlés et surtout car ils sont à dimension humaine, comme nous. Xavier Dufour - Directeur R&D - ITMP

Contactez-nous! 0810 710 300 www.arubacloud.fr

aruba CLOUD

MY COUNTRY. MY CLOUD.*

Cloud Public | Cloud Privé | Cloud Hybride | Cloud Storage | Infogérance

*MON PAYS. MON CLOUD.

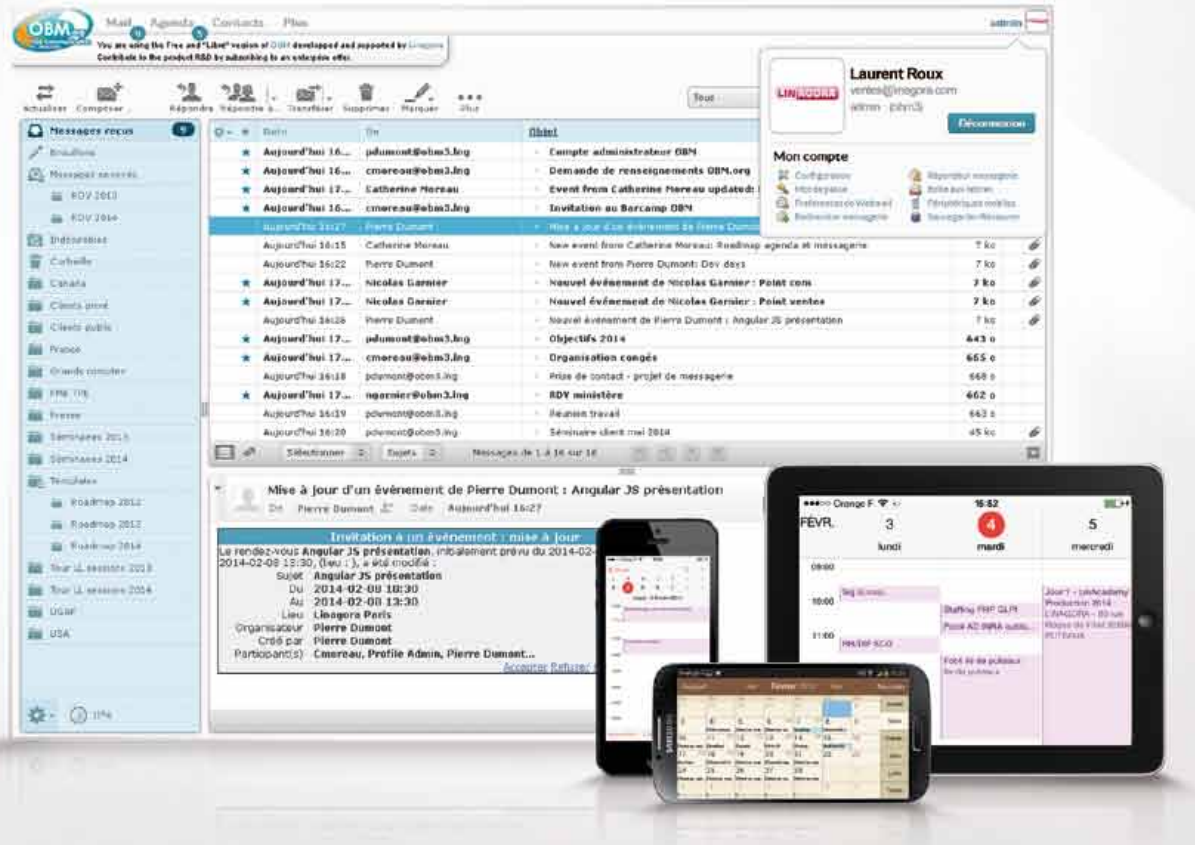
JEU CONCOURS

Gagnez un séjour à Montpellier pour les RMLL 2014

Tirage au sort lors du salon Solutions Linux le 21 mai sur notre stand D32

Info : www.linagora.com

EXIGEZ LA VRAIE SOLUTION LIBRE DE MESSAGERIE COLLABORATIVE : OBM.ORG !



ET MÉFIEZ-VOUS DES CONTREFAÇONS*

OBM.org est toujours la solution de messagerie des grandes administrations françaises : Ministère de l'Économie et des Finances, Ministère de l'Intérieur, Gendarmerie Nationale, Ministère de la Culture et de la Communication et Ministère de l'Agriculture, de l'Agroalimentaire et de la Forêt.

C'est la solution utilisée par la moitié des administrations françaises !

*OBM.org est sous licence GNU Affero GPL v3. Certaines autres solutions, soit disant "libres", utilisent le code d'OBM, mais sans respecter la licence. Dans ce cas, ce sont des contrefaçons. Ne devenez pas receleur, exigez l'original !

LINAGORA

www.linagora.com www.obm.org