N°197 octobre 2016

GNU LINUX MAGAZINE / FRANCE

DÉVELOPPEMENT ET ADMINISTRATION SUR SYSTÈMES OPEN SOURCE

JAVA / JEE

Générez vos applications avec JBoss Forge 3.3.1

p.06

CAMERA / 3D

RÉALITÉ AUGMENTÉE

INTÉGREZ VOS OBJETS 3D DANS LE MONDE RÉEL AVEC ARTOOLKIT

- Dessinez vos marqueurs
- Créez des objets 3D VRML et affichez-les

p.48

VISION ASSISTÉE / DÉTECTION

Recherchez des structures géométriques dans des images p.22

IOCCC / OBFUSCATION

Analysez un code C inintelligible p.40

ACL / RÉSEAU

Gérez vos ACLs de manière plus logique avec Capirca p.34

RÉELS / C

Mesurez et améliorez la précision de vos calculs p.14

FRAMEWORK / GARAGE

Accédez aux SGBDR en PHARO p.74

ET AUSSI : C/C++ ou Emscripten pour créer une bibliothèque Node.js ?

L 19275 - 197 - F: 7,90 € - RD

France MÉTRO. : 7,90 € – CH : 13 CHF – BEL/LUX/PORT.CONT : 8,90 € – DOM/TOM : 8,50 € – CAN : 14 $ CAD

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 66
express.ikoula.com

ENTERPRISE SERVICES

Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

sales-ies@ikoula.com
01 78 76 35 58
ies.ikoula.com

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

sales@ex10.biz
01 84 01 02 53
www.ex10.biz

GNU/Linux Magazine France
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND

10, Place de la Cathédrale - 68000 Colmar - France
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 – www.ed-diamond.com

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Tristan Colombo
Responsable service Infographie : Kathrin Scali
Réalisation graphique : Thomas Pichon
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 www.fsc.org

Membre April www.april.org

SUIVEZ-NOUS SUR :

https://www.facebook.com/editionsdiamond

@gnulinuxmag

LES ABONNEMENTS ET LES ANCIENS NUMÉROS SONT DISPONIBLES !

En version papier et PDF :
www.ed-diamond.com

Codes sources sur https://github.com/glmf

ÉDITORIAL

Je regardais dernièrement les smartphones sortis en 2016 et ceux prévus pour 2017. Je ne cherche pas à changer mon bon vieux smartphone qui n'a que trois ans demi, mais tous mes smartphones précédents n'ayant jamais eu l'occasion de fêter leur deux ans d'existence, je me méfie et commence à me renseigner... Quelle ne fut pas ma surprise de constater la grande « nouveauté » initiée cette année, la grande évolution technologique : sur pratiquement tous les modèles, la batterie est désormais non amovible (oui, même les modèles plaqués or avec touches en rubis à €€€€). Or, cet élément s'use très vite et le changer permet de prolonger la durée de vie de son smartphone : plus de batterie amovible implique une durée de vie plus courte des smartphones... mais un gain plus important pour les constructeurs (coûts de fabrication moins élevés et vente de plus de smartphones).

À l'heure d'une prise de conscience environnementale mondiale soulignée par la COP22, de l'accroissement du nombre d'objets recyclés (disparition des sacs en plastique chez les commerçants, etc.), et du tri sélectif, il est navrant de constater que, parallèlement, les objets que nous construisons peuvent de moins en moins souvent être réparés. Je parlais des smartphones, mais j'ai également pu voir un picoprojecteur dont la coque en plastique était scellée, un réfrigérateur que l'on propose de remplacer pour... une charnière cassée, une décapeuse thermique échangée sans l'avoir même inspectée, etc.

Le projet Google ARA a (avait ?) pour but de construire un smartphone modulaire permettant de ne changer que des éléments particuliers et non le smartphone dans son intégralité et de permettre ainsi de réduire l'impact économique et environnemental de la « course au meilleur smartphone ». Le premier de ces smartphones devait être commercialisé cette année, mais pour l'instant le site de Google ARA (http://www.projectara.com/) n'annonce qu'une version pour développeurs, mais rien de précis pour une version grand public.

Ainsi va donc le monde de la technologie et en attendant un smartphone complètement recyclé/able utilisons de façon ludique celui que nous avons entre les mains. Ainsi ce mois-ci, pourquoi ne pas tester la réalité augmentée en intégrant vos propres objets 3D dans la réalité ? C'est beaucoup plus amusant que de jouer à un certain jeu dont je tairai le nom, mais que tout un chacun aura reconnu et dont le principal intérêt est d'augmenter les bénéfices de la société éditrice (ainsi que la quantité de données collectées...?) et de faire courir une foule importante dans des lieux plus ou moins improbables.

Je vous souhaite une bonne lecture ainsi que de belles expériences de réalité augmentée...

Tristan Colombo

ACTUELLEMENT DISPONIBLE

GNU/LINUX MAGAZINE HORS-SÉRIE N°86 !

75 RECETTES POUR ACCÉLÉRER VOS DÉVELOPPEMENTS

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE N°197

GÉNÉRER SON APPLI AVEC JBOSS FORGE

Romain PELISSE [Sustain Developer @Red Hat]

La communauté de projet open source JBoss [1] s'est construite autour de son serveur d'application, Wildfly [2], mais a aussi été le berceau de librairies et de frameworks très utiles aux développements logiciels, comme par exemple Hibernate [3]. Pour le besoin de tous ces projets, mais aussi pour assurer un développement rapide, efficace et sûr, la communauté est aussi à l'origine de nombreux frameworks et outils de développement, malheureusement moins connus. Découvrons aujourd'hui le mystérieux et méconnu JBoss Forge, dans sa toute dernière version (3.3.1.Final du 31 août dernier), et qui est une excellente réponse à SpringBoot et autres générateurs d'applications Java.

Mots-clés : Java/JEE, Build, Gestion de dépendances, Maven, JBoss Forge

Résumé

Cet article présente JBoss Forge, un outil de génération de projets Java/JEE permettant, rapidement et de manière fiable et propre, de disposer d'un projet squelette, incluant tous les éléments nécessaires, pour démarrer son développement.

1 JBoss Forge en quelques lignes

1.1 Qu'est-ce que JBoss Forge

En quelques mots, JBoss Forge est un générateur de projet Java/JEE similaire à **AppFuse [4]** ou encore **Spring Boot [5]**. Son objectif principal est donc de libérer l'utilisateur de la conception correcte, de plus en plus difficile avec l'accroissement de la complexité d'une application, d'un fichier de construction – par exemple, le fameux `pom.xml` de **Maven [6]**.

En effet, si la plupart des fichiers de construction de ce type peuvent se limiter à quelques lignes pour un projet simple, la réalisation d'un applicatif complexe – par exemple, un projet JEE utilisant de nombreuses API (**JMS**, **JAX-RS**, **Bean Validation**) de la spécification, mais aussi embarquant plusieurs *frameworks* open source (**Hibernate Search**, **Hadoop**, **Infinispan**), ce dernier devient non seulement lui-même complexe, mais sa conception nécessite aussi de bien comprendre le fonctionnement interne de l'outil de construction, et les bonnes pratiques associées.

Bref, ceci devient rapidement un enfer, et, trop souvent, on peut perdre des journées entières de développement à essayer de faire fonctionner une construction logicielle comme on le souhaite. C'est bien là qu'apparaît tout l'intérêt d'utiliser une solution telle que JBoss Forge. Démonstration par l'exemple.

1.2 Installation

Sans surprise, il est très simple d'installer JBoss Forge. Il suffit pour ceci de télécharger l'archive, de la décompresser et de définir, comme presque toujours avec les programmes Java, une variable **FORGE_HOME** pour indiquer où réside le répertoire créé, et de redéfinir le **PATH** pour avoir les binaires et/ou scripts d'exécution de JBoss Forge dans le chemin de recherche de ces derniers :

```
$ export FORGE_HOME=/opt/java/toolsforge-
distribution-3.3.1.Final
$ export PATH=${PATH}:${FORGE_HOME}/bin
```

Rien de très compliqué, ni de magique. Testons rapidement pour vérifier que tout ceci a bien marché :

```
$ forge
Using Forge at /opt/java/forge-
distribution-3.3.1.Final


    ____
   |  __|__  _ __ __ _  ___
   | |_ / _ \| `__/ _` |/ _ \  \\
   |  _| (_) | | | (_| |  __/  //
   |_|  \___/|_|  \__, |\___|
                  |__/

JBoss Forge, version [ 3.3.1.Final ] -
JBoss, by Red Hat, Inc. [ http://forge.
jboss.org ]

[project]$
```

Note

Lors de l'écriture de cet article, aucun paquet RPM pour Forge n'était disponible pour la distribution Fedora 24. Néanmoins, de plus en plus de logiciels open source Java, spécialement ceux issus du giron JBoss, se voient rapidement dotés de tels paquets. Il n'est donc pas impossible, lorsque vous essaierez sur votre distribution, que l'on puisse installer Forge à l'aide de **Yum** ou d'un autre outil de gestion de paquets...

2 Création d'un projet avec JBoss Forge

Passons maintenant à la création d'un petit projet à l'aide de l'outil. Comme Forge est un outil en ligne de commandes, on peut aisément lui demander de l'aide sur les commandes disponibles :

```
$
alias           addon-remove          command-list          echo
ls              run
unalias         addon-update          config-clear          edit
....
```

Une fois la commande souhaitée identifiée – dans notre cas, il s'agit de **project-new**, on peut ensuite lui demander de l'aide sur cette dernière :

```
$ project-new
--named  --topLevelPackage  --version  --finalName  --targetLocation  --type  --buildSystem
```

De là, il est trivial, en une ligne de commandes, de générer notre projet :

```
$ project-new --named Belarquian --topLevelPackage org.belaran.belarquian --version
1.0-SNAPSHOT --final Belarquian
```

Sans surprise, cette commande génère une arborescence de projet Maven « classique » :

```
$ tree -L 3 Belarquian/
Belarquian/
├── pom.xml
└── src
    ├── main
    │   ├── java          # répertoire des classes Java
    │   ├── resources     # repertoire des fichiers de ressources (ex : web.xml)
    │   └── webapp
    └── test
        ├── java          # répertoire des tests Java
        └── resources     # répertoire des ressources de test (comme nous le verrons, on
place ici le fichier arquillian.xml)

8 directories, 1 file
```

Et de manière assez pratique et élégante, Forge se place directement dans le répertoire racine du projet comme l'illustre son changement de prompt :

```
$ project-new --named Belarquian --topLevelPackage org.belaran.belarquian --version
1.0-SNAPSHOT --final Belarquian
***SUCCESS*** Project named 'Belarquian' has been created.
[Belarquian]$
```

Jusqu'ici, rien de très extraordinaire et, honnêtement, d'autres outils – comme les fameux « maven-archetype » **[7]**, auraient pu arriver au même résultat. En fait, même simplement partir d'une copie d'un projet existant aurait pu fonctionner aussi bien.

C'est là où Forge apporte une intéressante plus-value, en permettant, toujours à l'aide de son outil en ligne de commandes, de modifier le projet et de lui ajouter,

par exemple une dépendance. Comme l'objectif ici est de réaliser une application JEE, nous allons donc commencer naturellement par indiquer cette nature particulière du projet, par l'ajout de la dépendance suivante :

```
[Belarquian]$ project-add-dependencies org.jboss.spec:jboss-
javaee-6.0:3.0.3.Final:provided:pom
***SUCCESS*** Installed [1] dependency.
```

Avant d'aller plus loin, ajoutons dès maintenant la dépendance, qui sera cruciale pour la suite, vers le *framework* de test unitaire **Junit**, et quittons l'interface de Forge :

```
$ project-add-dependencies junit:junit:4.12:test
***SUCCESS*** Installed [1] dependency.
```

Jetons maintenant un œil au fichier **pom.xml** généré par l'application :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.Belarquian</groupId>
  <artifactId>Belarquian</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  <build>
    <finalName>Belarquian</finalName>
    <plugins>
      <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.6</version>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.
sourceEncoding>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.jboss.spec</groupId>
      <artifactId>jboss-javaee-6.0</artifactId>
      <type>pom</type>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.spec</groupId>
        <artifactId>jboss-javaee-6.0</artifactId>
        <version>3.0.3.Final</version>
        <type>pom</type>
        <scope>provided</scope>
      </dependency>
      <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
</project>
```

On constate, sans surprise, que non seulement la configuration générée est propre et valide, mais elle respecte aussi les bonnes pratiques en cours sur la gestion des dépendances : numéros de version gérés dans la section **dependency management**, définition de l'encodage de sortie, etc.

À partir de là, vous pouvez déjà importer le projet dans votre IDE favori (**Eclipse**, **Netbeans**, etc.) ou simplement le construire à l'aide de la commande **mvn** :

```
$ mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------
[INFO] Building Belarquian 1.0-SNAPSHOT
[INFO] ------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-
resources) @ Belarquian ---
...
[INFO] --- maven-war-plugin:2.6:war (default-war) @ Belarquian ---
Downloading: https://repo.maven.apache.org/maven2/com/
thoughtworks/xstream/xstream/1.4.4/xstream-1.4.4.pom
Downloaded: https://repo.maven.apache.org/maven2/com/thoughtworks/
xstream/xstream/1.4.4/xstream-1.4.4.pom (0 B at 0.0 KB/sec)
...
[INFO] Packaging webapp
[INFO] Assembling webapp [Belarquian] in [/home/rpelisse/
Repositories/perso/articles/jboss-forge.git/project/Belarquian/
target/Belarquian]
[INFO] Processing war project
[INFO] Copying webapp resources [/home/rpelisse/Repositories/
perso/articles/jboss-forge.git/project/Belarquian/src/main/webapp]
[INFO] Webapp assembled in [32 msecs]
[INFO] Building war: /home/rpelisse/Repositories/perso/articles/
jboss-forge.git/project/Belarquian/target/Belarquian.war
...
[INFO] ------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------
[INFO] Total time: 3.703 s
[INFO] Finished at: 2016-09-04T12:14:27+02:00
[INFO] Final Memory: 18M/236M
[INFO] ------------------------------------------------------
```

Avant d'aller plus loin, prenons soin de placer notre répertoire sous un gestionnaire (distribué) de révision. Le plus connu aujourd'hui, et aussi le plus utilisé, est sans conteste **Git** :

```
$ git init
$ git status
Sur la branche master

Validation initiale

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera
validé)

        .gitignore
        pom.xml
        pom.xml.update
        src/
$ git add pom.xml src/ .gitignore
$ git commit -m "initial import"
[master (commit racine) 5f82cbf] initial import
 3 files changed, 87 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 pom.xml
```

3 Génération de code et de configuration avec JBoss Forge

3.1 Générer les objets métiers

Si JBoss Forge permet de mettre en place la configuration du projet, il permet bien évidemment d'aller beaucoup plus loin. On peut, par exemple, générer directement des objets métiers, et leur ajouter des champs :

```
$ jpa-new-entity --named TodoItem
***SUCCESS*** Persistence (JPA) is installed.
***SUCCESS*** JPA Entity org.belaran.belarquian.model.TodoItem was
created
[TodoItem.java]$ jpa-new-field --named label
***SUCCESS*** Field label created
```

Cette fonctionnalité est relativement « gadget », mais permet, si vous avez une idée assez claire de votre modèle de données, de rapidement générer ce dernier, sans même démarrer votre IDE !

En outre, Forge vous place « dans la classe », vous permettant d'en observer le contenu :

```
[TodoItem.java]$ ls

[fields]
id::java.lang.Long      serialVersionUID::long  version::int
[methods]
equals(java.lang.Object)::boolean  getVersion()::int
setId(java.lang.Long)::void        setLabel(java.lang.String)::void
getId()::java.lang.Long            toString()::java.lang.String
getLabel()::java.lang.String       hashCode()::int
setVersion(int)::void
```

Note

L'œil avisé notera que cette configuration correspond à la base de données de test déployée par défaut dans le serveur d'application JEE open source Wildfly **[2]**. Ainsi, si vous déployez votre service sur ce dernier, tout fonctionnera « comme un charme » sans même avoir à changer la configuration générée... La vie n'est-elle pas bien faite ? :)

3.2 Ajout du code du service

Comme évoqué plus haut, nous allons implémenter un service « minimal » sous la forme d'une seule classe Java, pour le moment sans réelles dépendances vers des infrastructures telles qu'une base de données ou une pile JMS, pour gérer une liste de « ToDo ». Oui, cet exemple ne remportera pas la palme de l'originalité, mais a toujours l'avantage d'être simple et didactique, et nous permettra de tester aisément le code et la configuration générés précédemment.

Là encore Forge permet d'aller très vite, et peut générer pour vous le point d'accès ReST (« endpoint ») :

```
$ rest-generate-endpoints-from-entities --targets org.Belarquian.
model.TodoItem
***SUCCESS*** Endpoint created
```

Le squelette généré est relativement primaire, mais a le mérite d'être déjà fonctionnel :

```
package org.Belarquian.rest;

...

@ApplicationPath("/rest")
public class RestApplication extends Application {}
```

Et Forge génère aussi une classe de service déjà fonctionnelle !

```
package org.Belarquian.rest;

...

@Stateless
@Path("/todoitems")
public class TodoItemEndpoint {
  @PersistenceContext(unitName = "Belarquian-persistence-unit")
  private EntityManager em;

  @DELETE
  @Path("/{id:[0-9][0-9]*}")
  public Response deleteById(@PathParam("id") Long id) {
    TodoItem entity = em.find(TodoItem.class, id);
    if (entity == null) {
      return Response.status(Status.NOT_FOUND).build();
    }
    em.remove(entity);
    return Response.noContent().build();
  }

  @GET
  @Path("/{id:[0-9][0-9]*}")
  @Produces("application/json")
  public Response findById(@PathParam("id") Long id) {
    TypedQuery<TodoItem> findByIdQuery = em
        .createQuery(
            "SELECT DISTINCT t FROM TodoItem t WHERE t.id = :entityId
ORDER BY t.id",
            TodoItem.class);
...
  }

  @PUT
  @Path("/{id:[0-9][0-9]*}")
  @Consumes("application/json")
  public Response update(@PathParam("id") Long id, TodoItem entity) {
...
  }
    }
```

Une simple exécution de maven permet donc de construire une archive « web », complètement fonctionnelle et configurée pour déployer notre API ReST !

```
$ mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------
[INFO] Building Belarquian 1.0-SNAPSHOT
[INFO] ------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-
resources) @ Belarquian ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
...
[INFO] Packaging webapp
[INFO] Assembling webapp [Belarquian] in [/home/rpelisse/Repositories/
perso/articles/jboss-forge.git/project/Belarquian/target/Belarquian]
[INFO] Processing war project
[INFO] Copying webapp resources [/home/rpelisse/Repositories/
perso/articles/jboss-forge.git/project/Belarquian/src/main/webapp]
[INFO] Webapp assembled in [37 msecs]
[INFO] Building war: /home/rpelisse/Repositories/perso/articles/
jboss-forge.git/project/Belarquian/target/Belarquian.war
[INFO] ------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------
[INFO] Total time: 2.368 s
[INFO] Finished at: 2016-09-04T12:22:07+02:00
[INFO] Final Memory: 18M/182M
[INFO] ------------------------------------------------------
rpelisse@tiberius Belarquian (master)]$ ls target/Belarquian.war
target/Belarquian.war
```

N'oublions pas, bien sûr, d'ajouter ce nouveau service à l'historique des changements du projet :

```
$ git commit -m "ajout d'un service ToDo minimaliste"
```

3.3 Générer les « Data Access Objects »

Un « design pattern » **[8]** de programmation pour l'accès des données en base, assez classique et courant dans le monde de la Java/JEE, est le « Data Access Objects ». Pour faire court, chaque objet encaspulant des données - soit des **POJO** dans le jargon Java - peut être mis à jour (ou récupéré depuis la base) à travers un DAO. Ce dernier fait souvent le pont entre l'application et l'outil d'ORM (*Object Relationship Mapping*, aussi un *design pattern* **[8]**).

Si cette approche est élégante, la construction des DAO est très souvent une tâche pénible et redondante. Heureusement, là encore, Forge propose de générer ce code pour vous !

```
$ [TodoItemEndpoint.java]$ jpa-generate-daos-from-entities
--targets  org.Belarquian.model.TodoItem
***SUCCESS*** Dao created
```

Une fois de plus, le code généré est déjà fonctionnel - et un peu plus « utile » que le squelette fourni précédemment pour la conception du service « ReST » :

```
package org.Belarquian.dao;
...
@Stateless
public class TodoItemDao {
  @PersistenceContext(unitName = "Belarquian-persistence-unit")
  private EntityManager em;

  public void create(TodoItem entity) {
    em.persist(entity);
  }
```

```
  public void deleteById(Long id) {
    TodoItem entity = em.find(TodoItem.class, id);
    if (entity != null) {
      em.remove(entity);
    }
  }

  public TodoItem update(TodoItem entity) {
    return em.merge(entity);
  }

...
    }
```

Au passage, on constate dans l'implémentation fournie par Forge l'utilisation des API fournies par JPA **[9]**. Ceci sous-entend donc de disposer d'une configuration appropriée pour indiquer comment les POJOs sont associés aux tables dans la base de données.

Heureusement, là aussi Forge a pris soin de générer la configuration associée :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://
java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="Belarquian-persistence-unit" transaction-
type="JTA">
    <description>Forge Persistence Unit</description>
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
      <property name="hibernate.transaction.flush_before_completion"
value="true"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.
H2Dialect"/>
    </properties>
  </persistence-unit>
</persistence>
```

3.4 Générer les objets depuis la base de données

Forge permet d'aller encore plus loin que la simple génération, déjà souvent pénible, de l'habituelle « plomberie » nécessaire à la construction d'un service. Il peut aussi prendre en charge une autre « plomberie » encore plus pénible : celle de la base de données et de son schéma de données.

Ainsi si vous travaillez avec un schéma de données existant, que vous souhaiteriez exposer par un service « ReST », vous pourriez suivre la même approche que précédemment, à l'exception près que, en lieu et place de la génération d'un POJO, vous ajouteriez la construction de ce dernier à partir des tables existantes :

```
$ jpa-generate-entities-from-tables --driver-location /home/rpelisse/.
m2/repository/org/postgresql/postgresql/9.2-1004-jdbc3/postgresql-
9.2-1004-jdbc3.jar --hibernate-dialect org.hibernate.dialect.
PostgreSQLDialect --jdbc-url jdbc:postgresql://localhost:5432/test
--user-name forge --user-password forge --save-user-password y
```

Conclusion

Le monde du développement a connu – et fort à parier qu'il connaîtra encore, son lot de générateurs de code, de configurations, d'artefacts, et même de générateurs de générateurs ! Ces solutions sont souvent éphémères, car très fortement couplées à une technologie, et posent généralement la question difficile de la maintenance du code généré.

Comme tous ses prédécesseurs, Forge n'échappe pas à ce dilemme, mais offre une approche simple et utile, plus orientée *bootstrap* (génération d'un projet de départ) plutôt que de tenter, souvent vainement, de remplacer l'IDE – voire même d'essayer de « montrer au développeur » comment programmer !

Non, définitivement, Forge, comme son nom l'indique, est un **outil**. Et, comme vous l'avez lu dans cet article, il s'avère très pratique, simple et élégant. Bref, c'est un bon outil, et en tant que tel il a probablement sa place parmi les vôtres ! ■

Références

[1] Communauté « open source » JBoss : http://www.jboss.org/

[2] Serveur d'applications JEE « open source » Wildfly (anciennement « JBoss AS ») : http://wildfly.org//

[3] Hibernate ORM : http://hibernate.org/

[4] Site officiel de AppFuse : http://appfuse.org/display/APF/Home

[5] Site officiel de Spring Boot : http://projects.spring.io/spring-boot/

[6] Maven : http://maven.apache.org/

[7] Archetypes de Maven : http://maven.apache.org/guides/introduction/introduction-to-archetypes.html

[8] Project Lombok : http://projectlombok.org/

[9] Les « design pattern » en programmation : https://fr.wikipedia.org/wiki/Patron_de_conception/

[10] Java Persistence API : http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html/

LOGICIEL « LIBRE » OU OPEN SOURCE : FINISSONS-EN AVEC LES POLÉMIQUES !

Dr Kiss Cool [Attention au deuxième effet...]

Deux clans s'affrontent depuis des années : d'un côté les libristes et de l'autre les tenant de l'Open source. Il est temps de mettre tout ce petit monde d'accord, car il n'y a qu'une vérité et je m'en vais vous l'asséner dans les lignes suivantes !

Tout d'abord, je tiens à remercier publiquement le rédacteur en chef qui me permet à nouveau d'être publié en lieu et place d'articles insipides publiés dans cette rubrique qui devrait être la mienne. Je sais que je vous ai manqué : me revoilà ! Et pour ce retour je veux frapper un grand coup. Oui, pour ce retour je veux enfin en finir avec les idées reçues sur le logiciel « libre » et le logiciel Open Source (vous aurez noté les guillemets autour de libre et les majuscules d'Open Source...). Pour que mon exposé puisse être compris de tous, je vais simplement définir ces deux types de logiciels et la conclusion découlera de ces simples définitions dénuées, je le précise pour les lecteurs qui ne me connaîtraient pas encore, de tout parti pris.

1 Le logiciel « libre »

Rien qu'à lire le nom on comprend tout de suite où les partisans de ce genre de développements veulent en venir : « libre », « Woodstock », « cooOool man », etc. Le logiciel « libre » ce sont donc des gens « cooOool » qui, dans un environnement serein inventent les logiciels de demain. Traduction : ce sont des barbus psychopathes qui développent dans leur coin des logiciels inutiles avec l'obsession du code propre, documenté et maintenable. La preuve : leur gourou, un certain Richard S. est tout à fait représentatif de ce courant de pensée avec ses licences « libres » et cette aversion du logiciel Open Source.

Vous connaissez beaucoup d'entreprises, vous, qui ont du temps à perdre à rédiger de la documentation, à faire des tests ou « optimiser » le code parce que « ouais, j'ai gagné 1ms et 15 lignes de code » ? Eh oui, nous sommes bien en 2016 et ces gens-là portent également le titre d'informaticien.

Quand ils se regroupent, les libristes ont la fâcheuse tendance de se prendre pour des égyptiens. C'est tout juste s'ils n'érigent pas quelques sphynx ou obélisques pour le plaisir. Ils se contentent généralement d'opter pour une organisation pyramidale pour faire allusion à leur amour de l'Égypte. Tout le monde participe : au bas de la pyramide ce sont les imberbes, les sans-grades qui signalent les erreurs. Plus haut, il y a les mal-rasés qui corrigent les erreurs. Au-dessus, on trouve les petits barbus, ceux qui ne parlent qu'entre eux et qui s'occupent du cœur du projet. Enfin tout en haut de la pyramide se trouve le pharaon, le Grand Barbu, celui qui dirige en despote et d'une main de fer tout ce petit monde lorsqu'ils ne sont pas capables de s'entendre entre eux.

Justement, en cas de mésentente, les barbus se séparent et essaient de créer de nouvelles colonies plus loin (sans doute des restes des heures passées à jouer à **Civilization**). Généralement, après quelque temps, le processus de la sélection évolutive fait son œuvre et il ne reste plus qu'une seule colonie, celle des barbus qui ont la plus longue... barbe.

Heureusement, pour faire contre-poids nous avons le logiciel Open Source.

2 Le logiciel Open source

Ah ! Parlons enfin de choses sérieuses ! Un code source accessible pour un client heureux ! Vous demandez un logiciel ? Il vous sera fait, mais en plus, il est... Open Source ! Vous aurez accès au code ! Vous pourrez le lire simplement : fi de ces fioritures que les libristes appellent des commentaires ou de la « documentation ». Les gens qui font de l'Open Source sont des gens sérieux !

Vous vous y connaissez un peu en programmation, mais vous ne comprenez rien au code d'un projet Open Source ? C'est normal ! C'est la preuve justement que c'est bien fait ! Vous n'êtes pas assez intelligent pour comprendre toutes les subtilités employées. Vous n'auriez jamais été capable d'écrire quelque chose d'aussi complexe. D'ailleurs certaines entreprises produisant du logiciel Open Source vous livrent dans le code source le code des exemples du *framework* qu'ils ont employé : peut-être l'ont-ils même modifié et lié au code de leur application pour un peu plus de cette belle magie qui fait rêver les développeurs des clients.

Les conventions de codage changent au fil du code... Je vais vous révéler un secret, jalousement gardé par la communauté Open Source : il n'y en a pas ! Mais non, à quoi ça sert ? Et puis c'est bien la preuve que de nombreux développeurs ont travaillé sur votre projet, pas quelques pauvres barbus en mal d'amitié qui ont réussi vaguement à communiquer pour s'imposer des contraintes de codage (ce qui permet en fait dans leurs tribus de savoir qui est le chef : « on suit mes règles donc j'ai la plus grosse... grammaire ». Dans le logiciel Open Source, les patrons encouragent l'utilisation de divers styles de programmation, permettant d'égayer le code et de ravir le client recevant son code comme un cadeau du père Noël, ses petits yeux pétillants d'admiration devant tant d'ingéniosité et certainement d'intelligence pour écrire des choses qui semblent irréelles, comme venues d'un autre monde :

- Hooo ! Que c'est beau ! Et ça qu'est-ce que c'est ? Là, cette ligne avec des signes de ponctuation ?
- (grosse voix suave) Hmmm, ceci est un opérateur ternaire monsieur, c'est ce qui se fait de mieux de nos jours !

La réponse peut varier bien entendu s'il s'agit d'une cliente (informaticienne elle aussi) :

- (toujours grosse voix suave, mais un peu mielleuse cette fois-ci) Hmmm, c'est du code informatique ma petite dame, c'est très compliqué !

Et puis il y a aussi toute cette magnifique phase de test où vous, clients, pourrez participer à la grande aventure du développement de votre projet ! Le développeur Open Source n'a pas besoin de tester son code, car son code est parfait et répond toujours au besoin, c'est donc le client qui aura le privilège de pouvoir, en avant-première, exécuter le code. Si ce dernier n'est pas satisfait, ne nous leurrons pas : c'est qu'il aura mal exprimé son besoin. Sa demande de correction d'erreur sera alors immédiatement traduite en demande d'évolution et chiffrée. Vous ne pensiez tout de même pas que des demandes d'évolutions (par exemple faire en sorte que le clic sur un bouton déclenche une action) soient prises en charge dans le contrat initial ? On n'est pas chez les barbus ici !

J'ai laissé planer le suspens jusqu'ici, mais vous l'aurez compris au vocabulaire employé : « client », « patron », etc. dénotent une relation purement commerciale. Plus de préoccupation d'égo de barbus à fournir un code propre et réutilisable, non, ici le code sera simplement parfait. Tellement parfait d'ailleurs qu'il ne faudra surtout pas tenter de changer la moindre ligne, d'ajouter la moindre amélioration au risque de tout casser. Non, le logiciel Open Source n'a pas obtenu ses majuscules pour rien : c'est une affaire de P-R-O-F-E-S-S-I-O-N-N-E-L-S !

Conclusion

Je pense que la conclusion s'impose d'elle-même : laissons le logiciel « libre » aux barbus et suivons la voie tracée par le logiciel Open Source, une voie qui nous mènera tout droit vers des logiciels où l'on pourrait envisager d'interdire l'accès au code pour la protection de celui-ci et des clients l'ayant acheté. L'Open Source est une transition douce vers ce modèle parfait que nous espérons tous parvenir à atteindre le plus rapidement possible !

Et surtout, n'oubliez pas :

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

Remerciements

Je tiens à remercier toutes les sociétés éditrices de logiciels Open Source, tous ces gens qui tentent, chacun à leur échelle de sauver ces barbus égarés, ces gens qui ne prennent pas leurs clients pour des imbéciles en leur faisant croire qu'ils sont capables de modifier un programme. Vraiment, à tous ceux-là je dis un grand merci !

PEUT-ON VRAIMENT CALCULER AVEC UN ORDINATEUR : MESURER ET AMÉLIORER LA PRÉCISION

Florent LANGROGNET [Ingénieur de Recherche CNRS - Laboratoire de Mathématiques de Besançon]

Après avoir expliqué dans la première partie comment étaient stockés les réels flottants et, dans la deuxième partie, les mécanismes des opérations sur ces réels, nous savons maintenant pourquoi les calculs peuvent mener à des résultats inexacts. Nous terminerons cette trilogie par une note optimiste : il existe des solutions pour mesurer et améliorer la précision.

Mots-clés : Calcul, flottants, précision, arithmétique par intervalles, stochastique

Résumé

Nous passerons tout d'abord en revue les solutions pour améliorer la précision de calcul : augmenter le nombre de bits servant à représenter les flottantes et/ou généraliser l'arrondi correct. Puis, nous présenterons 2 arithmétiques (par intervalles et stochastique) qui peuvent être mises en œuvre afin d'estimer la précision.

Ayant lu attentivement les deux premières parties **[1]** **[2]**, vous connaissez les raisons pour lesquelles les résultats des calculs sur ordinateur ne sont pas (en général) exacts. Nous allons maintenant agir sur les leviers pour améliorer la précision ou l'estimer.

La première piste consiste à augmenter la précision de la **représentation** des flottants (le nombre de bits permettant de les stocker). La deuxième a pour ambition de combler l'un des principaux défauts de la norme IEEE-754, à savoir, la non-obligation de l'**arrondi correct** pour toutes les opérations. Il nous restera alors une méthode radicale : **changer d'arithmétique**. Ce faisant, nous aurons à notre disposition des outils très intéressants (indispensables ?) pour mesurer la précision. Car, à défaut de l'améliorer (ou avant de chercher à l'améliorer), il paraît indispensable de pouvoir l'estimer.

1 Augmenter la précision

1.1 Augmenter le nombre de bits servant à représenter les flottants

Les flottants n'étant qu'un sous-ensemble des (vrais) réels, il est tentant de vouloir densifier leur présence. Plus on aura de flottants à notre disposition, plus la représentation d'un réel par un flottant pourra être précise. La distance entre

le réel et son représentant (le flottant) sera, en général, plus petite ; les erreurs d'arrondi seront alors réduites.

La norme IEEE-754 fixe le nombre de bits pour la représentation des réels simple et double précision (respectivement sur 32 et 64 bits), mais n'interdit pas de créer et d'utiliser d'autres réels (stockés sur un nombre plus grand de bits). Lorsque ce nombre peut être choisi, on parle alors de **précision arbitraire** (ou multi-précision). C'est l'objectif de certaines bibliothèques (par exemple **GMP** - *GNU Multi Precision*) qui mettent ainsi à disposition de tels réels (et les opérations associées) et permettent d'améliorer globalement la précision.

Basées sur les mêmes principes de l'arithmétique flottante et la norme IEEE-754, ces bibliothèques héritent aussi de ses défauts : aucun problème de l'arithmétique flottante ne disparaît avec ces bibliothèques, elles repoussent simplement les problèmes, tentent de minimiser les défauts sans les faire disparaître.

Il est également important de noter qu'il n'est pas possible (sauf cas particuliers) d'établir une fonction liant la précision et le nombre de bits servant à représenter les flottants. On sait alors *simplement* que les résultats seront plus précis, mais sans savoir si cette précision sera suffisante ou si elle peut l'être avec moins de bits représentatifs.

Enfin, ce gain de précision a un prix non négligeable sur les temps de calcul qui sont directement impactés.

1.2 Généraliser l'arrondi correct

Comme nous l'avons vu dans la deuxième partie **[2]**, la norme IEEE-754 n'impose l'arrondi correct que pour 5 opérations (**+**, **-**, *****, **/** et $\sqrt{\cdot}$). En utilisant d'autres opérations (ce qui est assez courant), il n'est pas garanti que le résultat obtenu soit le meilleur représentant du vrai résultat. Cette caractéristique est a l'origine de pertes de précision et de non-reproductibilité des résultats d'un environnement à un autre.

Pour pallier ce défaut, des bibliothèques proposent des algorithmes garantissant l'arrondi correct **pour toutes les opérations** sur les flottants simple et double précision. C'est le cas par exemple de **CRlibm** (*Correctly Rounded mathematical library*). Cette généralisation de l'arrondi correct a un coût (en terme de temps de calcul) qui est loin d'être négligeable. Par exemple, pour assurer l'arrondi correct du logarithme en double précision (c'est-à-dire garantir que le résultat de `log(x)` où `x` est un réel double précision soit l'arrondi du vrai résultat), il faut mener des calculs avec une mantisse de 118 bits. C'est à ce prix que l'on obtient la généralisation de l'arrondi correct permettant d'améliorer la précision des résultats.

Mais ce type d'approche ne règle pas tous les problèmes : les pertes de précision dues aux phénomènes d'absorption, de *cancellation* persistent.

1.3 Les deux mon général !

On vient de présenter deux solutions (indépendantes) pour améliorer la précision et certains se demandent si on ne peut pas agir simultanément sur ces deux leviers pour améliorer la précision . La réponse est positive ! La bibliothèque **MPFR** (*multiple-precision floating-point computations*) permet de disposer de flottants en multi-précision et de la garantie d'avoir des arrondis corrects pour toutes les opérations.

On dispose ainsi d'outils très intéressants pour améliorer la précision... au prix d'une augmentation du temps de calcul et de changements (non négligeables) dans nos programmes.

2 Recourir à d'autres arithmétiques pour estimer la précision

Avant de chercher à améliorer la précision, il faudrait sans doute savoir si elle doit l'être !

Nous avons vu que les calculs provoquaient des pertes de précision, mais est-ce toujours critique ? Une certaine imprécision n'est-elle pas acceptable dans certains cas ? En effet un résultat à `0,01` près peut être tout à fait acceptable dans certaines situations et pour certaines applications alors que pour d'autres cette précision n'est pas suffisante.

De même, vouloir augmenter la précision sans connaître ni celle d'origine, ni celle réellement obtenue peut ressembler à une fuite en avant dont le coût en terme de temps de calcul peut être considérable. Je vous propose donc dans la suite d'apporter des réponses à une nouvelle question : comment estimer, mesurer la précision ?

2.1 L'arithmétique par intervalles

Revenons à la question fondamentale (voir figure 1) de l'arithmétique flottante : comment choisir le meilleur représentant d'un réel ? Finalement, tout ce que nous avons proposé depuis le début de cette série d'articles n'est qu'un

ensemble de réponses, plus ou moins pertinentes, à cette question.

Plutôt que d'essayer de répondre à cette (délicate) question, nous allons repartir de ce que l'on connaît avec certitude : la vraie valeur est comprise entre deux flottants. On se propose alors de choisir l'intervalle ainsi défini comme représentant de la vraie valeur (voir figure 2).

On passe donc d'une approche où l'on cherche une valeur « proche » de la vraie valeur (une approximation) à une approche où l'on sait (avec certitude) que la valeur recherchée est minorée et majorée par 2 flottants.

C'est d'ailleurs ce que nous faisons pour donner la valeur d'un réel (par exemple $\sqrt{2}$ ou π) : nous pouvons soit donner une valeur approchée (avec une certaine précision, c'est-à-dire, un certain nombre de chiffres significatifs), soit donner un intervalle.

En réalité, cette idée n'est pas nouvelle, loin s'en faut. C'est ainsi qu'Archimède a procédé pour donner un encadrement de π. Pour estimer le périmètre d'un cercle, il encadre cette valeur par le périmètre d'un polygone régulier inscrit dans ce cercle, et par le périmètre d'un polygone régulier exinscrit (voir figure 3).

On peut alors réaliser des opérations sur ces nombres (définis par des intervalles) : c'est l'**arithmétique par intervalles**.

La formule générale d'une opération entre deux intervalles **X** et **Y** s'écrit $X\ op\ Y = ensemble\{x\ op\ y; x \in X, y \in Y\}$: c'est l'ensemble des valeurs obtenues par l'opération entre les réels de chaque intervalle.

Note

Par convention, on notera les intervalles avec des lettres majuscules **(X, Y)** et les réels avec des lettres minuscules **(x,y)**.

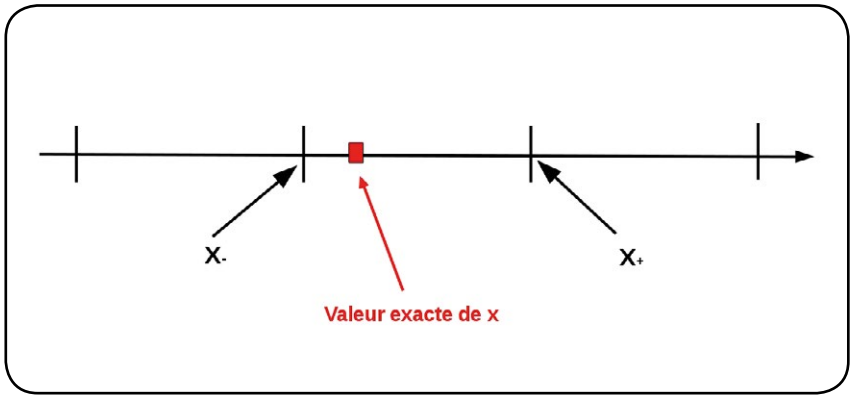


Fig. 1 : Quel est le meilleur représentant d'un réel ?

On peut appliquer cette formule aux opérations de base. Ainsi pour l'addition, on obtient $[\underline{x},\overline{x}] + [\underline{y},\overline{y}] = [\underline{x}+\underline{y}\ ,\ \overline{x}+\overline{y}]$ ($\underline{x}$ représente le flottant inférieur à **x** et $\overline{x}$ le flottant supérieur), $[\underline{x},\overline{x}] - [\underline{y},\overline{y}] = [\underline{x}-\overline{y}\ ,\ \overline{x}-\underline{y}]$ pour la soustraction et $[\underline{x},\overline{x}] \times [\underline{y},\overline{y}] = [min(\underline{x}\times\underline{y}\ ,\ \underline{x}\times\overline{y}\ ,\ \overline{x}\times\underline{y}\ ,\ \overline{x}\times\overline{y})\ ,\ max(id)]$ pour la multiplication.

Le principal atout de l'arithmétique par intervalles est qu'à chaque opération l'intervalle résultat contient le vrai résultat (inconnu). On obtient donc un encadrement de la vraie valeur. De plus, elle préserve l'associativité et la distributivité contrairement à l'arithmétique flottante (comme nous l'avons vu dans la deuxième partie).

Tout n'est pas parfait pour autant. Tout d'abord l'intervalle résultat peut être grand. On comprend bien l'intérêt d'avoir un encadrement du vrai résultat si l'intervalle est relativement petit, mais s'il est grand, l'information devient inutile. Quel est l'intérêt par exemple de pouvoir affirmer que le vrai résultat se situe dans l'intervalle $[0;+\infty[$? Et malheureusement c'est une tendance naturelle de l'arithmétique par intervalles qui peut sur-estimer l'intervalle résultat lors de chaque opération et, *in fine*, donner un intervalle résultat très (/trop) grand. Pour limiter cette tendance, on doit conduire les calculs avec tact, quitte à les réordonner (ce qui est permis puisque l'arithmétique par intervalles l'autorise), ce qui nécessite des compétences avancées.

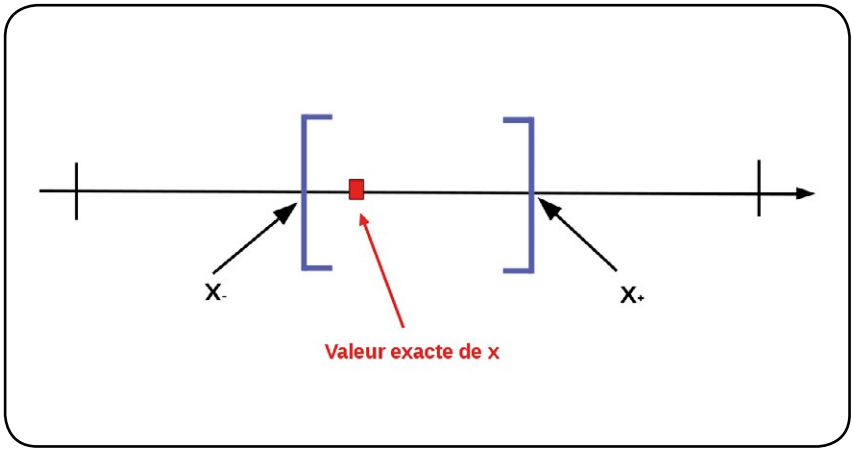


Fig. 2 : On choisit l'intervalle représentant le réel.

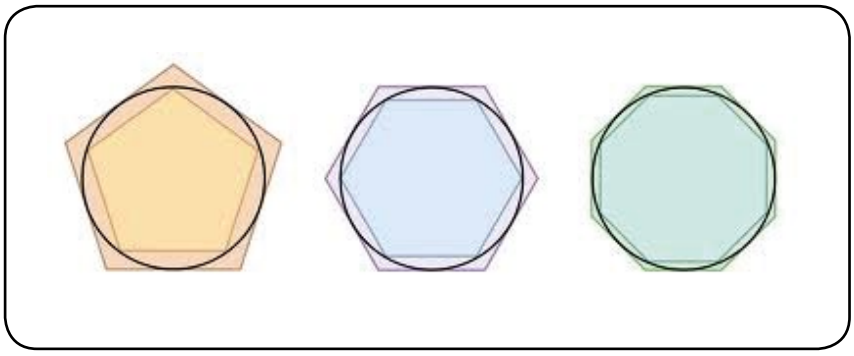


Fig. 3 : Estimation du périmètre d'un cercle par encadrement : le début de l'arithmétique par intervalle ?

De plus, la mise en œuvre de l'arithmétique par intervalles oblige à changer très souvent le mode d'arrondi (pour déterminer $\underline{x}$, on réalise une affectation avec le mode d'arrondi vers la gauche et pour $\overline{x}$ vers la droite) ce qui est très pénalisant en terme de temps d'exécution.

Je conseille à ceux qui souhaitent utiliser l'arithmétique par intervalles la bibliothèque **MPFI** (*Multiple Precision Floating-point Interval library*).

2.2 L'arithmétique stochastique

2.2.1 Impact du mode d'arrondi sur le résultat

Je reviens à la question fondamentale (voir figure 1) de l'arithmétique flottante : comment choisir le meilleur représentant d'un réel ? Et je pose la question qui arrive naturellement ensuite : quel est l'impact du choix du mode d'arrondi sur le résultat ?

Nous allons répondre à cette question au travers de deux exemples. Tout d'abord, rappelons qu'il existe quatre modes d'arrondi : **RD** (vers la gauche), **RU** (vers la droite), **RZ** (vers **0**) et **RN** (au plus près) qui est le mode d'arrondi par défaut.

Exemple 1 : Résolution de l'équation du second degré $0.3x^2 + 2.1x + 3.675 = 0$ Pour cela nous allons calculer la valeur du discriminant $\Delta = b^2 - 4ac$ avec le programme **equation.cpp** ci-dessous :

```
#include <iostream>
#include <fenv.h>
using namespace std;

int main(){
  fesetround(FE_TONEAREST);

  float a, b, c;
  cout<<"Valeur de a : "<<endl;
  cin>>a;
  cout<<"Valeur de b : "<<endl;
  cin>>b;
  cout<<"Valeur de c : "<<endl;
  cin>>c;

  float delta = b*b - 4*a*c;
  cout<<endl<<"delta : "<<delta<<endl;

  return 0;
}
```

Note

La multiplication assurant l'arrondi correct, il est conseillé de multiplier **x** par lui-même plutôt que d'utiliser la fonction **pow** (qui ne garantit pas l'arrondi correct) pour élever **x** au carré.

Ainsi, avec le mode d'arrondi par défaut (au plus près), on obtient le résultat suivant :

```
Valeur de a :
0.3
Valeur de b :
2.1
Valeur de c :
3.675

delta : -9.53674e-07
```

La valeur du discriminant étant négative, la suite conduirait à calculer deux valeurs complexes comme racine de l'équation.

Si l'on mène ces calculs en changeant le mode d'arrondi (vers la droite) selon le programme **equation_ru.cpp** dans lequel seule la ligne **fesetround(FE_UPWARD)** est différente, on obtient **0** pour valeur du discriminant et donc une racine double (**3.5**) comme résultat de cette équation.

```
#include <iostream>
#include <fenv.h>
using namespace std;

int main(){
  fesetround(FE_UPWARD);

  float a, b, c;
  cout<<"Valeur de a : "<<endl;
  cin>>a;
  cout<<"Valeur de b : "<<endl;
  cin>>b;
  cout<<"Valeur de c : "<<endl;
  cin>>c;

  float delta = b*b - 4*a*c;
  cout<<endl<<"delta : "<<delta<<endl;

  return 0;
}
```

```
Valeur de a :
0.3
Valeur de b :
2.1
Valeur de c :
3.675

delta : 0
```

Ainsi, un branchement conditionnel dépendant de la valeur d'un calcul (ici

le discriminant) peut avoir des conséquences très importantes sur la suite. Une petite imprécision sur cette valeur peut mener à des résultats bien différents.

Il est intéressant de noter que le mode d'arrondi « vers la droite » permet de trouver la solution exacte de cette équation contrairement au mode d'arrondi au plus près.

Exemple 2 : Retour à la somme de **n** fois la valeur **0.1** : $\sum_{i=1}^{n} x$

Avec le mode d'arrondi par défaut (au plus près), on obtient (pour **n =1000**) : **99.99904633**, soit une erreur de **0.001** alors qu'avec le mode d'arrondi « vers la droite », on obtient **100.0030442**, soit une erreur de **0.003** (**3** fois plus importante).

Conclusion : Ces deux exemples illustrent l'impact du mode d'arrondi sur la précision des calculs. De plus, ce n'est pas toujours avec le mode d'arrondi « au plus près » que l'on obtient les résultats les plus justes.

2.2.2 Et si l'on changeait le mode d'arrondi à chaque opération ?

À partir du constat précédent, on est en droit de se demander quel est l'impact d'un changement de mode d'arrondi à chaque opération. Nous allons reprendre le calcul de $\sum_{i=1}^{n} x$ en changeant de manière aléatoire le mode d'arrondi à chaque opération selon le programme **somme_mode_arrondi_aleatoire.cpp** ci-dessous :

```
#include <iomanip>
#include <iostream>
#include <fenv.h>
#include <stdlib.h>

using namespace std;

void changement_mode_arrondi(){
  int v = rand()%4;
  if (v == 0) fesetround(FE_DOWNWARD);
  if (v == 1) fesetround(FE_UPWARD);
  if (v == 2) fesetround(FE_TONEAREST);
  if (v == 3) fesetround(FE_TOWARDZERO);
}

int main(){
  //srand (time(NULL));
  float x=0.1;
  float res = 0.0;

  for (int nbEssai=0; nbEssai<100;
nbEssai++){
    res = 0.0;
    for (int i=0;i<1000;i++){
      changement_mode_arrondi();
        res+=x;
        x=0.1;
    }
    cout<<setprecision(10)<<res<<endl;
  }

  return 0;
}
```

L'exécution de ce programme donne les résultats suivants (on ne donne ici que les 21 premiers) :

```
./somme_arrondi_aleatoire
99.99987793
99.99987793
99.99971008
99.99988555
99.99993133
99.99978637
99.99985504
99.99980163
99.99988555
99.99971771
99.99998474
99.99986267
99.99999237
99.99980926
99.99987792
99.99989318
99.99983978
99.99963378
99.99997711
100.0000457
99.99980164
```

En poursuivant, on obtient 100 valeurs comprises entre **99.9996** et **100.00002**, soit une erreur toujours inférieure à **0.0004**.

De plus si l'on change de graine dans la suite de nombres aléatoires (en décommentant la ligne **//srand (time(NULL));**), on obtient d'autres séries de valeurs dont parfois la bonne valeur (**100**) !

Cette expérimentation conduit à deux conclusions qui vont nous mener à l'arithmétique stochastique : les résultats sont globalement meilleurs et il arrive que l'on trouve le bon résultat.

2.2.3 Concepts de l'arithmétique stochastique

L'objectif de l'arithmétique stochastique n'est pas d'améliorer la précision des résultats (même si parfois on y arrive), mais de l'**estimer**.

À chaque opération, l'arithmétique flottante choisit **un** représentant du vrai résultat. Ce choix (conditionné par le mode d'arrondi et par les implémentations des opérations mathématiques) a un impact sur la suite des calculs. En changeant le mode d'arrondi aléatoirement, on propage ainsi différemment les erreurs d'arrondis dans les calculs. Au final, on obtient **une** valeur parmi un ensemble de valeurs possibles. Sans entrer dans les détails, en modélisant ainsi le processus de calcul, les résultats obtenus représentent un échantillon d'une variable aléatoire (le résultat calculé). En ajoutant quelques hypothèses (souvent vérifiées en pratique), on montre que la distribution de cette variable aléatoire est gaussienne. Ainsi, si l'on dispose de quelques résultats (des réalisations de cette variable aléatoire), on peut estimer la vraie valeur (la moyenne de cette variable) avec une certaine précision, ce que l'on peut faire avec un test de Student. On montre alors qu'en choisissant un intervalle de confiance de **95%** et pour une taille d'échantillon de **3**, on garantit un nombre de bits exacts avec une très grande probabilité.

Ainsi en menant les calculs trois fois (et en changeant le mode d'arrondi à chaque opération), on est capable d'estimer le nombre de chiffres exacts (les chiffres qui sont les mêmes pour ces trois résultats). Par exemple si les trois résultats sont **1.234** ; **1.25** et **1.26**, on pourra alors affirmer que le vrai résultat est **1.2** avec une précision de **0.1** près.

2.2.4 Implémentation de l'arithmétique stochastique avec la bibliothèque Cadna

La bibliothèque **Cadna** implémente l'arithmétique stochastique décrite ci-dessus. Sans entrer dans les détails, on peut considérer qu'elle est composée de deux classes (**float_st** ou **double_st**) servant à représenter les réels stochastiques. Les données membres de ces classes sont trois réels (**float** ou **double**) décrivant trois valeurs possibles de ces réels stochastiques.

Toutes les opérations mathématiques ont été surchargées sur ces réels stochastiques. Ainsi une addition entre deux objets **float_st** sera traitée par la bibliothèque Cadna.

En pratique, il suffit donc de remplacer tous les réels (**float** et **double**) par des objets Cadna (**float_st** ou **double_st**) et d'ajouter quelques instructions (au minimum une ligne au début et une à la fin du programme).

Le programme **somme_cadna.cpp** permet de calculer $\sum_{i=1}^{n} x$ en arithmétique stochastique :

```
#include <stdio.h>
#include <iomanip>
#include <iostream>

using namespace std;

#include <cadna.h>

int main(){
  cadna_init(-1);

  float_st x;
  cout<<"Entrez la valeur de x"<<endl;
cin>>x;
  float_st res = 0.0;
  for (int i=0;i<1000;i++){
     res+=x;
  }
  cout<<"Somme des x (1000 fois) :
"<<res<<endl;

  cadna_end();

  return 0;
}
```

L'exécution de ce programme donne les informations suivantes.

```
./somme_cadna
-------------------------------------
CADNA_C 1.1.9 software --- University P.
et M. Curie --- LIP6
Self-validation detection: ON
Mathematical instabilities detection: ON
Branching instabilities detection: ON
Intrinsic instabilities detection: ON
Cancellation instabilities detection: ON
-------------------------------------
Entrez la valeur de x
0.1
Somme des x (1000 fois) : 0.100000E+003
-------------------------------------
CADNA_C 1.1.9 software --- University P.
et M. Curie --- LIP6
No instability detected
-------------------------------------
```

Cadna est capable de détecter des instabilités dans le code (nous y reviendrons), mais l'information essentielle ici est l'affichage du résultat avec le nombre de chiffres exacts : **0.100000E+003** (soit **100.000**). On peut donc affirmer que le résultat est **100** avec une précision de **0.001**.

Prenons un autre exemple : la suite de Muller :

$$\begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_{n+1} = 111 - \frac{1130}{u_n} + \frac{3000}{u_n . u_{n-1}} \end{cases}$$

Nous avons vu dans la première partie **[1]** que cette suite convergeait vers **100** (en arithmétique flottante) alors qu'en réalité elle converge vers **6**.

Le programme **muller_cadna.cpp** (ci-dessous) permet de calculer les valeurs cette suite en arithmétique stochastique :

```
#include <stdio.h>
#include <iomanip>
#include <iostream>

using namespace std;

#include <cadna.h>

int main(){
  cadna_init(-1);

  double_st u[30];
  u[0] = 2;
  u[1] = -4;

  for(int i=2;i<30;i++){
    u[i] = 111. - 1130./u[i-1] + 3000./
(u[i-1]*u[i-2]);
    cout<<"u["<<i<<"]"<<u[i]<<endl;
  }
  cadna_end();
  return 0;
}
```

Il donne les résultats suivants :

```
./muller_cadna
-------------------------------------
CADNA_C 1.1.9 software --- University P.
et M. Curie --- LIP6
Self-validation detection: ON
Mathematical instabilities detection: ON
Branching instabilities detection: ON
Intrinsic instabilities detection: ON
Cancellation instabilities detection: ON
-------------------------------------
u[2] 0.1850000000000000E+002
u[3] 0.937837837837837E+001
u[4] 0.78011527377520E+001
u[5] 0.715441448097E+001
u[6] 0.68067847369E+001
u[7] 0.6592632768E+001
u[8] 0.644946593E+001
u[9] 0.6348520E+001
u[10] 0.6274437E+001
u[11] 0.62186E+001
u[12] 0.6175E+001
u[13] 0.613E+001
u[14] 0.60E+001
u[15] 0.0
u[16] 0.0
u[17] 0.0
u[18] 0.0
u[19] 0.99E+002
u[20] 0.999E+002
u[21] 0.9999E+002
u[22] 0.99999E+002
u[23] 0.999999E+002
u[24] 0.99999999E+002
u[25] 0.999999999E+002
u[26] 0.9999999999E+002
u[27] 0.99999999999E+002
u[28] 0.9999999999999E+002
u[29] 0.99999999999999E+002
-------------------------------------
CADNA_C 1.1.9 software --- University P.
et M. Curie --- LIP6
CRITICAL WARNING: the self-validation
detects major problem(s).
The results are NOT guaranteed.
There are 12 numerical instabilities
9 UNSTABLE DIVISION(S)
3 UNSTABLE MULTIPLICATION(S)
```

On peut ainsi constater que :

- Cadna détecte neuf divisions instables et trois multiplications instables lors du calcul des trente premières valeurs de la suite. Cette indication devrait déjà inciter à la plus grande vigilance quant à la précision des résultats.
- La précision des premières valeurs diminue à chaque itération : la treizième valeur (`6.13)` n'a déjà plus que deux chiffres significatifs après la virgule et à partir de la quinzième, il n'y a plus aucun chiffre significatif ! Le symbole `@0` (appelé **zéro informatique**) signifie qu'il n'y a plus de chiffre significatif, le résultat obtenu est non différentiable de la valeur `0`. Les valeurs qui suivent sont donc sans intérêt puisqu'elles sont calculées à partir de valeurs de précision nulle.

Il est important de noter que le changement quasi-systématique du mode d'arrondi pénalise fortement les performances : les calculs ne sont pas trois fois plus lents (comme on pourrait s'y attendre), mais plutôt de cinq à dix fois (mais ce facteur dépend fortement des calculs effectués). De plus, la mise en œuvre est relativement intrusive, car il faut changer tous les types réels dans le programme. Enfin, si l'on a recours à d'autres bibliothèques numériques, les calculs qui y sont effectués ne sont pas menés en arithmétique stochastique (sauf à modifier et recompiler ces bibliothèques).

Malgré ces quelques limites et défauts, l'arithmétique stochastique permet de recueillir une information essentielle : une estimation de la précision d'un résultat numérique. Cette information doit ensuite être utilisée à bon escient ; elle permet par exemple de savoir (en fonction du contexte) si cette précision est suffisante.

Un résultat numérique (obtenu avec un ordinateur) peut être considéré comme une mesure en physique : il n'a de sens que si on est capable de donner l'erreur de mesure.

Conclusion

Nous arrivons au terme de cette trilogie qui avait pour objectif de donner un coup de projecteur sur ces questions essentielles et trop souvent méconnues ou ignorées. Les calculs que nous effectuons sont (plus ou moins) faux ; mieux vaut le savoir et connaître les raisons afin d'agir pour estimer la précision et, si besoin, l'améliorer. Si vous portez désormais un regard critique sur les résultats de vos calculs (sans tomber dans la paranoïa), j'aurai atteint mon objectif... ■

Pour aller plus loin

Les sources d'inspiration de cet article sont multiples et complémentaires :

- La page personnelle de l'auteur (https://lmb.univ-fcomte.fr/Florent-Langrognet) contient quelques exposés et cours sur ce sujet.
- L'article de F. Langrognet (et contributeurs), *« JDEV 2013 - Développer pour calculer : des outils pour calculer avec précision & Comment calculer avec des intervalles »* (*HPC Magazine*, novembre 2013, pp,-51-65) revient sur le traitement de cette thématique lors des JDEV (Journées du DEVeloppement logiciel) 2013, organisées par le réseau métier de l'enseignement supérieur et de la recherche DevLOG (Développement LOGiciel).
- L'école thématique CNRS *« Précision et reproductibilité en calcul numérique »* (http://calcul.math.cnrs.fr/spip.php?rubrique98) organisée par le réseau métier de l'enseignement supérieur et de la recherche Calcul en 2013.
- Le livre *« Handbook of Floating-Point Arithmetic »* de J.M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, et S. Torres (Birkhauser, 2010) est un livre de référence pour comprendre en profondeur l'arithmétique flottante.
- Les pages personnelles de Fabienne Jézéquel, Jean-Michel Muller, Nathalie Revol (acteurs de ce domaine de recherche) sont des points d'entrée très intéressants.
- Références aux bibliothèques de calcul :
  - gmplib : https://gmplib.org/
  - crlibm : http ://lipforge.ens-lyon.fr/www/crlibm/
  - mpfr : http://www.mpfr.org/
  - mpfi : http://mpfi.gforge.inria.fr/
  - Cadna : http://www-pequan.lip6.fr/cadna/

Références

[1] LANGROGNET F., *« Peut-on vraiment calculer avec un ordinateur ? »*, *GNU/Linux Magazine n°193*, mai 2016, p. 16 à 20.

[2] LANGROGNET F., *« Peut-on vraiment calculer avec un ordinateur : les opérations »*, *GNU/Linux Magazine n°194*, mai 2016, p. 22 à 31.

ACTUELLEMENT DISPONIBLE

MISC HORS-SÉRIE N°14 !

APPRENEZ À TESTER LES VULNÉRABILITÉS DE VOS SYSTÈMES ET DE VOS SERVEURS GRÂCE À METASPLOIT

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

RECHERCHE DE MOTIFS GÉOMÉTRIQUES DANS UNE IMAGE : LA TRANSFORMÉE DE HOUGH

Jean-Michel FRIEDT [Institut FEMTO-ST, dpt. temps-fréquence, Besançon]

Nous proposons d'aborder sur trois cas concrets la détection de motifs géométriques dans des images. La détection de droite est utilisée dans la navigation de véhicules, et donne l'opportunité de se familiariser avec les concepts de base de la transformée de Hough, avec une bijection entre l'espace de l'image et l'espace des paramètres représentant une droite, à savoir sa pente et sa distance à l'origine. Nous passons ensuite à la détection de cercles et ses trois paramètres – centre et rayon – et l'appliquons à la mesure d'angle de contact d'une goutte sur une surface. Finalement, une conique – représentant un réflecteur ponctuel dans une image RADAR où l'antenne est mobile – illustre le cas de quatre paramètres à identifier.

Mots-clés : transformée de Hough, détection lignes, détection cercles, détection coniques, RADAR, angle de contact, tension de surface

Résumé

La transformée de Hough propose un passage de l'espace bidimensionnel de l'image acquise par un capteur optique (caméra) vers l'espace des paramètres représentant des structures géométriques recherchées dans l'image – droites, cercles, et coniques dans les exemples qui sont abordés. Cette méthode de traitement d'images, généralisable à tout motif paramétrable par un jeu d'équations, est présentée sur trois cas concrets d'utilisations sur des images réelles.

De nombreuses applications de traitement d'images nécessitent de trouver des motifs géométriques dans une image, qu'il s'agisse du cas trivial de lignes (navigation d'un véhicule sur une chaussée marquée de lignes), ou plus complexe tel que nous l'illustrerons ici, de cercles ou de coniques. La transformée de Hough **[1][2]** vise à convertir une image dont les pixels représentent une distribution spatiale

d'intensité lumineuse (sortie d'un capteur d'intensité lumineuse derrière un montage optique de focalisation) vers un espace représentatif des paramètres de la structure géométrique recherchée. Par exemple, une droite dans l'espace **(x, y)** s'exprime par $y = a \times x + b$: la transformée de Hough dans l'espace des paramètres de la droite passe de **(x, y)** vers **(a, b)**. Le cas de la droite est simple, car nous passons de deux dimensions à deux dimensions, facilement représentable sur une image. Le cas du cercle ou de la conique sera à peine plus complexe puisque l'espace d'arrivée est à plus de deux dimensions : pour un cercle, nous aurons les coordonnées de son centre **(x_c, y_c)** et son rayon **R**. Une façon d'ajouter cette troisième dimension consiste à réaliser un film dans lequel **R** évolue dans le temps.

La philosophie de la transformée de Hough est de partir de l'espace d'arrivée – celui des paramètres du motif géométrique recherché – et de tester de façon exhaustive si les pixels de l'image d'entrée sont distribués sur le motif recherché. Pour chaque série de paramètres, la somme des intensités des pixels issus du capteur optique est assignée au point correspondant au jeu de paramètres en cours : la somme ne deviendra significative que si les pixels sont « convenablement » distribués le long du motif. Cette opération peut devenir très fastidieuse si nous testons tous les pixels possibles : pour une image de **N x N** pixels, nous devrons sommer les N^2 pixels pour N^2 valeurs des paramètres. Un motif ne concerne cependant que les bords de l'objet détecté : une balle n'est ronde que parce que sa circonférence représente un cercle, tandis que le motif de la balle elle-même est sans intérêt. Il est donc classique de précéder la transformée de Hough d'une détection de bords, dont l'implémentation la plus simple est le gradient. En effet, un « bord » se définit comme une rupture entre un premier motif (couleur de fond) et un second motif (couleur de l'objet), donc une dérivée significative. Une dérivée spatiale est un gradient, que nous calculons selon les deux directions de l'image **X** et **Y**, pour en déduire le critère de bord de l'image pour chaque pixel **P** : $bord(x,y) = \nabla_x^2 P(x,y) + \nabla_y^2 P(x,y) = (P(x+1,y) - P(x,y))^2 + (P(x,y+1) - P(x,y))^2$ Un seuil permet alors de ne sélectionner que les pixels pertinents et de réduire considérablement le temps de calcul, et la robustesse de la détection, en ne conservant que le pourtour des objets à détecter.

Nous allons appliquer ces concepts à trois cas concrets : la détection de droites dans les images utilisées notamment pour diriger un véhicule (détection des bandes de signalisation sur une route), la détection de cercles dans les images de gouttes d'eau déposées sur des surfaces dont l'énergie est représentative des fonctions chimiques en contact avec le solvant (méthode de l'angle de contact pour caractériser une surface), et finalement détection d'hyperbole dans les images issues de RADAR dont l'antenne se déplace linéairement au-dessus d'une cible ponctuelle.

1 Détection de droites

Nous allons nous familiariser avec un cas trivial de détection de droites dans une image réalisée spécifiquement à cet effet (voir figure 1). Formatés que nous sommes à penser en coordonnées cartésiennes, il semble naturel d'exprimer l'équation d'une droite sous forme $y = a \times x + b$ et rechercher les couples **(a, b)** qui correspondent le plus probablement aux droites visibles sur une image. Une mise en pratique démontre rapidement les faiblesses de cette approche : alors que le coefficient **b**, l'ordonnée à l'origine, est en unité de pixels et s'intègre naturellement dans le traitement, le coefficient **a** varie dans le premier octant de **0** à **1** (une variation peu favorable au calcul sur des entiers), puis dans le second octant de **1** à $+\infty$ (cette seconde borne étant très difficile à atteindre en pratique)... l'expression de **a** comme pente de la droite ne s'exprime pas simplement dans une distribution linéaire de valeurs du paramètre sur un nombre fini d'échantillons discrets le long d'un des axes dans le domaine de la transformée de Hough. Une façon plus intuitive **[3]** consiste à exprimer l'angle ϑ que fait la droite considérée avec l'axe des abscisses : évidemment **a** et ϑ sont reliés par le fonction **tan()**, mais cette fonction allant de $-\infty$ à $+\infty$ de façon non-linéaire, il sera plus simple d'exprimer ϑ – distribué régulièrement de $-\pi$ à $+\pi$ selon l'axe des abscisses du domaine de destination (de Hough) que d'exprimer **a**. La seconde variable dans l'expression polaire de la droite est ρ, la distance de la droite à l'origine. Dans ce contexte, une droite de la forme $y = a \times x + b$ s'exprime aussi $x \times \cos(\vartheta) + y \sin(\vartheta) = \rho$ et le domaine de Hough consiste à transformer l'ensemble des points remarquables **(x, y)** vers les valeurs les plus probables de (ϑ, ρ) définissant les droites selon lesquels les points d'intérêt se distribuent.

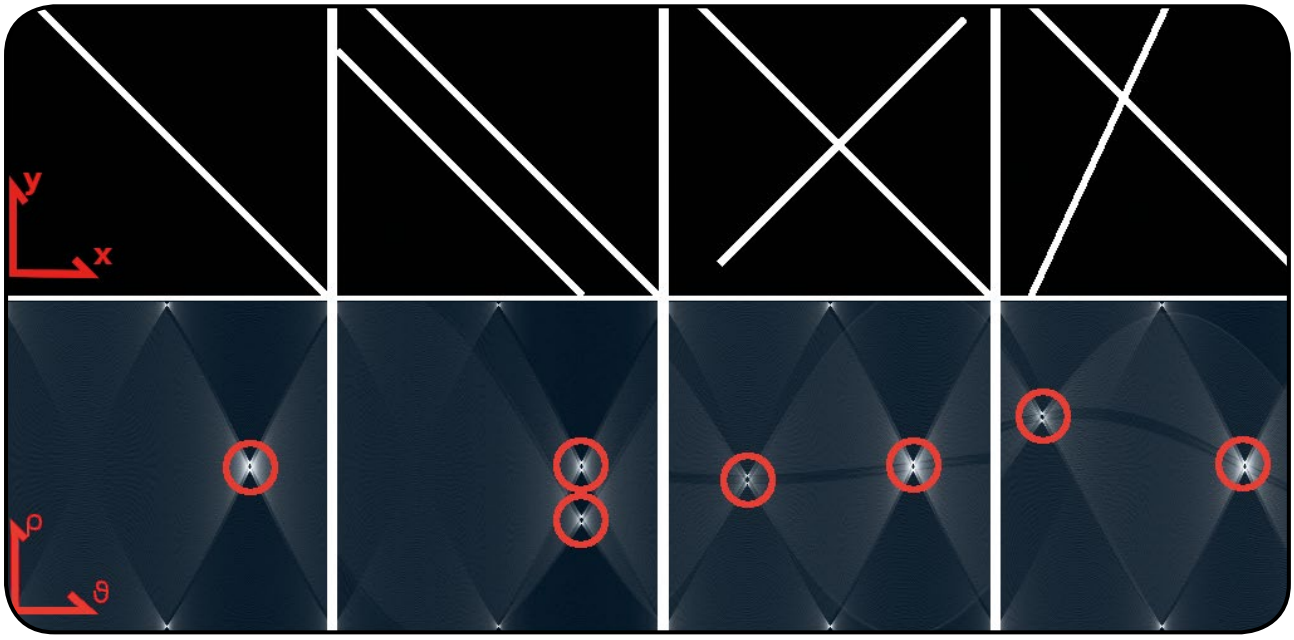


Fig. 1 : De gauche à droite : image d'une route (avenue de l'Opéra) avec sa signalétique ; identification des droites remarquables dans une image synthétique représentative des caractéristiques de la photographie originale.

Nous nous familiarisons dans un premier temps sur des images synthétiques (voir figure 2) avant d'appliquer ce traitement à une image réelle (figure 1). Dans ce premier exemple, nous nous sommes efforcés d'indiquer par un cercle les points caractéristiques dans le domaine de Hough des paramètres polaires (ϑ, ρ) représentatifs de(s) droite(s) visible(s) en blanc sur un fond noir. Le premier cas montre bien un unique point de forte probabilité dont l'abscisse représente la pente et l'ordonnée la distance à l'origine. Le second cas démontre la pertinence de l'analyse puisqu'une seconde droite parallèle à la première indique un second jeu probable de paramètres représentatifs des structures géométriques, à la même abscisse (même pente), mais d'ordonnées différentes (distance à l'origine différente). Les troisième et quatrième cas démontrent que l'abscisse des paramètres est bien la pente de la droite puisque changer cette pente fait varier l'abscisse d'un des jeux de paramètres probables.

En manipulant une structure de données contenant une image codée en tons de gris sur un octet (***p**) et ses dimensions (**x** et **y**) :

```
struct image {unsigned char *p;int x; int y;int bpp;};
```

Alors, le gradient de l'image d'entrée s'obtient dans une nouvelle structure de données par :

```
01: pictin=(struct image*)malloc(sizeof(struct image));
02: pictin->x=pict->x;pictin->y=pict->y;pictin->bpp=255;
03: pictin->p=(unsigned char*)malloc(sizeof(unsigned char)*pictin->x*pictin->y);
04: for (x=0;x<pictin->x-1;x++)
05:  for (y=0;y<pictin->y-1;y++) // calcul du gradient : P sur 1 octet donc P*P/255\
in[0..255]
06:    {pictin->p[x+y*pictin->x]=(((pict->p[x+1+y*pict->x]-pict->p[x+y*pict->x])*\
07:         (pict->p[x+1+y*pict->x]-pict->p[x+y*pict->x]) \
08:         +(pict->p[x+(y+1)*pict->x]-pict->p[x+y*pict->x])*\
09:         (pict->p[x+(y+1)*pict->x]-pict->p[x+y*pict->x]))/255);
10:  if (pictin->p[x+y*pictin->x]>max) max=pictin->p[x+y*pictin->x];
11:    }
12: for (x=0;x<pictin->x-1;x++)
13:  for (y=0;y<pictin->y-1;y++)
14:    if ((float)pictin->p[x+y*pictin->x]>((float)max*0.01)) // sature le gradient
15:   pictin->p[x+y*pictin->x]=255; else pictin->p[x+y*pictin->x]=0;
```

Puis, ayant obtenu cette image des gradients saturée qui indique les points significatifs, nous appliquons la transformée de Hough elle-même, en exploitant une table de sinus et de cosinus pré-calculée puisque les angles pour lesquels se fait la recherche sont déterminés par le nombre de pixels en abscisse de l'espace d'arrivée :

```
01: float dtheta,theta,*smat,*cmat,tmp;
02: dtheta=M_PI/(float)(psize);
03: smat=(float*)malloc(sizeof(float)*psize); // x*cos(th)+y*sin(th)=rho
04: cmat=(float*)malloc(sizeof(float)*psize); // th selon l'abscisse : precalcule sin() et cos()
05:
06: tmpi=(int*)malloc(sizeof(int)*psize*psize); max=0;
07: for (i=0;i<psize*psize;i++) tmpi[i]=0;
08: i=0;
09: for (theta=0;theta<M_PI;theta+=dtheta) {smat[i]=sin(theta);cmat[i]=cos(theta);i++;}
10: for (i=0;i<psize*psize;i++)
11:  {if (pictin->p[i] == 255) // ne garde que points significatifs
12:    {for (x=0;x<psize;x++) // % : modulo = x et / : division entiere = y
13:     {tmp=(float)(i/psize-(psize>>1))*smat[x]+
14:       (float)(i%psize-(psize>>1))*cmat[x]+(float)(psize>>1);
15:      if (( (int)rint(tmp)<psize) && ((int)rint(tmp)>0) )
16:       tmpi[(int)rint(tmp)*psize+x]++; // note de chaque couple de params (th,rho)
17:     }
18:    }
19:  }
```

Ici **psize** est la grandeur maximum entre la hauteur et la largeur de l'image d'entrée.

Comment ces figures dans le domaine de Hough ont été obtenues ? Dans un premier temps, les points remarquables sont identifiés, ici trivialement par seuillage puisque nous sommes dans un cas idéal sans bruit, avec des droites blanches sur fond noir. L'espace de destination est une nouvelle image, initialement vide (initialisée à **0**), de la même taille que l'espace de départ (par homogénéité avec les paramètres exprimés en pixels), mais cette fois dont l'ordonnée représente ρ et l'abscisse. Pour toute valeur de ϑ (boucle sur l'axe des abscisses), et pour chaque point remarquable de l'image d'entrée, nous ajoutons une unité à l'ordonnée ρ tel que $\rho = x\cos\vartheta + y\sin\vartheta$. Nous prenons bien sûr soin de vérifier que ρ est positif et inférieur à l'ordonnée maximale de l'image d'arrivée pour éviter d'accéder à une zone non-allouée de la mémoire. Cette accumulation de paramètres probables est itérée pour toutes les colonnes de l'image d'arrivée (boucle sur ϑ) et pour chaque point remarquable de l'image de départ.

À la fin, l'image d'arrivée présente des pixels d'autant plus clairs que le jeu de paramètres correspondant (ϑ, ρ) est plus probable : un seuillage dans cet espace d'arrivée donne les paramètres des droites identifiées dans l'image de départ.

Le cas d'une « vraie » image est rendu à peine plus complexe par l'étape de prétraitement qui doit permettre d'identifier les pixels distribués le long des droites avant d'en extraire les paramètres par transformée de Hough : dans ce cas, les bords des bandes qui nous intéressent sont détectés par des gradients (passage du blanc au noir) selon les directions des abscisses et des ordonnées. Le critère critique dans l'identification des bandes est le seuil sur le gradient : ici ce seuil est sélectionné manuellement, et un algorithme robuste de seuillage en illumination naturelle est complexe à identifier **[4]**. Dans ce cas particulier, le résultat est convaincant, et une comparaison entre la transformée de Hough du gradient de l'image et un cas synthétique dans lequel les droites ont été tracées manuellement sur un fond noir (cas idéal non-bruité) permet de retrouver quelle droite correspond à quel jeu de paramètres dans l'espace de Hough (voir figure 1, droite).

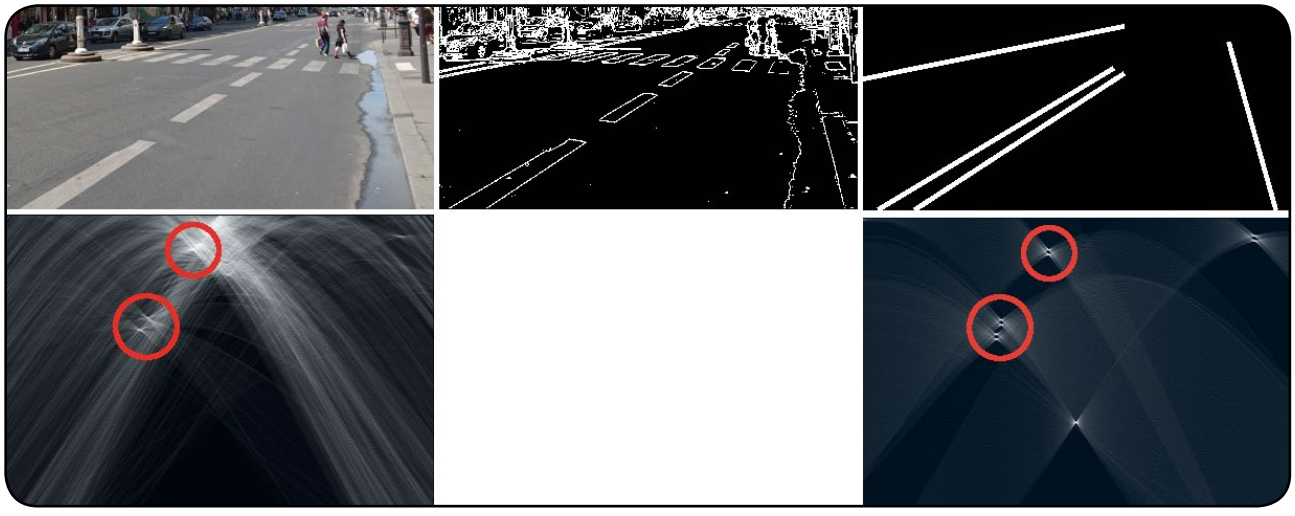


Fig. 2 : De gauche à droite : divers cas de droites avec pentes (abscisse dans la transformée polaire de Hough) et distance à l'origine (ordonnée dans la transformée polaire de Hough) variables.

Nous verrons que ces calculs deviennent d'autant plus longs que l'espace des paramètres est grand. Ici, seuls deux paramètres (ϑ, ρ) sont recherchés, mais sur une image de résolution typique d'un capteur actuel (8, 10 Mpixels), le calcul prend déjà plusieurs secondes, un délai inacceptable dans de nombreuses applications de navigation. Notre choix s'est donc porté directement sur une implémentation en **C** au lieu d'un prototypage sous **GNU/Octave** comme nous le verrons plus loin (cas des hyperboles) : les sources permettant de lire une image au format BMP, calculer les gradients en abscisse et ordonnée, *seuiller* puis calculer la transformée de Hough, sont disponibles sur http://jmfriedt.free.fr/hough.

2 Une surface propre est une surface qui mouille

2.1 Paramètres du cercle représentant au mieux une calotte sphérique

Ayant compris le concept sous-jacent à la transformée de Hough – à savoir accumuler les valeurs les plus probables des paramètres représentant les courbes selon lesquelles s'agencent les points remarquables d'une image – nous pouvons étendre le principe à la recherche de cercle. Un cercle est caractérisé par son centre (x_c, y_c) et son rayon **R**, reliés par l'équation $(x-x_c)^2 + (x-x_c)^2 = R^2$. Un ensemble de points de coordonnées (x, y) supposés distribués sur la circonférence d'un cercle doivent respecter cette équation, donc nous recherchons les valeurs communes de $\{x_c, y_c, R\}$. S'agissant de trois paramètres (donc un espace à 3 dimensions dans lequel nous recherchons une solution optimale), il n'est pas facile de le représenter sur une image (de dimension 2). Nous nous proposons donc d'explorer l'espace (bidimensionnel) des centres possibles du cercle (x_c, y_c) pour un paramètre qui évolue dans le temps qu'est le rayon **R**. Ainsi, pour chaque valeur de **R**, la transformée de Hough consiste à accumuler les valeurs possibles de $y_c = y - \sqrt{R^2 - (x - x_c)^2}$ avec x_c l'abscisse dans le domaine de Hough et y_c son ordonnée, pour chaque point remarquable (x, y). La valeur de **R** qui permet de maximiser un couple (x_c, y_c) fournit le triplé des paramètres caractérisant le cercle recherché. Mais pourquoi rechercher un cercle dans une image ?

Une application concrète de cet algorithme est proposée dans le contexte de l'analyse de l'énergie de surface d'un substrat, mesuré par l'angle de contact ϑ de solvants déposés par la surface. L'angle que fait une goutte de solvant déposée sur une surface est en effet déterminé par cette énergie de surface, qui définit donc la propension d'une surface à se « salir » (interagir avec son environnement) **[5]**. Nombreuses sont les études visant à *fonctionnaliser* les surfaces pour en faire varier l'énergie, que ce soit pour les rendre adhérentes (lors d'un vernissage ou d'un dépôt de peinture par exemple) ou au contraire répulsives (surface d'une poêle qui ne doit pas accrocher les aliments) **[6][7]**. Au niveau du contact de la goutte avec la surface, trois interfaces cohabitent : la surface solide, la surface du liquide et le gaz environnant. Ces trois interfaces possèdent chacune une énergie de surface, et la condition d'équilibre impose (équation de Young) :

$$\gamma_{solide-gaz} - \gamma_{solide-liquide} - \gamma_{liquide-gaz} \cos\vartheta = 0$$

Donc déterminer ϑ permet de déduire les propriétés de la surface considérée $\gamma_{solide-gaz}$. Une valeur faible de tension de surface correspond à une faible mouillabilité et donc une surface fortement hydrophobe (par exemple le teflon), tandis qu'une valeur élevée correspond à une forte affinité à interagir avec l'environnement et donc une surface hydrophile. Les traitements de surface

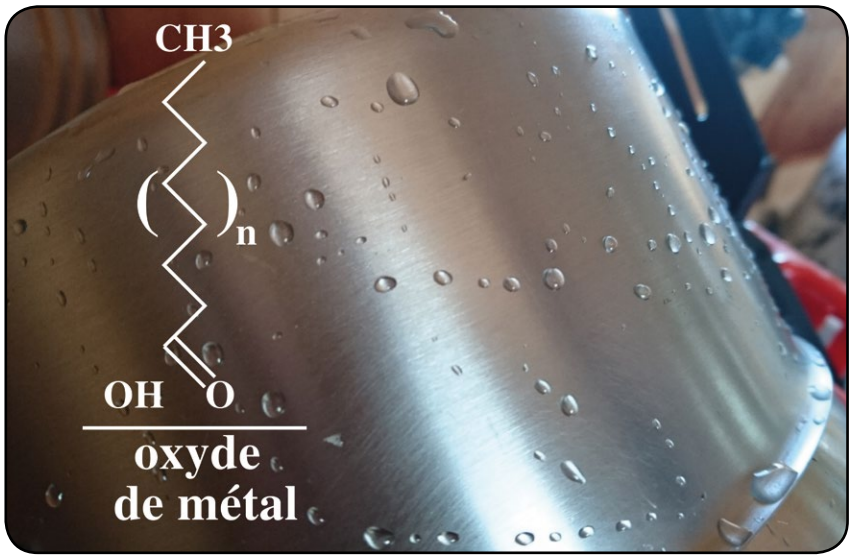


Fig. 3 : Exemple de surface d'ustensile de cuisine encore couvert d'une couche grasse, ne permettant pas aux gouttes d'eau de s'étaler sur la surface d'acier inoxydable. Les groupes acides carboxyliques, hydrophiles, s'orientent vers l'oxyde métallique présent en surface de la casserole, exposant les groupes hydrophobes qui expliquent l'angle de contact élevé observé.

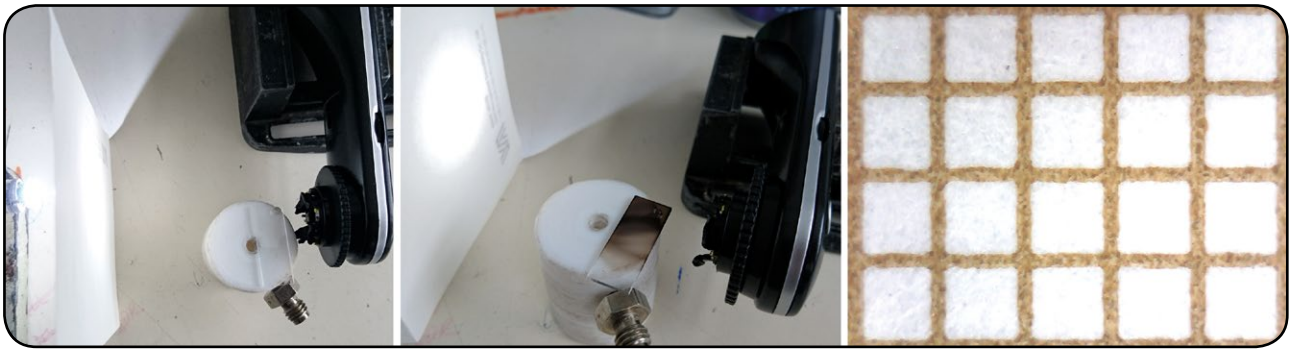


Fig. 4 : Montage expérimental de prises de vues – une webcam munie d'un objectif dont la distance focale est compatible avec des applications de macrophotographie – avec une illumination en contre-jour pour saturer en blanc le fond de l'image et faire apparaître la goutte en sombre sur fond clair. Nous constatons en prenant une photographie de papier millimétrique gradué (à droite) l'absence de déformation significative par l'optique de prise de vue.

visant à rendre les surfaces mouillables (par exemple pour les enduire d'une couche de vernis), soit en les chauffant soit en les irradiant, augmentent donc l'énergie de surface γ avant *fonctionnalisation*.

L'énergie de surface est dominée par les groupements chimiques en surface de l'objet considéré **[8]**, forte pour des groupes tels que carboxyle ($\vartheta \simeq 20$–$60°$ **[9]**) ou alcool ($\vartheta \simeq 70°$ **[10]**), mais faible pour le methyl ($\vartheta \simeq 112°$) et surtout les groupes chlorés ou fluorés ($\vartheta \simeq 118°$, voire 108–110° pour le teflon), avec une contribution relativement secondaire de la longueur de la chaîne carbonée qui relie ces groupes terminaux à la surface **[9]**.

Concrètement, tout le monde connaît empiriquement les conditions pour une surface « propre » (voir figure 3). Une surface « propre » est une surface sur laquelle une goutte d'eau s'étale, une surface « sale » (par exemple, grasse) est une surface sur laquelle une goutte d'eau reste sous forme de goutte. Au-delà de cette approche qualitative intuitive, nous pouvons formaliser ces concepts pour les rendre quantitatifs au travers des énergies de tension de surface qui définissent l'angle de contact d'un solvant avec une surface. Un paramètre amplificateur de cet effet est la rugosité dont nous discuterons plus loin.

Le montage expérimental pour mesurer une énergie de surface au travers de l'angle de contact ϑ consiste à déposer une petite **[11]** (<10µl) goutte de liquide sur la surface à analyser, prendre une photographie en contre-jour de la goutte (voir figure 4), et analyser son angle de contact avec la surface. L'angle de contact se détermine en ajustant un cercle sur la circonférence de la goutte – supposée suffisamment petite pour que la gravité soit négligeable sur les forces de tension superficielles et que la goutte soit donc de forme cylindrique – et en calculant l'angle à l'intersection avec le plan formé de l'échantillon qui supporte la goutte. Pour notre part, une webcam munie d'un objectif compatible avec une mesure en macrophotographie **[12]** et commandée par **vlc** permet d'acquérir films et captures d'écran pour un traitement des images présenté ci-dessous.

La séquence de traitements que nous vous proposons de suivre pour identifier l'angle de contact d'une goutte avec une surface est :

1. Identifier le plan de la surface, supposée horizontale, comme maximum du gradient ∇_y^2 selon l'axe vertical. En effet, le plan présente une rupture nette en se déplaçant verticalement le long de l'image ;
2. Identifier les limites de la goutte, en particulier ses bords haut, droit et gauche, pour circonscrire la zone d'analyse, et réduire les risques de trouver des cercles qui ne correspondent pas à la goutte étudiée. Ces bords se déduisent du plan qui a été identifié dans la première étape, en faisant l'hypothèse que la goutte se trouve au-dessus du plan la supportant, et que ses bords droit et gauche correspondent au maximum du gradient ∇_x^2 selon l'axe horizontal ;
3. Partant d'une hypothèse « raisonnable » sur le rayon probable de la goutte compte tenu des considérations géométriques précédentes, nous appliquons l'algorithme de Hough aux trois paramètres que sont le centre du cercle et son rayon ;
4. Ayant identifié le centre du cercle (x_c, y_c), son rayon R, et l'ordonnée du plan y_p sur lequel repose la goutte, l'angle de contact ϑ se déduit trivialement par $\sin(\vartheta) = (y_p - y_c)/R$.

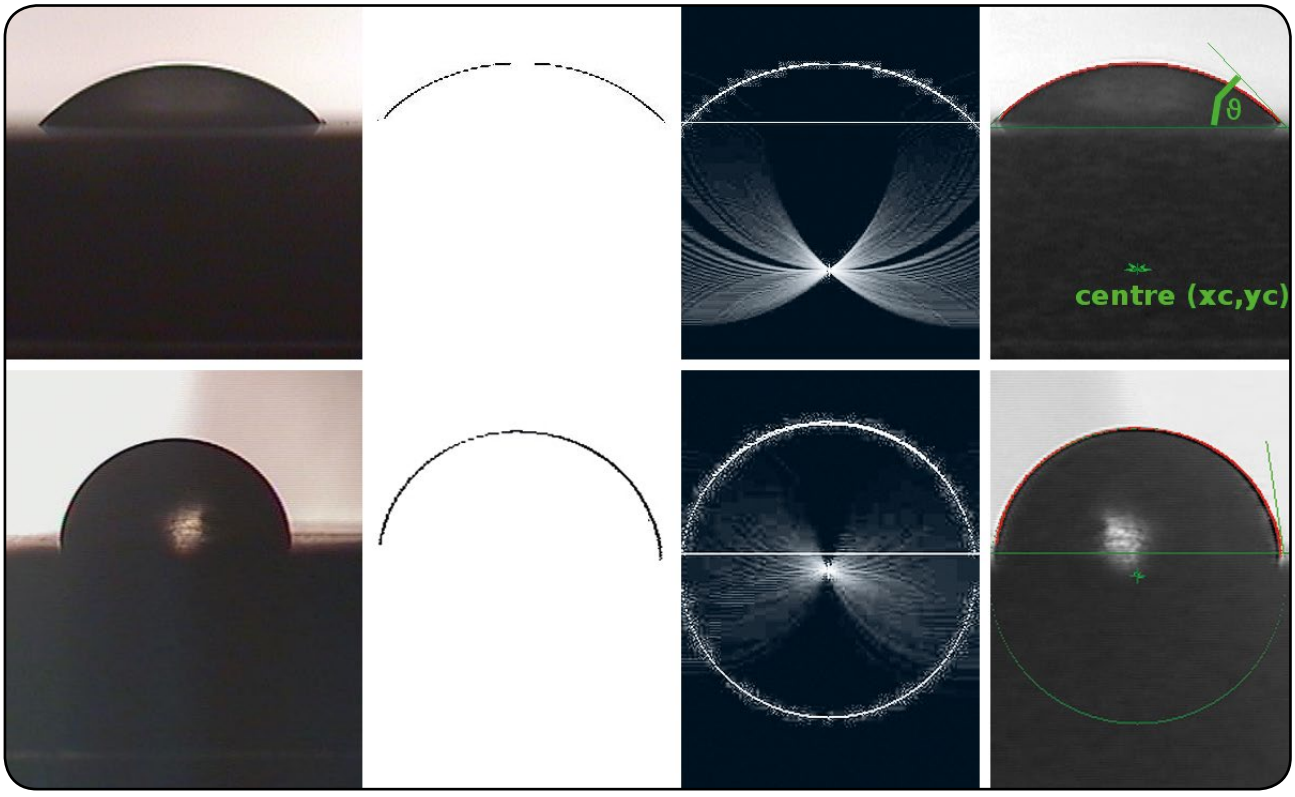


Fig. 5 : Séquence de traitements pour identifier l'angle de contact de deux gouttes d'eau sur des surfaces de polymères. De l'image brute, les points remarquables de la circonférence de la goutte sont identifiés par détection de contour (gradient selon les deux directions de l'image), puis transformée de Hough sur des itérations des divers rayons du cercle avec recherche, pour chaque rayon, du centre du cercle le plus probable. Finalement, le calcul de la tangente au point d'intersection de la goutte avec le plan qui la supporte donne, dans le cas du haut, 44,3° et dans le cas du bas, 83,4°.

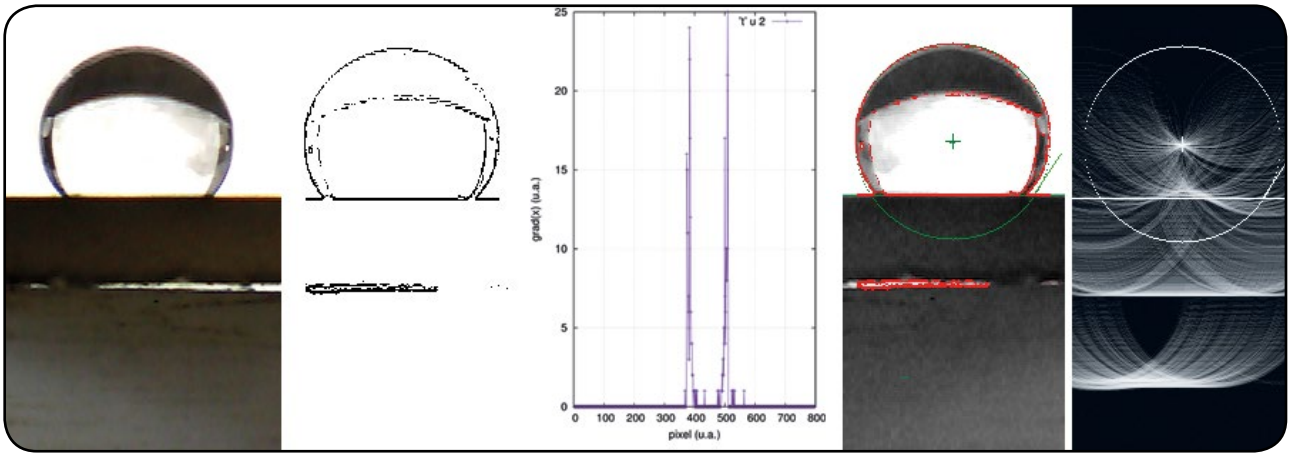


Fig. 6 : De gauche à droite : image originale acquise par caméra connectée sur port USB ; gradient sur la zone identifiée comme contenant la goutte ; évolution du gradient le long de l'axe des abscisses, illustrant clairement les deux maxima lorsque les bords de la goutte sont parcourus ; somme des contributions des pixels dans le domaine de Hough des paramètres du centre du cercle, pour la valeur du rayon maximisant la somme des intensités des pixels ; identification du cercle représentant la goutte (cercle vert) et tracé de la tangente au point de contact (droite verte). Noter le choix peu judicieux du support de la goutte qui ne couvre pas toute la largeur de la photographie, se traduisant par un gradient parasite dans la direction horizontale. Dans cet exemple, l'angle de contact est de 123,4°, une valeur élevée, mais qui n'est pas aberrante pour une lame de verre silanisée avec une terminaison fluorée.

Le résultat de ces traitements est illustré sur les figures 5 et 6, avec en bout de chaîne la superposition du cercle identifié comme représentant au mieux la forme de la goutte, la tangente au point d'intersection avec le plan supportant la goutte, et par conséquent l'angle de contact.

2.2 Fonctionnalisation d'une surface

Une approche opposée à la surface propre parce qu'hydrophile consiste à dire que toute saleté en solvant aqueux doit être repoussée par l'objet que nous voulons maintenir « propre ». Pour ce faire, une surface hydrophobe propose les propriétés requises. Un exemple trivial est la poêle recouverte de teflon pour empêcher les matières grasses de s'y fixer. Cependant, de même que le teflon ne veut pas que les matières grasses s'y fixent, une couche de teflon tend à se détacher de la surface qu'elle est supposée protéger. Une approche alternative consiste à générer une surface d'hydrophobicité médiocre, mais stable, et d'amplifier l'effet d'hydrophobicité par une rugosité aux échelles micrométriques et en deçà **[13]**. Cette approche est celle mise en œuvre dans les sprays « anti-salissants » à la mode, qui consistent à déposer des nanoparticules sur les surfaces à protéger (par exemple UltraTech Ultra-Ever Dry).

En effet, la rugosité de surface tend à amplifier la propriété d'hydrophobicité (ou d'hydrophilicité) selon l'équation de Wenzel qui affirme que pour une surface de rugosité **r**, l'angle de contact ϑ^* est relié à l'angle de contact ϑ de la même surface lisse par $\cos(\vartheta^*) = r \times \cos(\vartheta)$.

Un mécanisme pour rendre une surface hydrophobe est de la couvrir de suie de bougie ou d'un solvant organique (voir figure 7) **[14]** : les nano-particules ainsi déposées sont couvertes de sites hydrophobes ($-CH_3$) qui rendent la surface répulsive pour la goutte d'eau. Dans le cas des surfaces super-hydrophobes, la goutte reste sphérique sous

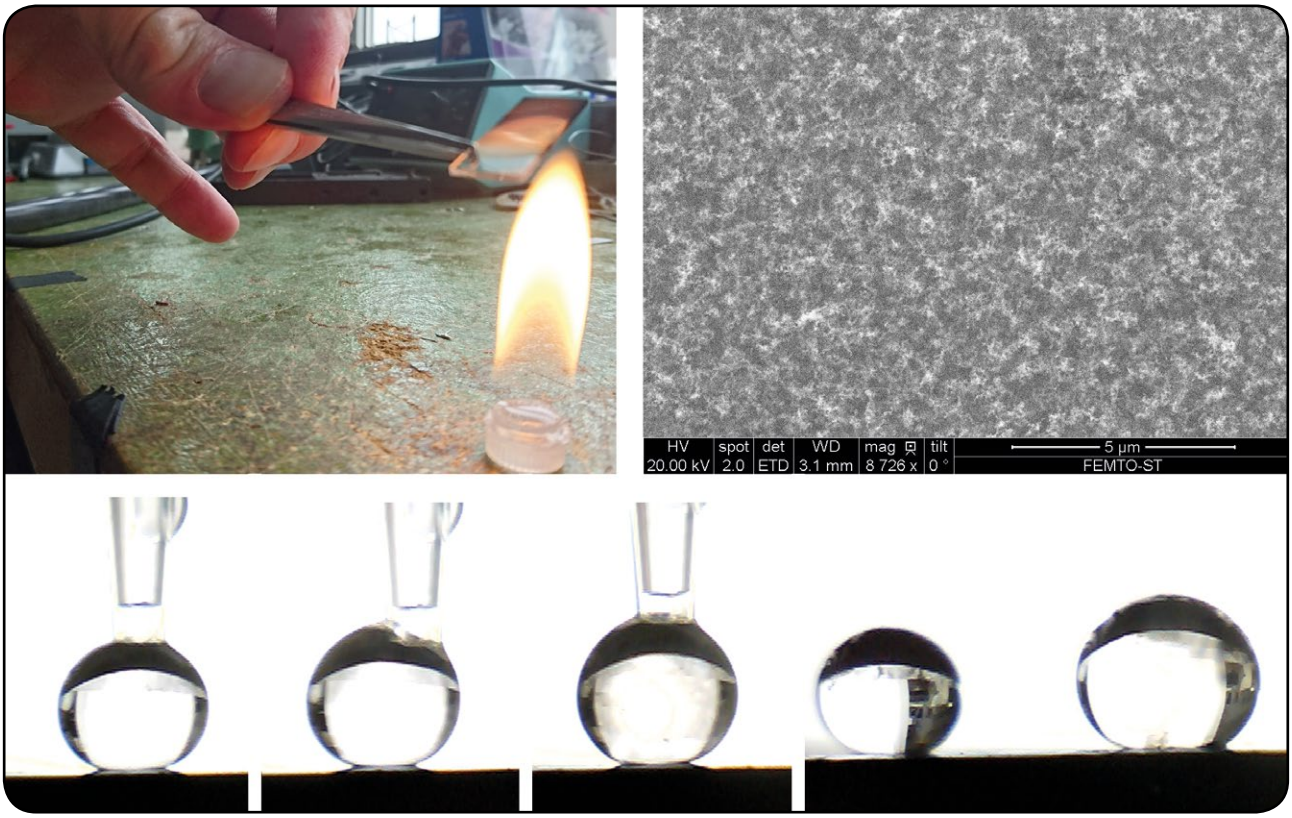


Fig. 7 : Haut, gauche : fonctionnalisation d'une surface de verre « propre » par de la suie produite par une flamme de solvant organique (isopropanol) **[14]**. *Droite : image au microscope électronique à balayage de la surface de verre couverte de suie. Aucune particule de dimensions micrométriques ou supérieures n'est observée, la structuration de la surface se fait sur une échelle de quelques dizaines à la centaine de nanomètres. Bas : séquences d'un film visant à déposer une goutte d'eau sur cette surface super-hydrophobe. Non seulement la goutte refuse de se détacher de la micropipette, mais une fois libérée de l'embout, la goutte roule pour s'enfuir rapidement du champ de vision de la caméra.*

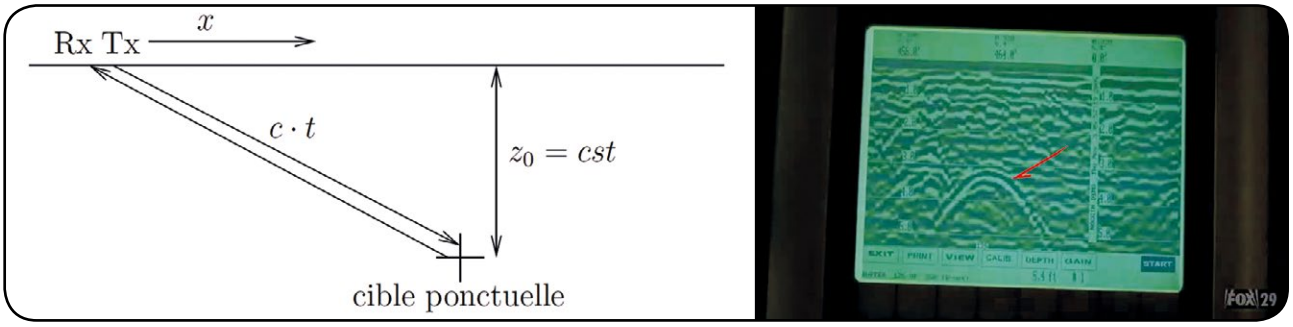


Fig. 8 : Gauche : schéma justifiant de l'observation de réflecteurs ponctuels comme hyperbole dans un RADARgramme de RADAR à ouverture synthétique ou RADAR de sol (GPR). Rx et Tx sont l'émetteur et le récepteur du RADAR dans une configuration bistatique. Droite : capture d'écran de l'épisode 7 de la saison 1 de la série télévisée Bones (34 minutes, 11 secondes du début de l'épisode) illustrant la distribution hyperbolique des réflexions de l'objet enterré (flèche rouge), observée par GPR par l'héroïne de la série.

réserve que son diamètre soit nettement plus petit que la longueur capillaire (donc avec une contribution négligeable de la gravité devant la tension de surface), et l'algorithme de détection de cercles de Hough fonctionne parfaitement.

Le lecteur désireux de ne pas développer ses propres implémentations des algorithmes, mais de se contenter d'expérimenter avec la détection de cercles et de droites pourra avantageusement exploiter la fonction **houghtf()** fournie avec la bibliothèque de traitement d'images de GNU/Octave.

3 Traitement d'images issues de RADAR à ouverture synthétique

Le dernier exemple que nous proposons est le cas d'un réflecteur ponctuel sondé par un dispositif mobile illuminant cette cible et observant les signaux réfléchis – cas typique d'un RADAR mobile, qu'il s'agisse de RADAR de sol (*Ground Penetrating RADAR* – GPR) ou RADAR à ouverture synthétique (SAR) dans lequel une antenne d'ouverture équivalente immense est synthétisée en déplaçant l'instrument le long d'une trajectoire supposée rectiligne **[15]**. Dans ce cas, pour une cible placée à une distance z_0 de la trajectoire (profondeur z_0 dans le cas du GPR qui se déplace à la surface du sol), le temps de vol de l'écho **t** en fonction de la position **x** de la source RADAR – en supposant une célérité de l'onde électromagnétique de **c** – vérifie $c^2 \cdot t^2 = x^2 + z_0^2$ (voir figure 8). Ainsi, un réflecteur ponctuel de profondeur z_0 sous terre se traduit, pour une mesure GPR, par une hyperbole dans le plan **(x, t)** puisque $\frac{c^2}{z_0^2} \cdot t^2 - \frac{x^2}{z_0^2} = 1$ avec z_0 constante est l'équation d'une hyperbole. Dans ce cas, la transformée de Hough se doit d'identifier quatre paramètres ; le décalage en abscisse et en ordonnée **(x_0, y_0)**, le centre **c** et la pente des asymptotes **p** ou en d'autres termes les coefficients **a** et **b** de l'hyperbole (la relation entre ces quatre termes étant **p = b / a** et **c^2 = a^2 + b^2** : $(y - y_0)^2/a^2 - (x - x_0)^2/a^2 = 1$. Ce problème est donc très similaire au cas du cercle, si ce n'est que **a** et **b** sont moins intuitifs que **R** le rayon du cercle.

L'application du même algorithme que précédemment, mais cette fois en choisissant les points remarquables le long des hyperboles **(x, y)** et en cherchant à maximiser la note **[16]** du jeu de paramètres **(x_0, y_0, a, b)** en balayant (boucles) **a** et **b** et en traçant pour chacun de ces couples l'image dans le domaine de Hough fournissant la note de chaque couple **(x_0, y_0)** s'obtient par $y_0 = y - a \cdot \sqrt{1 + (x - x_0)^2/b^2}$. Dans ce cas, x_0 correspond à l'abscisse dans l'image de Hough et les y_0 correspondants s'accumulent pour donner le résultat de la figure 9. Comme nous pouvons nous y attendre, la précision sur les valeurs de **a** et **b** est peu précise, mais avec une valeur maximale cohérente lorsque **a = b** qui correspond à une pente de l'asymptote parallèle à la première bissectrice, et une note qui chute rapidement lorsque **a** $\neq$ **b**.

```
01: clear all;close all
02: x=[-10:0.1:10]; a=5; b=5; % params de l'hyperbole tracée
03:
04: % on fabrique une image avec une hyperbole
05: y=-a*sqrt(1+x.^2/(b*b)); % y^2/a^2-x^2/b^2=1 ie y=+/-a*(1+x^2/b^2)
06: plot(x,y);hold on
07: y=-a*sqrt(1+(x-3).^2/(b*b))+4; % 2eme hyperbole décalée de
x0=3,y0=4
08: plot(x,y);axis off;colormap gray
09: print -dbmp hyperbole.bmp
10:
11: % on lit une image avec une hyperbole
```

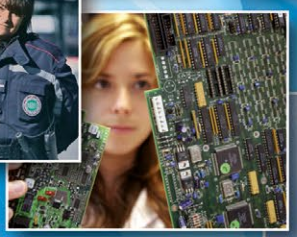
DOUANES & DROITS INDIRECTS

INFORMATICIENS

REJOIGNEZ LA DOUANE !

Mettez vos talents au service de la protection du territoire.

RÉSEAU INTERNATIONAL DE LA DOUANE



PROTÉGER LES CITOYENS ET L'ENVIRONNEMENT

VALIDATION

PARTICIPER AU FINANCEMENT DES SERVICES PUBLICS

RECHERCHE

ACCOMPAGNER LES ACTEURS DU COMMERCE MONDIAL

Contactez Infos Douane Service

0811 20 44 44 Service 0,06 € / min + prix appel

#LaDouaneRecrute

douane.gouv.fr / recrutement

Sur iOS et Android : douanefrance.mobi

@douane_france

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE DES FINANCES ET DES COMPTES PUBLICS

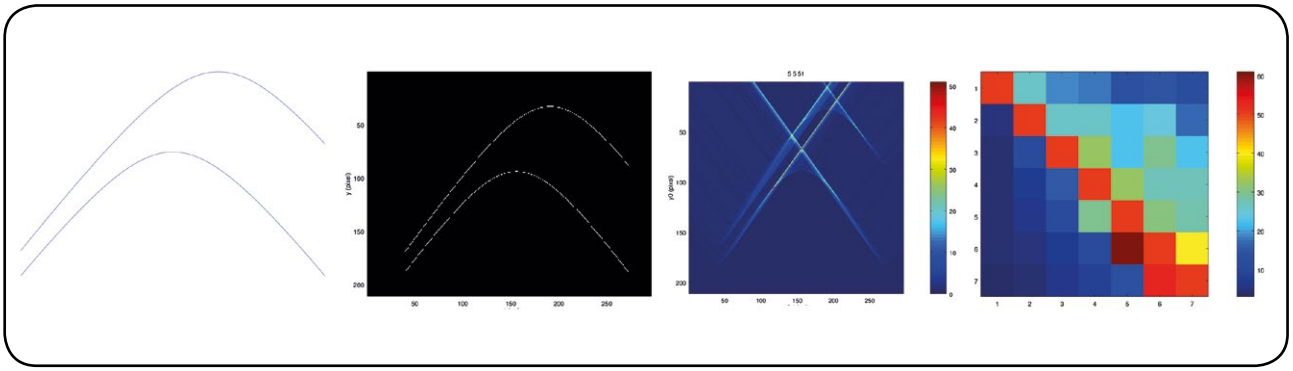


Fig. 9 : De gauche à droite : l'image originale considérée comportant deux hyperboles ; l'identification des points remarquables, ici par simple seuillage puisque dans ce cas synthétique les données ne sont pas bruitées ; la transformée de Hough présentant la probabilité de (x_0, y_0) *d'être le centre de la parabole pour une paire de* **a** *et* **b** *égaux (cas le plus favorable dans cet exemple) ; et finalement la note maximale de la transformée de Hough en fonction des valeurs de* **a** *(en abscisse) et* **b** *(en ordonnée).*

```
12: entree=imread('hyperbole.bmp'); % relit l'image generée
13: entree=entree(1:4:end,1:4:end,2);% trop grand : ne garde que
1/4 de l'img
14: figure;imagesc(entree); colormap(gray)
15: [y,x]=find(entree>0); %figure; plot(x,y) % points remarquables
16: % a=5;b=5;
17: for a=1:7 % recherche des couples (a,b)
18:  for b=1:7
19:    [sy,sx]=size(entree); espace=zeros(sx,sy);
20:    for x0=1:sx % recherche du centre pour chaque (a,b)
21:     for k=1:length(x)
22:       tmp=y(k)-a*sqrt((x(k)-x0).^2/(b*b)+1); % (y-y0)^2/a^2-
(x-x0)^2/b^2=1
23:       tmp=floor(tmp); % ie y-y0=a*sqrt([1+(x-x0)^2/b^2])
24:       if ((tmp>0) && (tmp<sy)) espace(x0,tmp)=espace(x0,tmp)
+1;endif
25:     end % ^^^ note de chaque jeu de params
26:    end
27:    figure;imagesc(espace');
28:    m(a,b)=max(max(espace')) % note max = probabilité d'avoir
les bons params
29:    [yt,xt]=find(espace'==m(a,b));yy(a,b)=yt(1);xx(a,b)=xt(1);
30:    title([num2str(a),' ',num2str(b),' ',num2str(m(a,b))])
31:  end
32: end
```

Conclusion

Le traitement d'images n'est pas une fin en soi, mais répond à des besoins : nous avons illustré une méthode classique d'identification de structures géométriques dans des images – la transformée de Hough – dans trois cas concrets d'utilisation. Dans un premier temps, nous avons recherché les paramètres représentant les droites visibles dans une image, et validé le concept sur l'identification de bandes signalétiques sur route. Une deuxième application porte sur l'identification de cercles représentant des images de calottes sphériques, et ce en vue du calcul d'angle de contact de gouttes déposées sur des surfaces. Le cas particulier des surfaces super-hydrophobes illustre le cas des cercles observés lorsque la goutte, repoussée par la surface, reste sphérique. Finalement, le dernier exemple porte sur l'identification de coniques, et en particulier d'hyperboles telles qu'observées dans les images RADAR où les antennes se déplacent par rapport à une cible fixe. La méthode de Hough se généralise à toute courbe paramétrable, avec un calcul d'autant plus long que le nombre de paramètres est élevé. L'étape préliminaire d'identification des points remarquables – par exemple par détection de contours ici illustrée par de simples gradients dans les deux axes de la photographie – permet de considérablement réduire le temps de calcul. Le lecteur est encouragé à étendre ces concepts à ses propres problèmes, et en particulier pour des structures géométriques différentes de celles abordées ici. ■

Remerciements

L'inspiration de la nanostructuration de la lame de verre par la suie de bougie vient d'une présentation, *« Hacking Atoms »*, par Thomas Deits à la conférence OMH2013 (Hollande). Les références bibliographiques qui ne sont pas librement disponibles sur le Web ont été obtenues sur Library Genesis (http://gen.lib.rus.ec). Les ressources – images et programmes – associées à cette prose sont disponibles sur http://jmfriedt.free.fr/hough.

Références

[1] HOUGH P. V. C., *« Method and means for recognizing complex patterns »*, brevet US3069654 (18 déc. 1962)

[2] HART P. E., *« How the Hough Transform Was Invented »*, *IEEE Signal Processing Magazine*, pp.18–22 (nov. 2009)

[3] R.O. DUDA R. O. et HART P. E., *« Use of the Hough Transformation to Detect Lines and Curves in Pictures »*, Comm. of the ACM 15 (1), pp.11–15 (1972)

[4] KUMAR A. M. & SIMON P., *« Review of Lane Detection and Tracking Algorithms in Advanced Driver Assistance System »*, *Int. Journal of Computer Science & Information Technology 7* (4), pp.65–78 (2015).

[5] DE GENNES P.-G., F. Brochart-Wyart, D. Quéré, *« Gouttes, bulles, perles et ondes »*, Belin (2002)

[6] BALL P., *« Made to measure – the materials for the 21st century »* (1997) et en particulier l'étude des fonctionnalisations de surface par angle de contact au chapitre 10

[7] TABELING P., *« Introduction à la microfluidique »*, Belin (2003) et son chapitre 7 sur les effets capillaires

[8] BAIN C. D., TROUGHTON E. B., TAO Y.-T., EVALL J., WHITESIDES G. M. et NUZZO R. G., *« Formation of Monolayer Films by the Spontaneous Assembly of Organic Thiols from Solution onto Gold »*, J. Am. Chem. Soc. 111 (1989), 321–335

[9] BAIN C. D. et WHITESIDES G. M., *« Depth sensitivity of wetting : monolayers of ω -mercapto ethers on gold »*, J. Am. Chem. Soc. 110 (1988), 5897–5898

[10] HOLMES-FARLEY S. R., REAME R. H., McCARTHY T. J., DEUTCH J. & WHITESIDES G. M., *« Acid-Base Behavior of Carboxylic Acid Groups Covalently Attached at the Surface of Polyethylene : The Usefulness of Contact Angle in Following the Ionization of Surface Functionality »*, Langmuir 1, pp.725–740 (1985)

[11] Petit devant la longueur capillaire **[5]** donnée par $\sqrt{\frac{\gamma}{\rho \cdot g}}$ avec γ la tension superficielle, ρ la masse volumique du fluide et **g** la constante de gravité – pour l'eau, γ =70 mN/m et la longueur capillaire est 2,7 mm, donc une goutte de 10µL ou de rayon 2 mm respecte la contrainte de négliger les effets de la gravité sur la forme de la goutte

[12] http://www.conrad.fr/ce/fr/product/191251/Camera-_microscope-_numerique-_USB-_plat-_20-_MPix-_zoom-_x-_65-_et-_x-_250-_Conrad pour 80 euros

[13] ARRÉ A. et K.L. MITTAL K. L. (Ed.), *« Superhydrophobic Surface »*, VSP (2009), ou SENEZ V., THOMY V. et DUFOUR R., *« Nanotechnologies for Synthetic Super Non-Wetting Surface »*, Wiley (2014) et en particulier son chapitre 5 intitulé *« Characterization Techniques for Super Non-wetting Surfaces »* qui met en garde contre les techniques triviales présentées ici

[14] DENG X., MAMMEN L., BUTT H.-J. et VOLLMER D., *« Candle soot as a template for a transparent robust superamphiphobic coating »*, Science 335(6064), pp. 67–70 (2012)

[15] GOLOVKO M. M. et POCHANIN G. P., *« Automatic Measurement of Ground Permittivity and Automatic Detection of Object with GPR Images Containing a Response from a Local Object »*, dans *« Ultrawideband Radar : Applications and Design »* édité par J.D. Taylor, et en particulier le paragraphe 7.3 (p.234) titré *« Use of the Hough Transform for Detection of Hyperbolic Curves »*

[16] Le nombre de pixels vérifiant une distribution le long d'une forme géométrique d'équation donnée, valeur sommée dans chaque pixel de l'espace de Hough

GESTION DES ACCESS CONTROL LIST DE VOS RÉSEAUX

Cyrille COLIN [Direction du numérique, Université de Lorraine]

La lecture d'un fichier d'ACL peut vite devenir fastidieuse, et même si ce fichier est bien commenté, il est difficile, voire impossible, de savoir si une règle existe déjà ou si un ajout ne risque pas de bloquer un service important.

Mots-clés : capirca, acl, allow, deny, permit, iptables, cisco, juniper, git

Résumé

Et si nous gérions nos ACLs différemment ? En séparant la définition des réseaux et des services, et les autorisations. C'est ce que propose Capirca, un outil de génération d'ACLs initié par Google, qui supporte plusieurs plateformes comme Juniper, Cisco et Iptables.

1 Les ACLs

Les *Access Control List* (ACL) réseau sont un ensemble de règles, qui interprétées une à une, permettent d'autoriser ou d'interdire les accès à des adresses, ports, etc.

Sous Linux, nous connaissons le couple **Netfilter**/Iptables avec des règles de ce style :

```
iptables -I INPUT -p tcp --destination-port 22 -j ACCEPT
```

Celle-ci permet d'autoriser une connexion entrante sur le port **22** (ssh). Au niveau des options, on a **-I** pour **--insert** qui insère la règle dans la chaîne **INPUT**, **-p** pour **--protocol** pour filtrer sur le protocole utilisé **tcp**, **udp**, **icmp**... ou **all**, ou sa représentation numérique et enfin **-j** pour **--jump** pour indiquer la cible, ici **ACCEPT** pour accepter la connexion.

Nous retrouvons les ACLs sur tout ou partie de nos matériels réseaux, aussi la règle précédente aura sur un routeur Cisco la forme :

```
permit tcp any any eq 22
```

Ou en précisant la machine cible :

```
permit tcp any host 172.24.100.100 eq 22
```

Et pour Juniper :

```
term accept-ssh {
  from {
    destination-address {
      172.24.100.100/32;
    }
    protocol tcp;
    destination-port 22;
  }
  then {
    accept;
  }
}
```

1.1 Problématique

À ma première lecture d'ACL sur un switch Cisco, je me suis retrouvé face à ça :

```
...
permit tcp 172.24.100.0 0.0.0.127 host
172.30.110.154 eq www
permit tcp 172.24.100.0 0.0.0.127 host
172.30.110.154 eq 443
permit tcp 172.24.100.0 0.0.0.127 host
172.30.110.154 eq ftp
permit tcp 172.24.100.0 0.0.0.127 host
172.30.110.154 gt 950
...
```

Le fichier, très long, est volontairement tronqué, mais hormis le fait de ne pas comprendre une seule ligne de celui-ci il m'était difficile d'appréhender :

- les réseaux autorisés ;
- les machines concernées ;
- les services ouverts sur une machine.

Ensuite se pose rapidement la question de comment rajouter une règle et est-ce que celle-ci n'est pas déjà « présente » dans le fichier ? Comment ajouter un ensemble de services à un ensemble de machines ? etc.

1.2 Prémices de solutions

Dans l'idée, il faudrait pouvoir décrire une règle de façon plus générique en indiquant : les machines concernées et les services, ainsi que l'autorisation associée comme :

```
SERVEUR_WEB SERVICES_WEB ACCEPT
" accept-webservices"
```

Nous aurions défini :

```
SERVEUR_WEB = 172.24.100.100/32
HTTP = 80
HTTPS = 443
HHTPALT = 8080
SERVICES_WEB = HTTP HTTPS HTTPALT
```

Pour nous permettre de générer des ACLs Cisco, nous aurions :

```
remark accept-webservices
 permit tcp any host 172.24.100.100 eq 80
 permit tcp any host 172.24.100.100 eq 443
 permit tcp any host 172.24.100.100 eq 8080
```

Ou pour iptables :

```
# accept-webservices
-A FORWARD -p tcp -m multiport --dports
80,443,8080 -d 172.24.100.100/32 -m
state --state NEW,ESTABLISHED,RELATED
-j ACCEPT
```

L'avantage d'une solution de ce type est son « évolutivité », si nous rajoutons un serveur web dans notre infrastructure, nous pourrons modifier la règle en :

```
SERVEURS_WEB SERVICES_WEB ACCEPT
" accept-webservices"
```

Avec les définitions suivantes :

```
SERVEUR_WEB_1 = 172.24.100.100/32
SERVEUR_WEB_2 = 172.24.100.111/32
SERVEURS_WEB = SERVEUR_WEB_1 SERVEUR_WEB_2
```

Nous obtiendrons les règles suivantes :

```
remark accept-webservices
 permit tcp any host 172.24.100.100 eq 80
 permit tcp any host 172.24.100.100 eq 443
 permit tcp any host 172.24.100.100 eq 8080
 permit tcp any host 172.24.100.111 eq 80
 permit tcp any host 172.24.100.111 eq 443
 permit tcp any host 172.24.100.111 eq 8080
```

À partir de là, il est facile de comprendre que l'ajout d'un autre serveur ou l'autorisation d'un autre service n'en sera que beaucoup plus simple !

1.3 Capirca, générateur d'ACL multiplateformes

Nos amis (ou pas) de chez Google ont libéré un outil développé en interne pour la gestion de leurs ACLs et c'est l'outil que nous allons étudier. L'avantage de cet outil est qu'il sait gérer de multiples plateformes dont notamment Juniper, Cisco, Iptables... sachant qu'il est possible d'étendre ces règles à de nouveaux matériels (ce qui est déjà le cas dans des versions de développement).

Cependant rien n'empêche de l'utiliser pour un seul type de matériel en partant du principe qu'il s'agit ici d'améliorer la gestion des filtres et leur maintenance et que, malgré tout, si un changement de technologie était nécessaire, il serait possible de générer les ACLs dans un autre format, sous réserve de l'utilisation de mots-clés génériques (versus mots-clés spécifiques).

Ce logiciel, ainsi que les bibliothèques attenantes, sont écrites en **Python** et sous licence Apache v2.0.

Le principe de Capirca est de proposer un langage de description de règles à partir duquel seront générées les règles pour tel ou tel matériel.

Le fichier de « policy » pour l'exemple précédent serait **webservers.pol** :

```
header {
  comment: "Règles pour les serveurs web"
  target: iptables FORWARD DROP
  target: cisco service-web-in extended
}
term accept-webservices {
    destination-address: SERVEURS_WEB
    destination-port: WEB_SERVICES
    protocol: tcp
    action: accept
}
term default-deny {
  comment: "Rejeter le reste"
  action: deny
}
```

Nous verrons plus en détail la syntaxe des fichiers de « policy ».

2 Installation de Capirca

Rentrons dans le vif du sujet, l'installation, et commençons par le téléchargement des sources disponibles sur **GitHub [1]** :

```
$ git clone https://github.com/google/
capirca.git
$ cd capirca
$ ./setup.py build
$ sudo ./setup.py install
```

Ceci installera le tout dans **/usr/local/lib/python2.7/dist-packages/** (sur une Ubuntu 14.04 LTS). Pour un accès plus rapide, nous créons un petit raccourci sur le script Python :

```
$ sudo vi /usr/local/bin/aclgen
```

Nous inscrivons dans ce fichier :

```
#!/bin/bash
/usr/bin/python /usr/local/lib/
python2.7/dist-packages/aclgen.py
```

Puis nous ajoutons les droits en exécution sur le script :

```
$ sudo chmod +x /usr/local/bin/aclgen
```

Avant de nous lancer dans la configuration de notre installation, arrêtons-nous sur les sources. Testons dans un premier temps le raccourci sur **/usr/bin/aclgen** :

```
$ aclgen
```

Cette commande devrait produire :

```
writing ./filters/sample_cisco_lab.acl
writing ./filters/sample_gce.gce
writing ./filters/sample_ipset
...
writing ./filters/sample_speedway.ipt
writing ./filters/sample_speedway.ipt
writing ./filters/sample_speedway.ipt
writing ./filters/sample_srx.srx
22 filters rendered
```

En effet, les sources sont fournies avec quelques exemples qu'il sera intéressant de regarder.

2.1 Fichiers et structure de fichiers

Pour son fonctionnement, Capirca se sert de 3 types de fichiers :

- les **.net** pour définir les ressources réseaux (réseaux, machines, ensemble de réseaux, ensemble de machines) ;
- les **.svc** pour définir les services et les ensembles de services ;
- les **.pol** pour définir les règles ;
- et les **.inc** pour les fichiers de *policy* destinés à être inclus dans un fichier de règles.

La structure de fichiers proposée par défaut est la suivante :

```
--- def/
  NETWORK.net
  SERVICES.svc
--- policies/
  --- include/
    untrusted-networks-blocking.inc
  sample_cisco_lab.pol
  sample_juniper_loopback.pol
  ...
--- filters/
```

Suite à l'exécution précédente, un ensemble de fichiers a été créé dans le répertoire **filters**.

3 Fonctionnement

3.1 Description des services

Les services sont décrits dans un fichier **.svc**. Vous pouvez créer plusieurs fichiers, cependant, il faudra veiller à ne pas dupliquer les clés, auquel cas une erreur de duplication sera levée.

La syntaxe est simple : une étiquette (clé) suivie du signe **=**, la description du service, éventuellement un commentaire derrière un caractère dièse :

```
OPENVPN = 1194/tcp # OpenVPN over tcp
          1194/udp # OpenVPN over udp
```

Il est possible de grouper les services :

```
BOOTPS = 67/udp   # BOOTP server
BOOTPC = 68/udp   # BOOTP client
DHCP = BOOTPS
       BOOTPC
```

Et de spécifier des plages :

```
HIGH_PORTS = 1024-65535/tcp 1024-65535/udp
```

3.2 Description des réseaux

Les réseaux sont décrits dans un fichier **.net**, comme pour les services. Vous pouvez, et c'est même préférable, créer plusieurs fichiers : **network.net**, **servers.net**, **hosts.net**...

La syntaxe reste la même, une étiquette (clé) suivie du signe **=** et de la ou des plages réseaux. Il est possible de grouper les définitions :

```
EDUSPOT_NETWORK = 10.13.0.0/16 # default ssid
EDUROAM_NETWORK = 10.14.0.0/16 # eduroam ssid
WIFI_NETWORKS =  EDUSPOT_NETWORK
                 EDUROAM_NETWORK
```

Pour une machine, nous utiliserons un masque de **/32** :

```
DNS_SERVER = 172.24.100.5/32 # serveur dns
```

Vous pouvez mixer adresse IPv4 et IPv6 :

```
M2M_NETWORK = 172.23.110.0/24 # Ipv4 M2M
      2a01:e35:8af3:/48 # Ipv6 M2M
```

3.3 Les ACLs

Les ACLs sont décrites dans un fichier de *policy* **.pol**, celui-ci peut contenir un ou plusieurs filtres composés de conditions (*terms*).

Les filtres sont identifiés par un en-tête (*header*) dans lequel on désigne la cible (*target*) de génération comme Cisco, Iptables, etc. Il est éventuellement associé à un champ de description (*comment*). Nous utilisons un filtre pour une direction donnée comme *target* : **iptables INPUT**. Un filtre peut avoir plusieurs cibles, cependant comme nous le verrons ultérieurement, certains mots-clés sont spécifiques à certaines cibles et ne peuvent convenir à tous les usages.

Un en-tête de filtre pourrait ressembler à ça :

```
header {
  comment: "Exemple d'entête"
  target: iptables FORWARD
  target: speedway FORWARD
  target: juniper servers-in
  target: cisco servers-in
}
```

3.3.1 Les targets

Pour le moment, les cibles sont Iptables, **Speedway**, Cisco, et Juniper. D'autres cibles existent, mais sont encore en développement : **Junipersrx**, **PF** (Packetfilter), etc. En fonction de chaque cible, il y a des paramètres spécifiques.

Par défaut, la génération se fait pour les adresses IPv4 et IPv6. Voici la syntaxe pour les différentes cibles :

- **Iptables et Speedway**

Ces deux cibles sont destinées à Iptables.

Speedway utilise une syntaxe équivalente à la sortie de la commande **iptables-save**, qui permet l'utilisation de **iptables-restore**.

```
target: [iptables|speedway]
[INPUT|OUTPUT|FORWARD|custom]
{ACCEPT|DROP} {truncatenames} {nostate}
{inet|inet6}
```

- **Cisco**

```
target: cisco [filter name]
{extended|standard|object-
group|inet6|mixed}
```

- **Juniper**

```
target: juniper [filter name]
{inet|inet6|bridge}
```

3.3.2 Les conditions (*term*)

Chaque filtre contient des conditions définies par le mot **term** suivi de son nom et entre accolades les critères de la condition :

```
term accept-webservices {
    destination-address: SERVEURS_WEB
    destination-port: WEB_SERVICES
    protocol: tcp
    action: accept
}
```

Les critères de conditions sont composés d'un mot-clé et d'une ou plusieurs étiquettes. Il existe deux types de mots-clés : ceux qui sont supportés par tous les générateurs et ceux qui sont spécifiques à un ou plusieurs générateurs.

Il convient de faire attention au fait que certains générateurs vont simplement ignorer des mots-clés non supportés sans message de notification. Généralement, il est préférable pour les règles à cibles multiples de ne pas utiliser les mots-clés spécifiques.

4 L'outil definate

Pour nous aider à générer les fichiers de définition des réseaux, il existe l'outil **definate**. Il propose de générer, à partir d'un fichier de description, une liste de réseaux/machines. Pour l'instant, seule la résolution DNS est proposée, mais definate se veut un *framework* capable d'exécuter plusieurs générateurs afin d'aller chercher les informations d'un réseau. Ainsi, nous pouvons imaginer que rapidement, il sera possible de récupérer des informations via SNMP sur la configuration de nos réseaux.

5 Configuration de travail

Il y a deux façons, voire trois, de construire les ACLs, la troisième étant un mix des deux premières.

On peut construire des ACLs orientées services :

```
term accept-webservices {
    destination-address:  WEB_SERVERS
STARGATE_SERVER ANOTHER_SERVER
    destination-port: WEB_SERVICES
    protocol: tcp
    action: accept
}
```

Ou orientées réseaux/machines :

```
term accept-stargate-server {
    destination-address:  STARGATE_SERVER
    destination-port: WEB_SERVICES SSH
LDAP RADIUS
    protocol: tcp
    action: accept
}
```

Il n'y a pas vraiment de règles pour construire la logique de rédaction des règles, celles-ci dépendant en grande partie du nombre de machines, de la diversité et de la répartition des services, mais aussi des technologies réseau que vous utilisez : VLAN, VRF, etc.

6 Test des ACLs

Comment garantir que l'injection d'une ACL ne va pas interrompre un service essentiel ? Pour cela, Capirca offre la possibilité de tester un fichier d'ACLs et de l'automatiser grâce à une librairie : **AclCheck**.

Cela permettra de vérifier les continuités de service, mais aussi de savoir comment des accès réseau seront traités par les filtres.

Pour un test rapide, il existe un outil en ligne de commandes, **aclcheck_cmdline.py**, qui accepte les options suivantes :

- **--definitions-directory** : répertoire contenant les définitions (par défaut **./def**) ;
- **-p**, **--policy-file** : le fichier de *policy* (par défaut **./policies/sample.pol**) ;
- **-d**, **--destination** : adresse IP de destination (par défaut **200.1.1.1**) ;
- **-s**,**--source** : adresse IP source (par défaut **11.1.1.1**) ;
- **--protocol** : protocole (par défaut **tcp**) ;
- **--destination-port** : port de destination (par défaut **80**) ;
- **--source-port** : port source (par défaut **1025**).

Voici un exemple d'utilisation :

```
# ./aclcheck.py --source-port 5222
--destination-port 80 -s 78.241.212.58
-d  195.10.72.0/24 --protocol tcp -p
./policies/web_servers.pol
```

7 En pratique

7.1 Gestion des modifications et versionning git

Dans notre répertoire de travail, nous créons un dépôt git, enlevons les « filters », qui sont des fichiers générés et le précisons dans le fichier **.gitignore** :

```
# rm -r filters
# git init
# cat 'filters' >> .gitignore
# git add .
# git commit -a -m 'Premier import acls
sous capirca'
```

7.2 Travail collaboratif

Pour rejoindre la nécessité d'effectuer la gestion de version, il sera intéressant de gérer les demandes et que leurs résolutions soient couplées au système de gestion de version.

L'idée est donc d'utiliser **Redmine**/**Gitolite** et de créer un projet « gestion réseau » et un groupe « gestionnaire réseau » (pour les demandeurs) qui aura les droits de rapporteur afin de pouvoir effectuer les demandes. Pour ma part, j'enlève le droit de « visualiser les sources » aux rapporteurs et pour ce type de projet, je n'active que les modules « suivi de demandes », « wiki » et « dépôt de sources ». Il est inutile de surcharger l'interface.

Les avantages d'utiliser un outil comme Redmine sont multiples. D'une part, les demandes sont centralisées, ce qui permet de travailler à plusieurs. Les demandes peuvent être documentées, classées et leurs résolutions automatiquement notifiées aux demandeurs. Le wiki permet de rappeler rapidement les bons usages en termes de demandes d'ACL, de définir le vocabulaire spécifique, de mettre à disposition les plans et autres informations utiles.

7.2.1 Modification d'une ACL

Suite à une demande d'un gestionnaire de réseau, vous recevez un ticket **#127** pour l'ouverture du ssh sur la machine « stargate » à l'adresse **172.24.100.13**.

Rapatriement du projet sur sa machine :

```
# git clone git@mon_serveur:gestion-reseau.git
```

gestion-reseau est le nom du dépôt créé depuis Redmine.

Si vous avez déjà le projet, vous pouvez simplement le synchroniser :

```
# git pull
```

Il faut ensuite modifier les ACLs. Dans les définitions de réseaux/machines doit exister une entrée du type :

```
STARGATE_SERVER = 172.24.100.13/32
```

Dans les définitions de service, SSH doit déjà exister. Aussi, il ne reste plus qu'à ajouter dans le fichier de *policies* concerné la condition suivante :

```
term accept-stargate {
    destination-address: STARGATE_SERVER
    destination-port: SSH
    protocol: tcp
    action: accept
}
```

Puis de générer les ACLs :

```
# aclgen
```

Vous pouvez vérifier le résultat dans **filters** :

```
# cat filters/mypolicy.acl    //pour cisco
...
remark accept-stargate
 permit 6 any host 172.24.100.13 eq 22
...
```

Ensuite, il faut enregistrer les changements (*commit*) en spécifiant que ce *commit* résout la demande **127** avec le mot-clé « fixes » (un des mots-clés par défaut), ce que nous ferons dans le commentaire :

```
# git commit -a -m "ajout ssh sur
stargate fixes #127"
```

Il faudrait ici activer les ACLs générées sur les machines ou les routeurs pour lesquels elles sont destinées. Ensuite nous poussons (*push*) les modifications sur le serveur :

```
# git push
 Counting objects: 11, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 605 bytes, done.
Total 6 (delta 3), reused 0 (delta 0)
remote:
remote: Notifying Redmine (http://dev.
monserveur.fr) about changes to this repo
(gestion-reseau => gestion-reseau)
remote:
remote: Hitting the Redmine Gitolite hook
for ab477a73718f63640f90d68f061661fc9d750695
97bda72bf901e917cfde99a010824a4f62544d74 refs/
heads/master
remote: Response: OK
remote:
remote: Hitting the Redmine fetch changesets URL
remote:
To git@dev.monserveur.fr:gestion-reseau.git
   ab477a7..97bda72  master -> master
```

Pendant le *push*, nous pouvons voir le déclenchement du *hook* de Gitolite qui notifie Redmine d'une modification sur le *repository* **gestion-reseau**, ce qui permettra de modifier le statut de la demande **#127** en « résolu » (suivant le paramétrage de Redmine) et donc de notifier les demandeurs.

Conclusion

Capirca propose une autre façon d'envisager la gestion des ACLs, « plus logique » et découpée, basée sur les réseaux et les services. Sa mise en œuvre n'est pas forcément aisée, car elle remet en cause nombre de pratiques, mais son utilisation sur de grandes infrastructures, gérées à plusieurs, permet à terme un meilleur suivi et une diminution des risques. ■

Référence

[1] GitHub de Capirca : https://github.com/google/capirca

LES SECRETS DE FABRICATION D'UNE ENTRÉE GAGNANTE DE L'IOCCC

Etienne DUBLÉ [Ingénieur de Recherche CNRS au LIG]
Avis et relecture : T. BRAUD, P. BRUNISHOLZ, H.-J. AUDÉOUD, T. CLAEYS, E. MORIN, C. CHARGY

Vous rappelez-vous des articles « Le coin du vieux barbu » où David Odin analysait les entrées gagnantes de l'IOCCC ? Je vous propose de passer de l'autre côté du miroir, afin de découvrir la recette que j'ai employée pour cuisiner mon entrée gagnante de l'IOCCC 2015.

Mots-clés : IOCCC, Obfuscation, langage C, Python

Résumé

L'IOCCC (*International Obfuscated C Code Contest*) est un concours où les candidats proposent un code C le plus indéchiffrable possible. Ma proposition pour l'IOCCC 2015 ayant été retenue parmi les quatorze gagnantes [1], j'expose dans cet article quelques techniques et astuces employées à cette occasion.

L'histoire de l'IOCCC

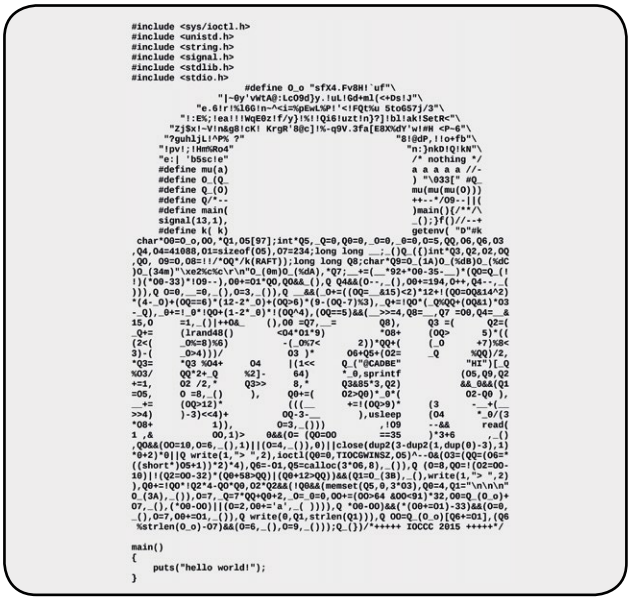
En 1984, L.C. Noll et L. Bassel travaillaient sur le portage des programmes **bash** et **finger**. Ce code était apparemment très peu lisible. L.C. Noll envoya un message sur le forum *net.lang.c* en invitant les abonnés à faire le même exercice. Il fut surpris du nombre de réponses... Il proposa alors aux volontaires d'écrire leur propre code obscurci, ce qui lança la première édition du concours **[2]**.

Plus de 30 ans après sa création (voir encadré), le concours de l'IOCCC existe toujours. En ce qui me concerne, j'en ai entendu parler la première fois en 2003, en regardant le site de Fabrice Bellard (auteur de **qemu**, entre autres). Il a gagné le concours en 2000 (avec une méthode de calcul rapide de nombres premiers) et en 2001 (en écrivant un compilateur C de seulement 3301 octets...). Plus tard, j'ai beaucoup aimé les articles de *GNU/Linux Magazine* où David Odin tentait de déchiffrer les entrées gagnantes. Et finalement, l'année dernière j'ai décidé de me lancer... La figure 1 présente le code que j'ai proposé. Dans la suite vous découvrirez le cheminement que j'ai suivi et nous détaillerons certaines parties de ce code.

1 Les règles et conseils des juges

Les candidats sont invités à suivre un certain nombre de règles et de conseils.

Les conseils **[3]** sont bien sûr indicatifs, mais au final très utiles. Par exemple, on y apprend que certains thèmes (calcul de pi, etc.) ont été traités souvent, et les juges sont donc très exigeants sur ces sujets.



```
#include <sys/ioctl.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>
                #define O_o "sfX4.Fv8H!`uf"\
              "|~0y'vWtA@:LcO9d]y.!uL!Gd+ml{<+Ds!J"\
           "e.6!r!%1GG!n~^<i=%pEwL%P!'<!fQt%u St<6S7j/3"\
         "!:E%;!ea!!!WqE0z!f/y}!%!!Q1&!uzt!n}?]!bl!ak!SetR<"\
       "Zj$x!~Vn&g8!cK! KrgR'8@c]!%-q9V.3fa[E8X%dY'w!#W <P~6"\
     "?guhljL!^P% ?"                                  "8!@dP,!lo+fb"\
    "!pv!;!HmM#o4"                                     "n:}nkb!Q!kN"\
    "e:| 'b5sc!e"                                      /* nothing */
    #define mu(a)                                      a a a a a //-
    #define O_(Q_                                      ) "\033[" #Q_
    #define Q_(O)                                      mu(mu(mu(O)))
    #define Q/*--                                      ++--*/O9--||(
    #define main(                                      )main(){/*^\
    signal(13,1),                                      _(};}f()//--+
    #define k( k)                                      getenv( "D"#k
 char*O0=O_o,O0, *Q1,O5[97];int*Q5,_Q=0,Q0=0,_O=0,_0=0,O=5,QQ,O6,Q6,Q3
,Q4,O4=41088,O1=sizeof(O5),O7=234;long long __;_(){Q_({}int*Q3,Q2,O2,OQ
,QO, O9=O,O8=!!/*OQ*/k(8&AFT);}long long Q8;char*Q9=O_(1A)O_(%d8)O_(%dC
)O_(34m)"\xe2%c%c\r\n"O_(0m)O_(%dA),*Q7;__+=(__*92+*O0-35-__)*(QO=Q_(!
!){*O0-33)*!O9--),O0+=O1*QO,QO&&_(),Q_Q4&&(O-+,_(),O0+=194,O++,Q4--,_(
)))),Q O=0,__=0,_(),O=3,_())),Q __&&(_O+=!(QQ=__&15)<2)*12+!(QO=OQ&14^2)
*(4-_O)+(OQ==6)*(12-2*_O)+(OQ>6)*(9-(OQ-7)%3),_Q+=!QO*(_QVQQ+(OQ&1)*03
-_Q),_0+=!_0*!QO+(1-2*_0)*!(OQ^4),(OQ==5)&&(__>>=4,Q8=__,Q7 =00,Q4=__&
15,O     =1,_()|++O&__    (),O0 =Q7,__=          Q8),     Q3 =(     Q2=(
_Q+=     (1rand48()       <04*Q1*9)              *O8+     (OQ>     5)*((
(2<(     _O%=8)%6)              -(_O%7<       2))*QQ+(     (_O      +7)%8<
3)-(     _O>4)))/               O3 )*     O6+Q5+(O2=     _Q       %QQ)/2,
*Q3=     *Q3 %O4+      O4       |(1<<     Q_("@CADBE"              "HI")[_Q
%O3/     QQ*2+_Q       %2]-     64)      *_0,sprintf              (O5,Q9,Q2
+=1,     O2 /2,*       Q3>>     8,*      Q3&85*3,Q2)              &&_0&&(Q1
=O5,     O =8,_()      ),       Q0+=(    O2>Q0)*_0*(              O2-Q0 ),
__+=     (OQ>12)*               (((__      +=!(OQ>9)*     (3      -__+(
>>4)     )-3)<<4)+              OQ-3-__          ),usleep     (O4     *_0/(3
*O8+           1)),             O=3,-(_()))          ,!O9     --&&    read(
1 ,&           O0,1)>     0&&(O= (QO=OO          ==3S     )*3+6    ,-(_()
,QO&&(OO=10,O=5,_(),1)||(O=4,_(),0)||close(dup2(1,dup(0)-3),1)
*O+2)*0||Q write(1,"> ",2),ioctl(Q0=0,TIOCGWINSZ,O5)^--O&(O3=((QQ=(O6=*
((short*)O5+1))*2)*4),Q6=-O1,Q5=calloc(3*O6,8),_()),Q (O=8,QO=!(O2=OO-
10)||(Q2=OO-32)*(QO+5B>QQ)|(Q0+12>QQ))&&(Q1=O_(3B),_(),write(1,"> ",2)
),O0+=!QO*!Q2*4-QO*Q0,Q2*Q2&&(!Q0&&(memset(Q5,0,3*O3),Q6=4,Q1="\n\n\n"
O_(3A),_()),O=7,_Q="7"QQ+Q0+2,_O=_0=0,OO+=(OO>64 &OO<91)*32,O0=Q_(O_o)+
O7,_(),(*O0-OO)||(O=2,O0+='a',_( ))),Q *O0-O0)&&(*(O0+=O1)-33)&&(O=0,
_(),O=7,O0+=O1,_()),Q write(0,Q1,strlen(Q1))),Q O0=Q_(O_o)[Q6+=O1],(Q6
%strlen(O_o)-O7)&&(O=6,_(),O=9,_()));Q_[)]/*++++* IOCCC 2015 ++++*/

main()
{
    puts("hello world!");
}
```

Fig. 1 : Le code du programme finalisé.

Les règles **[4]** doivent en revanche être respectées à la lettre. La règle la plus contraignante est la limite de taille du code source : il ne doit pas dépasser **4096** octets. De plus, en testant avec l'outil **iocccsize.c**, la taille indiquée doit être inférieure ou égale à **2053**. Cet outil renvoie une taille plus faible parce que les mots -clés du langage ne comptent que pour **1** octet, les espaces sont ignorés, etc.

2 À la recherche d'une idée gagnante...

Ensuite, il vous faut une idée sympa. À quoi va servir votre programme ? Cela peut paraître secondaire, le but étant l'obfuscation, mais si votre proposition ne surprend pas les juges, vos chances sont minces. Bien sûr, si vous avez une idée vraiment géniale et inédite en matière d'obfuscation, alors vous pourrez gagner même si le programme en lui-même est très basique. On en a eu quelques beaux exemples en 2015 **[5]**. Dans le même ordre d'idée, certains gagnent avec des *1-liners*... Donc tout est possible.

Mais revenons au cas le plus fréquent. Quelle est donc la dernière fois qu'un collègue vous a montré une technique ou un outil épatant ? Ou bien, en lisant *GNU/Linux Magazine* (par exemple), n'avez vous pas eu envie de creuser un thème en particulier ? Si ça ne vous dit rien, vous trouverez peut-être l'inspiration en regardant les entrées gagnantes des années précédentes. On discerne assez bien quelques sujets dont les juges paraissent friands : les maths, l'informatique théorique, les jeux vidéo, etc. Si vous hésitez entre plusieurs idées, sachez que vous pouvez soumettre jusqu'à 8 propositions différentes ! Un auteur peut donc gagner plusieurs fois la même année (cela a été le cas en 2015 par exemple).

Concernant l'objet de cet article, c'est le numéro de mai 2015 de *GNU/Linux Magazine* qui m'a inspiré. F. Le Roy nous y présentait **blessed-contrib**, un outil pour afficher des *dashboards* avec de vraies courbes dans le terminal **[6]**. L'outil repose sur la plage des **256** caractères « Braille ». Grâce à eux, on peut considérer chaque caractère du terminal comme une mini-bitmap de taille **2x4**, la plage de **256** caractères donnant ainsi toutes les combinaisons possibles pour allumer ou éteindre chacun des **8** points.

En fait, on peut aller plus loin dans le côté visuel : si on remplace un caractère par un autre qui contient un point supplémentaire, on fait « apparaître » ce point supplémentaire. Avec une boucle et une temporisation, on peut donc visualiser un tracé de courbe progressif, pour simuler, par exemple, quelqu'un qui écrit à la main. C'est ce que j'ai choisi de faire pour cette entrée de l'IOCCC 2015.

La figure 2 présente l'utilisation du programme en mode interactif. On le voit en train de tracer le mot « hello ».

Fig. 2 : Le programme final lancé en mode interactif.

3 La manière simple d'obtenir un résultat compliqué

Maintenant qu'on a une idée, on peut commencer à travailler et implémenter une première version. En ce qui me concerne, j'ai commencé par un prototype fonctionnel en **Python**. Avec ce langage, on peut faire un prototypage rapide d'abord, puis se rapprocher petit à petit du programme C, les fonctionnalités bas-niveau de la librairie standard étant proches de celles du C.

Attention quand même à ne pas trop repousser le portage en C. Ce sera de toute façon vite nécessaire pour savoir si vous êtes proche ou non des limites de taille imposées.

Notez aussi que seule la partie visible de votre travail devra être en langage C : rien ne vous empêche d'automatiser les tâches annexes dans un autre langage. Par exemple, grâce à un script en Python, j'ai généré une longue chaîne de caractères utilisée dans le code source (la macro `O_o`, nous en reparlerons). J'ai aussi scripté le formatage final en Python, comme nous le verrons plus loin.

4 De l'obfuscation... beaucoup d'obfuscation !

Ensuite, on va commencer à s'amuser. Que peut-on imaginer pour rendre un programme absolument indéchiffrable ? Ma proposition, par exemple, est dotée des sept couches d'obfuscation qui suivent.

4.1 Complexité fonctionnelle

Dans l'idée que j'ai voulu développer, il y a quelques subtilités qui compliquent l'analyse du programme. Ce n'est donc pas à proprement parler une couche d'obfuscation que j'aurais rajoutée, mais ça tombe plutôt bien : quitte à obfusquer, il vaut mieux partir d'une base déjà complexe.

Pour remplacer un caractère par un autre, il faut pouvoir positionner le curseur à sa guise sur le terminal. Cela implique l'utilisation de caractères d'échappement fort peu lisibles **[7]**. Les juges ont déjà rencontré ces caractères plusieurs fois dans le contexte de l'IOCCC, mais je doute qu'ils les aient mémorisés...

Une autre subtilité se situe au niveau de l'encodage des caractères Braille. En réalité, pour écrire les « vrais » caractères Braille il suffit d'une bitmap `2x3` (et non `2x4`). Cependant, dans l'Unicode, on a 2 points supplémentaires en bas. Cela rend la numérotation des points très étrange : les points 1 à 6 sont numérotés de haut en bas en 2 colonnes, et les points 7 et 8 sont comptés en ligne. Le calcul du caractère Braille à afficher est donc forcément bizarre...

4.2 Obfuscation des identifiants et noms de variables

Il s'agit d'une forme d'obfuscation très classique. On choisit des caractères qui se ressemblent, comme le `0`, le `Q` et le zéro par exemple, on réutilise les noms de variables dans des contextes différents, on s'amuse à enfreindre les règles de nommage habituelles (les fonctions en majuscules, les macros en minuscules), etc. Le but est toujours le même : faire passer un identifiant donné pour ce qu'il n'est pas.

Au début de mon code (figure 1 ou en ligne **[8]**), on lit par exemple la définition de macro suivante :

```
19: #define main() main(){ \
20:     signal(13,1), _();}f()
```

Cela ressemble à une macro récursive. À un détail près : les macros récursives cela n'existe pas ! ;) En fait, cette macro est utilisée à la fin du code, dans ce qui ressemble beaucoup à la fonction `main()` :

```
52: main()
53: {
54:     puts("hello world!");
55: }
```

En réalité, après préprocessing, on obtient :

```
main(){ signal(13,1), _();}f()
{
    puts("hello world!");
   }
```

La fonction `main()` n'est donc pas exactement là où on l'attendait. Et la fonction `f()` n'est jamais appelée dans le code... Elle est juste là pour brouiller les pistes.

4.3 Obfuscation par le préprocesseur

Cette obfuscation n'est généralement pas la plus robuste face à l'expertise des juges. Mais cela peut être un plus, surtout si vous implémentez quelques contre-mesures pour perturber l'exploration des juges. Analysons quelques éléments de mon programme à ce sujet :

```
    ...
18: #define Q     O9--||(
    ...
27: ... Q O=0,__=0,_(),O=3,_()) ...
    ...
```

La définition de la macro `Q` en ligne 18 ouvre une parenthèse sans la refermer. De ce fait, à chaque fois que cette macro est utilisée, il faut fermer la parenthèse, comme en ligne 27, et au final le code se trouve avec plus de parenthèses fermantes que de parenthèses ouvrantes. Les « enjoliveurs de code », que les juges utilisent sûrement, n'aiment pas ce genre de choses :

```
$ uncrustify [...] -f prog.c > prog_
uncrustify.c
[...]
prog.c:23 Unmatched PAREN_OPEN
[...]
```

Mais attention, les juges peuvent utiliser l'option `-E` de `gcc` pour voir l'allure du code après remplacement des macros. Effectivement, dans la sortie de `gcc -E`,

le parenthésage est rétabli, rendant cette obfuscation plutôt vaine. Il faudrait donc décourager les juges d'utiliser `gcc -E`... Voilà comment je m'y suis pris.

```
   ...
 7: #define O_o   "sfX4.Fv8H!`uf"\
                  "|~Øy'vWtA@[...]"\
   ...            [...]
14:               "e:| 'b5sc!e"
   ...
15: #define mu(a) a a a a a
   ...
17: #define Q_(0) mu(mu(mu(0)))
   ...
23: ... Q_({) ...
   ...
47: ... Q_(O_o) ...
   ...
50: ... Q_(}) ...
   ...
```

La macro `Q_` permet de répéter $5^3 = 125$ fois son argument. Je l'utilise pour répéter les accolades ouvrantes et fermantes, ainsi que la macro `O_o`. La répétition des accolades permet de générer un grand nombre de blocs de codes imbriqués (inutiles). Imaginez l'indentation générée par un enjoliveur de code après ça... Quant à la macro `O_o`, il s'agit d'une longue chaîne de caractères (`252` octets, elle est définie entre les lignes 7 et 14), la répéter 125 fois permet donc de rendre le résultat de `gcc -E` assez lourd...

Mais bon. Les juges auront tôt fait de redéfinir cette macro `Q_` pour passer de 125 répétitions à une seule :

```
#define Q_(0) 0
```

En tout cas, ils peuvent essayer. Car dans ce cas, le programme ne fonctionne plus :). En effet, pour parcourir `O_o`, le pointeur est incrémenté de `97` en `97`, et non pas de `1` en `1`, on a donc tôt fait de dépasser sa taille.

Prenons l'exemple d'un tableau de sept cases. Le plus simple est de numéroter les cases successivement : `0`, `1`, `2`, `3`, `4`, `5`, `6`. Mais on peut aussi compliquer un peu la chose en numérotant de `5` en `5`. On applique aussi un modulo `7` pour ne pas dépasser la taille. Cela donne : `0`, `5`, `3`, `1`, `6`, `4`, `2`. On a bien parcouru toutes les cases (car `5` et `7` sont premiers entre eux). Si on numérote de `5` en `5` sans appliquer le modulo (`0`, `5`, `10`, `15`, etc.), alors on doit concaténer cinq instances du tableau initial pour arriver au même résultat sans dépassement.

La macro `O_o` est indexée suivant ce principe. La méthode cachée, pour pouvoir se passer des `125` répétitions, est donc de rajouter des modulos `252` (la taille de `O_o`) à divers endroits du code. Voilà qui devrait quand même occuper les juges un moment, s'ils veulent passer par là...

4.4 Obfuscation liée au compactage

La macro `O_o` encode le tracé de chaque caractère géré par le programme. Il a fallu trouver un encodage compact, sinon on aurait gaspillé une bonne partie des `4096` ou `2053` octets autorisés.

En fait, chaque caractère à tracer correspond à un « circuit » à parcourir. J'ai donc encodé les opérations successives à effectuer (avancer, tourner à gauche, à droite, etc.) sous la forme de `4` bits par opération. En concaténant les opérations, on obtient donc un entier de `4*n` bits qui correspond au circuit à parcourir. Il fallait ensuite un moyen pour encoder ces entiers de manière compacte dans le code source.

Pour compacter des informations en langage C, le plus efficace est d'utiliser une chaîne de caractères. On peut y encoder des valeurs entre `0` et `127` en utilisant le caractère ASCII correspondant. Par exemple `'b'` pour la valeur `98`. Mais certaines valeurs doivent être échappées. Par exemple, la valeur `10` s'écrira `\n`, ce qui fait 2 caractères dans le code source (`backslash` et `n`). Et il y a pire. La valeur `1` par exemple ne correspond pas à un caractère imprimable, on devra donc l'écrire en hexadécimal, avec quatre caractères : `\x01` !!

En observant la table des caractères ASCII **[9]**, on voit que les 95 caractères entre l'espace (`32`) et `~` (`126`) sont imprimables. Dans cette plage, seuls `"` et `\` devront être échappés dans une chaîne C.

Vous souvenez-vous de vos premiers cours d'informatique, quand vous avez appris à compter dans des bases différentes de la base décimale ? Disons que l'entier que nous devons coder a la valeur `125674529`. En base `10`, il se décompose ainsi :

$$125674529=1*10^8+2*10^7+...+2*10^1+9*10^0$$

Sur le même principe, décomposons-le en base `95` (en prenant le reste de divisions successives par `95`) :

$$125674529=1*95^4+51*95^3+55*95^2+14*95^1+74*95^0$$

Ainsi, coder cet entier revient à coder les coefficients successifs `1`, `51`, `55`, `14` et `74`. S'agissant de la base `95`, ces coefficients sont des nombres compris entre `0` et `94`. Ça tombe bien, on a justement repéré une plage de `95` symboles successifs dans la table ASCII. Pour coder `1`, on prend donc le deuxième symbole de cette plage (car on démarre à `0`), et on obtient `'!'`. On procède de même pour les autres. Et au final, on a codé notre nombre en cinq symboles seulement. Notez qu'en base décimale, il fallait neuf symboles !

À l'intérieur de la macro `0_o`, j'ai donc utilisé ce principe pour encoder le circuit correspondant à chaque caractère à tracer. Cela permet à la fois un encodage compact et une couche d'obfuscation supplémentaire.

4.5 Obfuscation dans la structure du programme

À ce stade, notre programme est déjà relativement complexe. Mais on peut faire mieux, en prenant à rebrousse-poil les principes de programmation structurée les plus élémentaires :).

Premièrement, on regroupe tout dans une seule fonction. De ce fait, un appel de `f()` vers `g()` devient un appel récursif à `f()`, ce qui est tout de suite moins clair. Ensuite, on enlève les autres éléments structurants du programme. On remplace les boucles via un nouvel appel récursif. On essaie aussi de dissimuler le branchement conditionnel (le `if`), mais ça, ça reste difficile.

Prenons un exemple de code simple :

```
if (a==2) {
   b = 3;
}
```

On peut utiliser les opérateurs booléens pour cacher ce `if`, mais cela reste facilement lisible :

```
(a==2)&&(b=3);
```

Essayons avec l'opérateur ternaire :

```
b=(a==2)?3:b;
```

On a écrit que `b` ne prend la valeur `3` que lorsque la condition est remplie, sinon il garde sa valeur. La variation suivante nous permet d'avoir une clause `<else>` à `0` :

```
b+=(a==2)?3-b:0;
```

En C, si la condition est vraie, sa valeur est `1`, et `0` sinon. Donc on peut écrire :

```
b+=(a==2)*(3-b);
```

À ce stade, le problème reste l'opérateur `==`. En le voyant, on pense tout de suite à une condition, donc le juge cherchera un `if` dissimulé. Mais on peut le remplacer, lui aussi. En effet, l'expression `(a==2)` est équivalente à `!(a-2)`. On obtient donc (après avoir consulté la table de précédence des opérateurs **[10]**) :

```
b+=!(a-2)*(3-b);
```

Ou mieux, au cas où un œil expert y verrait encore un `if`, on met la condition à la fin :

```
b+=(3-b)*!(a-2);
```

Et voilà, au final il est plutôt bien caché ce `if` non ? On est pourtant parti d'un cas trivial. Imaginez la chose dans un cas complexe et saupoudré de diverses couches d'obfuscation...

Un autre exemple de dissimulation de `if` concerne le code d'initialisation. Un programme plus simple aurait une structure du type :

```
if (!init_done) {
   <init>;
   init_dt = 1;
}
```

En effet, le programme étant regroupé dans une seule fonction récursive, on va souvent passer par là et il faut s'assurer que `<init>` ne sera exécuté qu'une fois au début.

Pour une partie de ce code, j'ai adopté une approche différente : j'ai conçu ce `<init>` pour qu'il puisse être appelé `N` fois sans avoir aucun effet, sauf la première fois. Donc je n'avais plus besoin du `if`.

Notez enfin que comme les expressions sont combinables à volonté, elles sont très pratiques pour l'obfuscation. Dans mon unique fonction, en dehors de la déclaration des variables au début, il n'y a pas de point-virgule : j'ai combiné tout le traitement en une seule et unique expression. Ce n'est pas très compliqué, on peut par exemple combiner 2 expressions en une seule avec la virgule. Mais cela rend la structure du programme encore plus obscure, à mon avis.

Au départ je voulais aller plus loin dans ce registre. Mais, au bout d'un moment, on ne comprend plus son propre programme, et là on est obligé de s'arrêter...

4.6 Obfuscation dans le comportement du programme

Pour comprendre cette partie, revenons un instant sur deux *use cases* du programme :

- **Use case 1** : mode interactif

```
$ ./prog
>
```

Le programme semble suivre le pseudo-code suivant :

```
boucle infinie
    afficher "> "
    lire une ligne de texte entrée par
l'utilisateur
    effectuer le rendu
fin boucle
```

- **Use case 2** : mode non-interactif (entrée standard redirigée)

```
$ ./prog < file.txt
```

ou

```
$ echo hello | ./prog
```

Dans ce cas, il n'y a pas d'interaction avec l'utilisateur. Le pseudo-code semble plutôt être :

```
boucle infinie
    lire les caractères arrivant sur l'entrée standard
    effectuer le rendu
fin boucle
```

En réalité, que ce soit l'utilisateur ou pas qui lui « parle », le programme lit toujours les données sur son entrée standard. Il n'y a donc pas de différence dans l'implémentation à ce niveau. De plus, dans sa configuration standard, le terminal *bufferise* déjà l'entrée clavier ligne par ligne : il n'y a rien de spécial à faire à ce titre pour gérer le cas « interactif ». Au final, la *seule* différence à implémenter dans le code pour différencier ces deux *use cases* est l'affichage ou non de **"> "** !!

Un programme classique utiliserait la fonction **isatty()** pour savoir si **stdin** est un terminal, et donc afficher ou non ces deux caractères. Mais bien sûr, pour un code de l'IOCCC, ce serait trop simple ;). Faisons un petit test avec un programme plus didactique :

```
$ cat test.c && gcc -o test test.c
[... #includes ...]
int main() {
  write(fileno(stdin), "> \n", 3);
}
$ ./test
>
$ ./test < file.txt
$ echo hello | ./test
$
```

Ce programme tente d'écrire **"> \n"** sur son entrée standard (**stdin**). Voilà une idée bizarre, non ? Normalement, quand on écrit, c'est sur la sortie standard (**stdout**)...

Les trois essais qui suivent doivent vous rappeler fortement les *use cases* décrits plus haut. On s'aperçoit que dans le premier cas, l'écriture réussit, et elle échoue dans les deux autres. En réalité, dans le premier cas, **stdin = stdout = <le_terminal>**. Il est donc logique que ça fonctionne. En revanche dans le deuxième cas, **stdin** est le fichier **file.txt** ouvert en lecture, on ne peut donc pas y écrire. Et dans le troisième cas, **stdin** est un pipe ouvert en lecture, donc là aussi l'écriture échoue.

Vous l'aurez deviné, mon programme utilise la même astuce : en écrivant le prompt sur **stdin**, celui-ci n'apparaîtra que dans le cas interactif.

Au final, dans mon code il n'y a donc aucune distinction entre le mode interactif et le mode non-interactif. Le programme exécute *exactement les mêmes instructions* dans ces deux cas. Et l'une de ces instructions, suivant qu'elle échoue ou non, permet de parfaire l'illusion.

4.7 Inversion des descripteurs de fichiers

Lors de son initialisation, mon programme s'amuse à inverser les descripteurs de fichier **0** (d'habitude **stdin**) et **1** (d'habitude **stdout**) via plusieurs appels à **dup()**, **dup2()** et **close()** (voir encadré page suivante).

Après inversion, **0** correspond donc à **stdout** et **1** à **stdin**. Ainsi, le programme lit sur le descripteur **1** (lignes 40-41) et écrit sur **0** (ligne 49). Logique. Cependant, on trouve quand même une écriture sur le descripteur **1** via l'instruction **write(1,"> ",2)** en ligne 45 : ce sont les 2 caractères du prompt interactif, qui sont donc écrits sur **stdin**, conformément à ce qui est dit dans la section précédente.

5 Le formatage

J'ai choisi de formater le code sous la forme d'un cadenas fermé, comme si je défiais les juges de l'« ouvrir ». En réalité, ce formatage est une opération délicate : vous manipulez la version finale de votre code, que vous ne comprenez plus depuis un bon moment, et qui explose à la moindre modification mineure !

J'ai voulu automatiser cette étape, mais il ne me restait que très peu de temps avant la date limite de soumission. J'ai donc été contraint de faire simple. Mon script **layout.py** prend 2 fichiers en paramètre : **compacted.c** qui contient le code à formater, sous la forme d'un bloc de code sans espaces, et **layout.txt** qui définit le masque à appliquer. Voici un exemple de fichier **layout.txt** pour appliquer la forme de la lettre D :

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

L'étoile indique à **layout.py** d'écrire le prochain caractère du code à cet endroit, l'espace et le **\n** doivent juste être reportés en sortie.

Bien sûr, vous aurez des noms de variable ou de fonction coupés au milieu. On essaie alors de permuter des choses

Descripteurs de fichier, dup(), dup2(), késako ??

Un processus identifie chaque fichier ouvert par un entier positif, que l'on nomme « descripteur de fichier ». Ainsi, la valeur de retour de **open()** est un descripteur de fichier, que l'on pourra passer en paramètre aux fonctions classiques de manipulation de fichiers, telles que **read()**, **write()**, etc.

Au démarrage d'un processus, celui-ci hérite déjà de 3 descripteurs de fichiers : **0** pour **stdin** (entrée standard), **1** pour **stdout** (sortie standard), **2** pour **stderr** (sortie erreur). On peut le vérifier en utilisant **fileno(<flux>)**, par exemple **fileno(stdin)** renverra **0**. Un appel réussi à **open()** alloue le plus petit descripteur encore libre. L'appel à la fonction **close()** provoque la libération du descripteur concerné ; celui-ci pourra alors être réutilisé lors d'un prochain **open()**. Le test suivant illustre ces notions :

```
$ cat test.c && gcc -o test test.c
[... #includes ...]
int main() {
  close(1);
  open("/tmp/hop", [...]);
  printf("hello\n");
}
$ ./test
$ cat /tmp/hop
hello
$
```

Ce programme redirige sa propre sortie standard vers un fichier. Il y parvient en fermant son descripteur **1** (**stdout**), puis en ouvrant le fichier **/tmp/hop**. Remarquez que je ne prends pas la peine de vérifier la valeur du descripteur retourné par **open()** : après l'appel à **close(1)**, le plus petit descripteur libre sera forcément **1**. La fonction **printf()**, qui écrit en interne sur le descripteur **1**, se trouve ainsi à écrire sur notre fichier nouvellement ouvert, à la place de **stdout**.

Passons à la fonction **dup()**. Celle-ci permet de dupliquer un descripteur. On se retrouve alors avec deux descripteurs qui pointent vers le même fichier ouvert. La valeur de retour de **dup()** est le nouveau descripteur alloué. Comme pour **open()**, le plus petit descripteur libre est choisi. En combinant avec **close()**, on peut échanger des descripteurs, par exemple **0** et **1** :

```
saved_stdin=dup(0);   // sauvegarde stdin
close(0);             // libere 0
dup(1);               // duplique 1(stdout) vers 0
close(1);             // libere 1
dup(saved_stdin);     // duplique saved_stdin vers 1
```

L'expression que j'utilise pour l'échange est plus élaborée (ligne 42) :

```
close(dup2(3-dup2(1,dup(0)-3),1)*0+2)
```

En effet, elle fait partie du code **<init>** décrit en section 4.5 : on peut l'appeler **N** fois, mais passée la première itération elle nous laissera toujours dans le même état. Par ailleurs, j'utilise aussi la fonction **dup2()**, qui permet d'indiquer le descripteur à allouer, plutôt que de prendre le plus petit disponible.

comme **<e1>+<e2>** en **<e2>+<e1>** dans **compacted.c** pour voir si ça passe. Au pire, on ajoute un espace par-ci par-là. Pour procéder de manière progressive, **layout.py** prend un numéro de ligne en troisième paramètre. De ce fait, il applique le masque uniquement sur les **n** premières lignes, et il concatène le reste du code compact. On fait alors un test complet (test limite de taille, test compile, test fonctionnel) du genre :

```
$ ./layout.py compacted.c layout.txt 1 >
prog.c && \
    ./iocccsize -i < prog.c && gcc -o
prog prog.c && echo hello | ./prog
```

Si tout fonctionne avec **n=1**, on essaie avec **n=2**, et ainsi de suite.

6 Les fichiers associés

Quand on soumet un programme à l'IOCCC, on peut fournir un certain nombre de fichiers associés. Pour ma part, j'ai hésité à fournir un fichier contenant l'encodage du tracé des caractères, plutôt que de l'intégrer au code source (macro **0_o**). Mais les juges pourraient interpréter ceci comme un moyen d'extraire du code pour satisfaire les limites de taille. À vous donc de voir quels fichiers annexes vous fournissez.

On doit aussi fournir un **Makefile**, ainsi qu'un fichier de remarques en **markdown**. Si vous gagnez, les juges commentent également le code et concatènent vos remarques pour générer le fichier **hint.html** publié sur le site **[11]**. Normalement, dans ce fichier, vous expliquez comment utiliser votre programme, en quoi il est obfusqué, et toute autre subtilité que les juges risquent de ne pas détecter. Laissez quand même du mystère... Par exemple, j'ai fait remarquer que le programme fournit deux modes, « interactif » et

« non-interactif », et que pourtant la fonction `isatty()` n'est pas utilisée. Juste de quoi piquer la curiosité des juges en somme. Je les invite aussi à définir la variable d'environnement `DRAFT=1`, ou encore à donner `prog.c` en entrée du programme, pour voir. Je m'aperçois d'ailleurs que je n'ai pas abordé ces petites choses dans l'article... Est-ce que cela va aussi piquer la curiosité de mes lecteurs ? :)

Les juges font également remarquer que la revue des propositions peut devenir un peu fastidieuse quand le soir approche, et qu'ils apprécient donc un fichier de remarques avec un peu d'humour.

Conclusion

Vous voilà parés pour écrire votre première entrée pour l'IOCCC : j'espère bien voir fleurir les mentions FR parmi les gagnants des prochaines éditions ! ■

Références

[1] Les fichiers de mon entrée gagnante : http://www.ioccc.org/years.html#2015_duble

[2] L'appel à proposition de 1984 : https://groups.google.com/d/msg/net.lang.c/lx-TAuEyeRI/HdOOnNx6LC0J

[3] Les conseils des juges : http://www.ioccc.org/2015/guidelines.txt

[4] Les règles du concours : http://www.ioccc.org/2015/rules.txt

[5] Une entrée de 2015 vraiment originale : http://www.ioccc.org/2015/endoh2/hint.html

[6] LE ROY F., *« Créez des dashboards sexy dans vos terminaux »*, *GNU/Linux Magazine n°182*, mai 2015, p.78.

[7] Déplacement du curseur dans un shell : http://www.tldp.org/HOWTO/Bash-Prompt-HOWTO/x361.html

[8] Le code de mon entrée gagnante : http://www.ioccc.org/2015/duble/prog.c

[9] La table des caractères ASCII : http://www.asciitable.com/

[10] La priorité des opérateurs : http://en.cppreference.com/w/c/language/operator_precedence

[11] Le fichier `hint.html` de mon entrée : http://www.ioccc.org/2015/duble/hint.html

PARTEZ À LA DÉCOUVERTE DE LA RÉALITÉ AUGMENTÉE

Thierry GAYET [CTO AMA RENNES]

Après le déferlement de Pokémon GO, GNU/Linux Magazine se devait d'aborder le sujet. En effet, le sujet de la réalité virtuelle ou augmentée prend aujourd'hui de plus en plus d'ampleur, tant dans le domaine des loisirs que professionnel. C'est ce que nous allons voir autour d'un sujet qui mettra en œuvre le framework ARTtoolkit de la société DAQRI.

Mots-clés : Réalité augmentée, Android, ARToolKit, java, OpenGL, C++

Résumé

Afin de mettre en œuvre ARToolKit, nous allons présenter cette librairie qui se retrouve assez régulièrement au cœur des projets utilisant la réalité augmentée sur des images réelles.

Afin de créer une forte réalité augmentée, cette librairie utilise des capacités de suivi vidéo qui calcule la position de la caméra réelle ainsi que l'orientation par rapport aux marqueurs physiques ou des marqueurs de fonction naturelle en temps réel. Une fois que la position de la caméra réelle est connue, une caméra virtuelle peut être positionnée au même point et les modèles d'infographie 3D dessinés sont superposés exactement sur le marqueur réel.

ARToolKit résout donc deux des problèmes-clés dans la réalité augmentée : suivi de point de vue et interaction d'un objet virtuel. C'est ce que nous allons voir, comprendre et mettre en œuvre au sein d'un exemple simple.

ARToolKit a été initialement développé par Hirokazu Kato du *Nara Institute of Science and Technology* en 1999 et a été libéré par l'Université de Washington HIT Lab. En 2001, le groupe **ARToolWorks** a été constitué et la version v1.0 open source de ARToolKit a été libérée par le laboratoire de HIT.

ARToolKit a été l'un des premiers SDKs de réalité augmentée pour mobile : d'abord sur **Symbian** en 2005, puis sur **Apple/iOS** avec l'**iPhone 3G** en 2008 et enfin sur **Google/Android** dès 2010 avec une version professionnelle par ARToolWorks un peu plus tard en 2011. ARToolKit a été finalement acquise par **DAQRI** et réédité en open source LGPLv3 à partir de la version 5.2 , le 13 mai 2015, y compris toutes les fonctionnalités qui étaient auparavant uniquement disponibles dans la version professionnelle sous licence. Parmi ces fonctionnalités, il y a le support mobile et le suivi des caractéristiques naturelles.

L'utilisation est libre dans le cadre d'une utilisation non commerciale, des licences commerciales sont disponibles sous le nom ARToolWorks.

1 Description de ARToolKit

ARToolKit est un logiciel permettant aux programmeurs de développer facilement des applications de réalité augmentée avec une palette de possibilités

comme le divertissement, les médias, l'industrie, le médical, la publicité et la recherche universitaire.

Le code source de ce projet est hébergé sur **GitHub** et fournit en téléchargement un SDK compilé pour toutes les plateformes suivantes : **Mac OS X**, **Windows**, **GNU/Linux**, Android, iOS. Il fournit enfin un *plug-in* ARToolKit pour **Unity3D**.

Chacun des téléchargements de plateformes individuelles comprend un ensemble d'exemples d'applications pour vous aider à appréhender cette librairie. Une documentation détaille les API et fournit des didacticiels et des exemples de débutant à des niveaux d'experts. Un soutien supplémentaire peut être trouvé en rejoignant le forum ARToolKit géré par la communauté.

À l'inverse des effets spéciaux utilisés au cinéma, les principales problématiques de la réalité augmentée proviennent du fait que les mouvements de caméra et de la scène ne sont pas connus à l'avance. L'enrichissement de la réalité doit se faire en temps réel et à la volée lors de l'acquisition des images sans aucune information de départ sur l'environnement. Afin de donner l'illusion que les objets réels et virtuels appartiennent au même monde, il faut pourtant tisser un lien entre eux afin de les positionner correctement (coordonnées, échelle, orientation) par rapport aux objets réels filmés.

Bien que plusieurs méthodes peuvent être imaginées pour cela, ARToolkit utilise un système de marqueurs (*pattern*). Un marqueur est un motif simple sur fond blanc entouré d'un cadre noir (voir figure 1). L'algorithme doit tout d'abord détecter puis identifier ce marqueur, puis en déduit en temps réel sa position et son orientation par rapport à la caméra. Cet algorithme sera détaillé dans la partie suivante.

Par la suite, l'ajout d'objets virtuels se fait par rapport aux repères en trois dimensions dont l'origine, la taille et les directions sont calculées image par image à partir du marqueur.

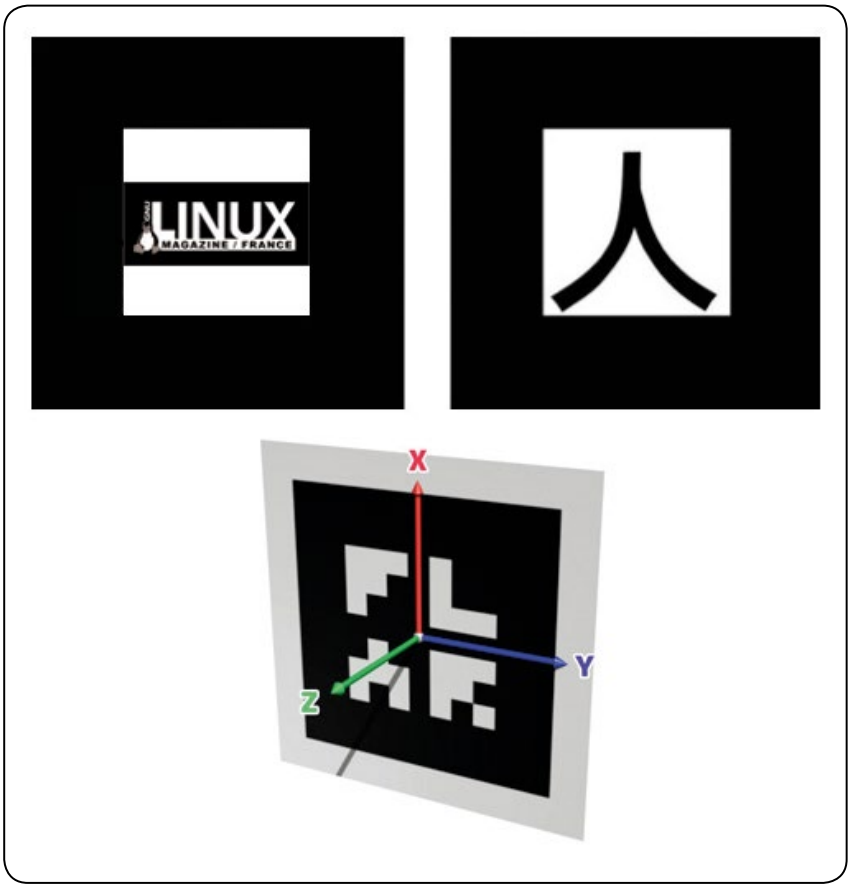


Fig. 1 : Exemples de patterns reconnus par ARToolKit.

2 Principe de fonctionnement de ARToolkit

2.1. Algorithme de détection : principe général

L'algorithme de détection d'ARToolkit effectue une suite d'opérations sur chaque image du flux vidéo afin de repérer la présence d'un marqueur, puis de l'identifier parmi les différents marqueurs chargés dans l'application (dans le cadre d'une application multi-marqueurs). Son fonctionnement est illustré en figure 2 et peut se résumer ainsi :

1. La caméra capture la vidéo et l'envoie vers l'ordinateur ;
2. Le programme « binarise » l'image puis recense tous les cadres noirs dans chaque image de la vidéo ;

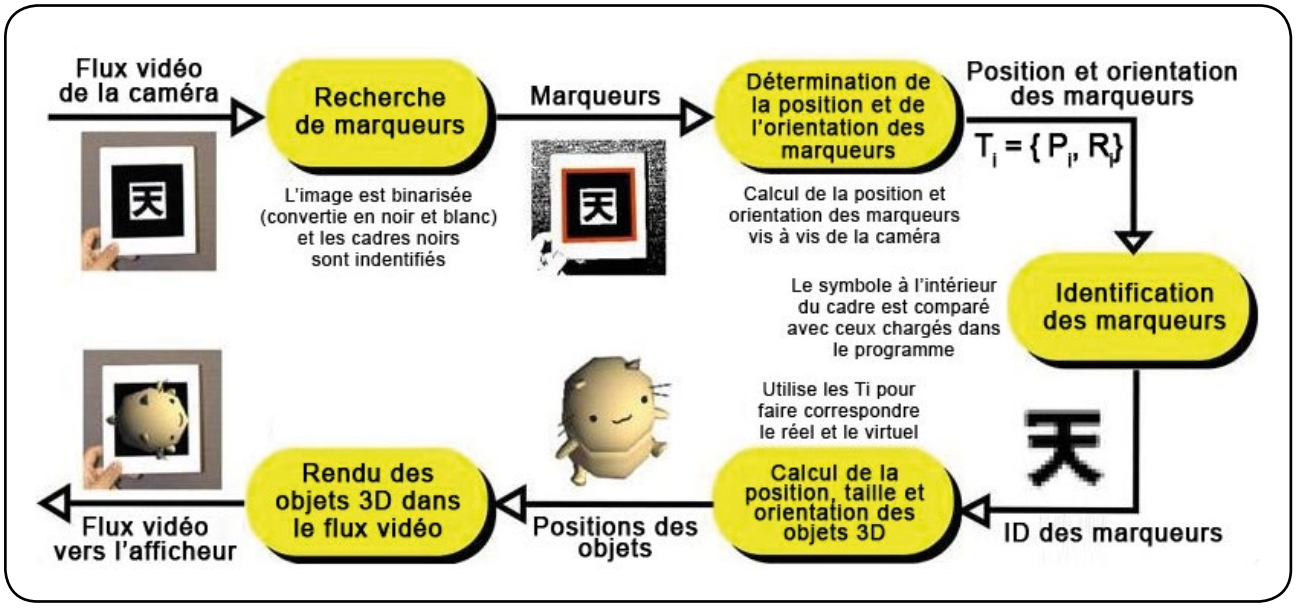


Fig. 2 : Fonctionnement de l'algorithme de détection d'ARToolKit.

3. Pour chaque cadre trouvé, le programme utilise des formules mathématiques pour déterminer sa position et son orientation vis-à-vis de la caméra ;
4. Le motif présent à l'intérieur de chaque cadre est comparé avec les patterns chargés dans le programme afin de lui associer l'augmentation virtuelle qui lui est associée ;
5. Cette augmentation, qui prend généralement la forme d'un objet 3D, est alors générée à partir de la position et de l'orientation du marqueur et est superposée à l'image capturée. Sa position, orientation et échelle sont alors ajustées par rapport au marqueur en temps réel.

2.2 Binarisation

La première étape fondamentale de l'identification est donc la binarisation de l'image issue de la caméra en noir et blanc, sans aucune nuance de gris (voir figure 3). Cette dernière est évaluée selon un paramètre nommé niveau de seuil, ou frontière, configuré en en-tête du programme.

Ce dernier convertit tout d'abord le flux en niveaux de gris et tous les pixels dont la valeur est inférieure au seuil, seront noirs et inversement.

2.3 Normalisation

Bien entendu dans la majorité des cas, le motif sera déformé par l'orientation du marqueur vis-à-vis de la caméra.

C'est là que le carré noir trouve toute son utilité, en effet à partir de la position de chacun de ses sommets il est possible de normaliser le motif afin de pouvoir procéder à son identification.

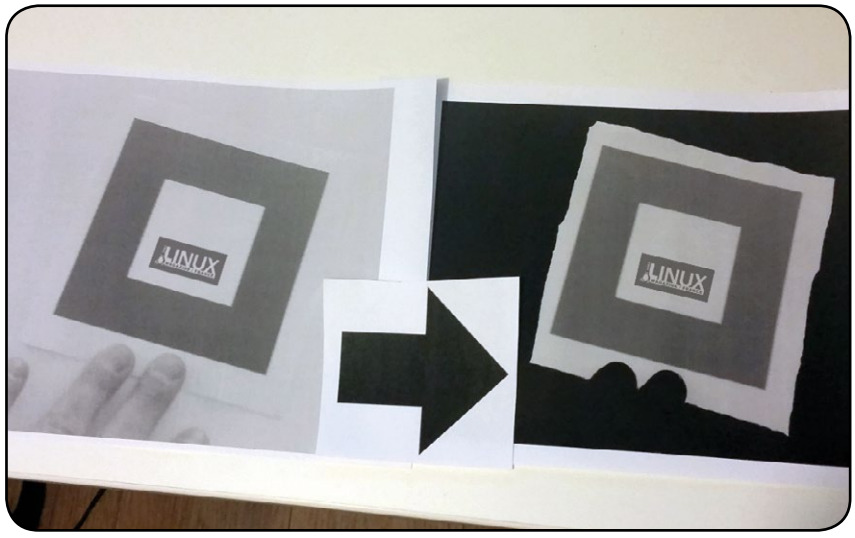


Fig. 3 : Binarisation de l'image.

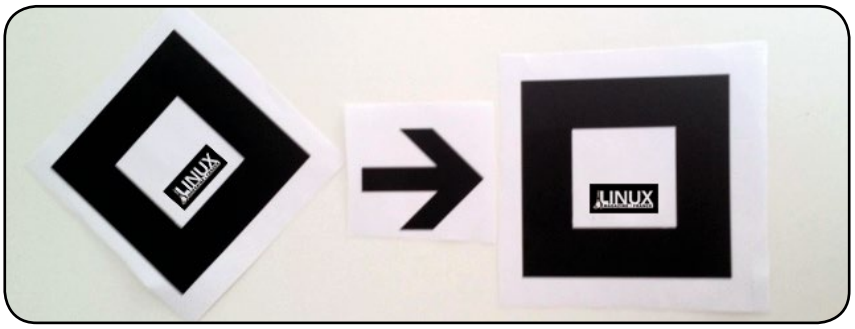


Fig. 4 : Normalisation du motif.

2.4 Identification du marqueur

Le motif redressé est ensuite comparé aux *patterns* chargés dans le programme (voir figure 5). Ces derniers sont localisés dans un fichier texte où chaque pixel est représenté par sa valeur en niveaux de gris (de **0** à **255**).

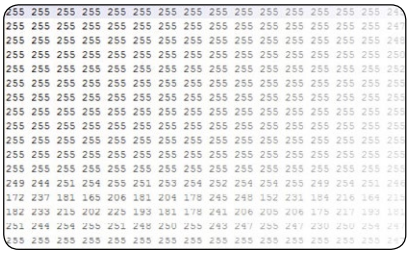


Fig. 5 : Exemple de pattern utilisé pour détecter un marqueur.

Les degrés de similitude ou « confidence » sont ensuite calculés par rapport à chaque fichier **.pat** dans quatre sens différents, et le plus haut est retenu. Cela permet ensuite de déterminer et de générer l'augmentation associée.

3 Installation du SDK ARToolkit

3.1 Récupération et configuration

ARToolKit est facilement téléchargeable à l'adresse suivante : http://artoolkit.org/download-artoolkit-sdk (voir figure 6). À noter aussi l'existence du GitHub https://github.com/artoolkit/artoolkit5.

L'installation s'effectue en plusieurs temps :

1. Téléchargement de l'archive de la bibliothèque ;
2. Extraction et configuration des sources par rapport aux modules à activer ;
3. Compilation des modules.

Commençons par télécharger l'archive :

```
$ mkdir -p arttoolkit && cd arttoolkit/
$ wget http://www.artoolkit.org/dist/artoolkit5/5.3/ARToolKit5-bin-5.3.2r1-Linux-i686.tar.gz
--2016-09-04 17:50:48--  http://www.artoolkit.org/dist/artoolkit5/5.3/ARToolKit5-bin-
5.3.2r1-Linux-i686.tar.gz
Resolving www.artoolkit.org (www.artoolkit.org)... 54.215.18.24
...
$ tar zxf ARToolKit5-bin-5.3.2r1-Linux-i686.tar.gz
$ ls -al
total 14648
drwxrwxr-x  9 tgayet tgayet     4096 Sep  4 17:55 .
drwxr-xr-x 14 tgayet tgayet     4096 Sep  4 17:53 ..
-rw-rw-r--  1 tgayet tgayet 14746816 Apr  1 05:01 ARToolKit5-bin-5.3.2r1-Linux-i686.tar.gz
-rw-rw-r--  1 tgayet tgayet    72425 Mar 29 07:20 ChangeLog.txt
-rwxrwxr-x  1 tgayet tgayet    22592 Mar 22 00:21 Configure
-rw-rw-r--  1 tgayet tgayet    45179 Aug 11  2015 LICENSE.txt
-rwxrwxr-x  1 tgayet tgayet      621 Jun 15  2015 Makefile.in
-rw-rw-r--  1 tgayet tgayet    10348 Mar 29 07:20 README.txt
drwxrwxr-x  7 tgayet tgayet     4096 Sep  4 17:55 bin
drwxrwxr-x  6 tgayet tgayet     4096 Jun 15  2015 doc
drwxrwxr-x 15 tgayet tgayet     4096 Sep  4 17:55 examples
drwxrwxr-x  9 tgayet tgayet     4096 Sep  4 17:55 include
drwxrwxr-x  5 tgayet tgayet     4096 Sep  4 17:55 lib
drwxrwxr-x  4 tgayet tgayet     4096 Sep  4 17:55 share
drwxrwxr-x 15 tgayet tgayet     4096 Sep  4 17:55 util
```

Une fois décompressé, il est nécessaire de procéder à quelques paramétrages. Le premier va servir à définir le répertoire racine pour ARToolKit :

```
tgayet@tgayet:~/Documents/arttoolkit$ ./share/artoolkit5-setenv
*** Successfully set ARTOOLKIT5_ROOT to /home/tgayet/Documents/arttoolkit ***
```

Le suivant permet l'activation des modules via le script shell **Configure** :

```
tgayet@tgayet:~/Documents/arttoolkit $ ./Configure

This script configures ARToolKit libraries, utilities and examples. For details on
dependencies and further information please visit http://artoolkit.org/documentation/doku.
php?id=8_Advanced_Topics:build_artoolkit


Standard compiler is gcc and g++ with libstdc++. Do you want to use the Clang compiler with
libc++ instead? (y or n)
Enter : n
```

```
Do you want to enable the Video4Linux2 capture module? (y or n)
Enter : y
Do you want to use it as the default input device? (y or n)
Enter : y
Do you want to enable the Video4Linux capture module? (y or n)
Enter : y
Do you want to enable the IEEE 1394 Digital Video Camera capture module? (y or n)
Enter : n
Do you want to enable the GStreamer capture module? (y or n)
Enter : n
Enable the VRML renderer and example? (y or n)
Enter : n
Enable the OpenSceneGraph renderer and examples? (y or n)
Enter : y
  create ./Makefile
...
Done.
```

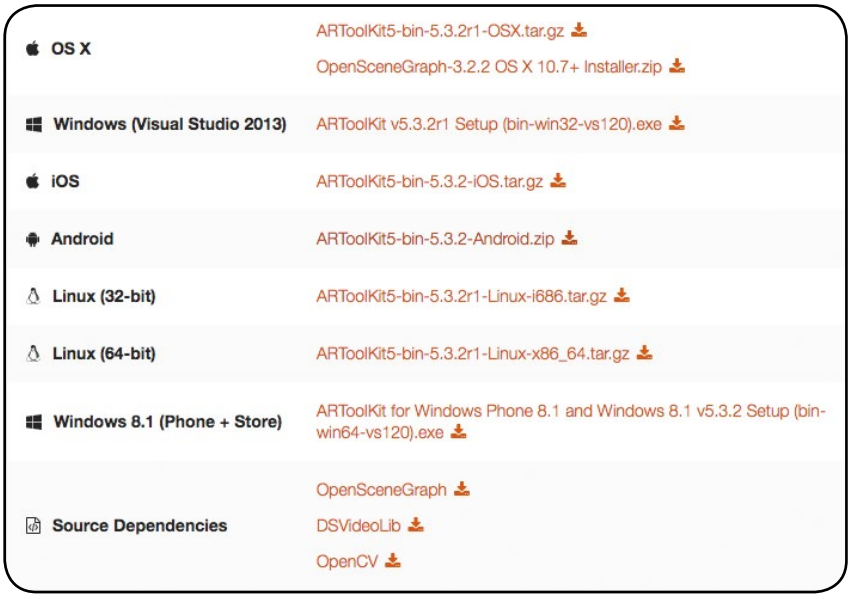


Fig. 6 : Page de téléchargement d'ARToolKit. Comme on peut le voir ci-dessus, la bibliothèque est disponible en 32 ou 64 bits pour GNU/Linux.

3.2 Compilation

Avant de lancer la génération, installons quelques dépendances :

```
$ sudo apt-get install freeglut3-dev clang libv4l-0 libv4l-dev libgl1-mesa-dev libjpeg-
dev libdbus-glib-1-2 libdbus-glib-1-dev libopencv-dev libc++-dev libopenscenegraph100v5
libopenscenegraph-dev libltdl-dev libboost-all-dev libgtkglext1 libgtkglext1-dev libgnomeui-0
libgnomeui-common libgnomeui-dev xorg-dev freeglut3-dev mesa-utils libudev-dev llvm
```

À noter que pour la compilation, un compilateur **gcc** 4.8 ou supérieur (minimum 4.4) est recommandé pour ARToolKit. Les dépendances peuvent évoluer selon les versions de distribution Linux. ARToolKit nécessite la version 1.5 d'**OpenGL**. Pour vérifier la version :

```
tgayet@tgayet:~/Documents/arttoolkit $
glxinfo | grep "OpenGL version"
libGL: OpenDriver: trying /usr/lib/x86_64-
linux-gnu/dri/tls/swrast_dri.so
libGL: OpenDriver: trying /usr/lib/x86_64-
linux-gnu/dri/swrast_dri.so
libGL: driver does not expose __
driDriverGetExtensions_swrast(): /usr/
lib/x86_64-linux-gnu/dri/swrast_dri.so:
undefined symbol: __driDriverGetExtensions_
swrast
libGL: Can't open configuration file /home/
amahealth/.drirc: No such file or directory.
libGL: Can't open configuration file /home/
amahealth/.drirc: No such file or directory.
libGL error: failed to load driver: swrast
OpenGL version string: 1.4 (2.1 ATI-1.42.15)
```

Si jamais vous avez un problème, n'hésitez pas à exporter la variable d'environnement suivante :

```
$ export LIBGL_DEBUG=verbose
```

Si vous n'avez pas la bonne version, vous pouvez télécharger une version plus actuelle depuis le site même de MESA EGL : http://mesa3d.sourceforge.net / https://www.khronos.org/egl/ .

Ensuite, nous allons *builder* la version 1.18.9 de la librairie **openvrml** qui nous sera nécessaire plus tard :

```
$ wget http://downloads.
sourceforge.net/project/openvrml/
openvrml/0.18.9/openvrml-0.18.9.tar.
gz?r=https%3A%2F%2Fsourceforge.net%2Fp
roject%2Fplatformdownload.php%3Fgroup_
id%3D7151&ts=1473065827&use_mirror=netix
$ mv openvrml-0.18.9.tar.
gz\?r=https%3A%2F%2Fsourceforge.net%2F
project%2Fplatformdownload.php%3Fgroup_
id%3D7151 openvrml-0.18.9.tar.gz
$ tar zxf openvrml-0.18.9.tar.gz && cd
openvrml-0.18.9
```

Aujourd'hui, l'implémentation de la bibliothèque **libboost_thread** est sécurisée avec l'usage en mode *multi-threading* (on dit habituellement *thread-safe*). Auparavant, cette bibliothèque avait un suffixe qui n'a plus lieu d'être et qui doit être retiré dans le fichier autotools (**configure.ac**) :

C'est simple... c'est possible sur :

http://www.ed-diamond.com

... OU SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE AU VERSO ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

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.

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

LINUX MAGAZINE / FRANCE

Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

.....

.....

VOICI TOUTES LES OFFRES COUPLÉES AVEC GNU/LINUX MAGAZINE !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

Prix en Euros / France Métropolitaine

Offre	SUPPORT / ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC	PAPIER + PDF (PDF 1 lecteur) Réf	PAPIER + PDF (PDF 1 lecteur) Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Tarif TTC	PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Réf	PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Tarif TTC
LM	11n° GLMF	LM1	65,-	LM12	95,-	LM13	149,-	LM123	174,-
LM+	11n° GLMF + 6n° HS	LM+1	118,-	LM+12	177,-	LM+13	197,-	LM+123	256,-
LES COUPLAGES « LINUX »									
A	11n° GLMF + 6n° LP	A1	95,-	A12	140,-	A13	218,-	A123	263,-
A+	11n° GLMF + 6n° HS + 6n° LP + 3n° HS	A+1	182,-	A+12	263,-	A+13	300,-	A+123	386,-
B	11n° GLMF + 6n° MISC	B1	100,-	B12	147,-	B13	233,-	B123	280,-
B+	11n° GLMF + 6n° HS + 6n° MISC + 2n° HS	B+1	172,-	B+12	248,-	B+13	300,-	B+123	381,-
C	11n° GLMF + 6n° LP + 6n° MISC	C1	135,-	C12	197,-	C13	312,-	C123	374,-
C+	11n° GLMF + 6n° HS + 6n° LP + 3n° HS + 6n° MISC + 2n° HS	C+1	236,-	C+12	339,-	C+13	403,-	C+123	516,-
LES COUPLAGES « EMBARQUÉ »									
F	11n° GLMF + 6n° HK* + 4n° OS	F1	125,-	F12	188,-	F13	229,-*	F123	292,-*
F+	11n° GLMF + 6n° HS + 6n° HK* + 4n° OS	F+1	183,-	F+12	275,-	F+13	287,-*	F+123	379,-*
LES COUPLAGES « GÉNÉRAUX »									
H	11n° GLMF + 6n° HK* + 6n° LP + 6n° MISC + 4n° OS	H1	200,-	H12	300,-	H13	402,-*	H123	499,-*
H+	11n° GLMF + 6n° HS + 6n° HK* + 6n° LP + 3n° HS + 6n° MISC + 2n° HS + 4n° OS	H+1	301,-	H+12	452,-	H+13	493,-*	H+123	639,-*

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | OS = Open Silicium | HC = Hackable

* HK : Attention : La base Documentaire de Hackable n'est pas incluse dans l'offre.

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :

http://www.ed-diamond.com pour consulter toutes les offres !

```
$ mv configure configure.orig
```

Nous allons le *patcher* pour pouvoir fonctionner avec la libboost_thread. Détail du patch :

```
--- configure.ac 2012-04-29 10:07:01.000000000 +0200
+++ openvrml-0.18.9/configure.ac 2016-09-05 12:06:39.516086022 +0200
@@ -100,12 +100,12 @@
 #
 # Allow users to specify any Boost library name suffix
 #
-AS_IF([test -z "${BOOST_LIB_SUFFIX+x}"], [BOOST_LIB_SUFFIX=-mt])
-AC_ARG_VAR([BOOST_LIB_SUFFIX], [Boost library name suffix [default=-
mt]])
+#AS_IF([test -z "${BOOST_LIB_SUFFIX+x}"], [BOOST_LIB_SUFFIX=])
+#AC_ARG_VAR([BOOST_LIB_SUFFIX], [Boost library name suffix
[default=]])

 AC_CACHE_CHECK([for boost_thread$BOOST_LIB_SUFFIX library],
 [ov_cv_boost_thread],
-[ov_cv_boost_thread=no
+[ov_cv_boost_thread=yes
 ov_save_LIBS=$LIBS
 LIBS="-lboost_thread$BOOST_LIB_SUFFIX $LIBS"
 AC_LANG_PUSH([C++])
@@ -115,8 +115,8 @@
 AC_LANG_POP
 LIBS=$ov_save_LIBS
 ])
-AS_IF([test X$ov_cv_boost_thread = Xno],
-      [AC_MSG_FAILURE([libboost_thread$BOOST_LIB_SUFFIX not found])])
+#AS_IF([test X$ov_cv_boost_thread = Xno],
+#      [AC_MSG_FAILURE([libboost_thread$BOOST_LIB_SUFFIX not found])])

 #
 # The XmlTextReader interface appears in libxml 2.5.
```

Et pour *patcher* le fichier :

```
tgayet@tgayet:~/Documents/arttoolkit/openvrml-0.18.9 $ patch -p0 <
openwrml-configure-ac.patch
tgayet@tgayet:~/Documents/arttoolkit/openvrml-0.18.9 $ autoreconf
-i --force
```

Une fois le fichier configuré patché et regénéré, on peut lancer la configuration puis le build :

```
tgayet:~/Documents/arttoolkit/openvrml-0.18.9 $ /configure --disable-
script-node-javascript --disable-script-node-java --disable-mozilla-
plugin --disable-examples --disable-xembed -disable-player -prefix=/usr
tgayet@tgayet:~/Documents/arttoolkit/openvrml-0.18.9 $ make -j4 # assez
long
tgayet@tgayet:~/Documents/arttoolkit/openvrml-0.18.9 $ sudo make
install
```

Maintenant que la brique logicielle openvrml est installée, nous pouvons générer les bibliothèques d'ARToolKit :

```
tgayet@tgayet:~/Documents/arttoolkit $ cd ..
tgayet@tgayet:~/Documents/arttoolkit $ make -j4 # assez long
```

Une fois terminé, après un ou deux cafés, tous les exemples ont normalement dû être compilés. Un **ls** dans le répertoire **bin** affichera **simpleLite**, **simpleOSG**, etc.

3.3 Mac OS X

Pour le monde Apple, une fois l'archive décompressée, il est nécessaire de lancer le projet **xcode** pour d'éventuelles modifications. À noter que le SDK est fourni avec des exemples précompilés, c'est-à-dire prêts à l'emploi, ce qui fait gagner du temps ! Plus globalement, ARToolKit est développé premièrement pour Mac OS X puis *packagé* sous Windows ou Linux.

3.4 Tests

Une fois le logiciel installé et configuré, on peut lancer certains tests comme **simpleLite** :

```
tgayet@tgayet:~/Documents/arttoolkit/bin $ ./simpleLite
```

Comme détaillé dans la documentation (https://artoolkit.org/documentation/doku.php?id=7_Examples:example_simplelite), simpleLite détecte un marqueur « Hiro » et affiche un cube de couleur au-dessus (voir l'affichage du cube en figure 7).

Fig. 7 : Affichage d'un cube par dessus le marqueur « Hiro » dans l'exemple simpleLite.

Si la détection fonctionne comme sur la photo, bravo vous venez de mettre en place votre première application de réalité augmentée. De plus, cela confirmera que votre compilation fonctionne correctement.

4 Les marqueurs et fichiers de signature

4.1 Que sont les marqueurs?

Les marqueurs sont les carrés que ARToolKit reconnaît dans un flux vidéo. Les marqueurs sont des modèles physiques créés ou imprimés. ARToolKit est livré avec des fichiers PDF pour certains marqueurs préconstruits. Par exemple, vous trouverez le marqueur Hiro, que vous pouvez imprimer et apposer sur une surface cartonnée de sorte qu'il reste bien à plat. Les marqueurs sont les entrées optiques pour ARToolKit (voir en figure 8 le marqueur Hiro).

Fig. 8 : Marqueur Hiro.

Les marqueurs ont cependant quelques contraintes :

- Ils doivent être carrés ;
- Ils doivent avoir une bordure continue (généralement en noir ou en blanc pur) et ils doivent utiliser un fond de couleur contrastant (généralement à l'opposé de la couleur de la bordure). Par défaut, l'épaisseur de la bordure est de 25% de la longueur d'un bord du marqueur ;
- La dernière contrainte est que la zone à l'intérieur de la frontière, que nous appelons *image du marqueur*, ne doit pas être symétrique et en rotation. La zone à l'intérieur de la frontière peut être en noir et blanc ou en couleur (ARToolKit fournit un moyen de suivre avec une grande précision lorsque l'image du marqueur est colorée).

ARToolKit version professionnelle offre une fonctionnalité supplémentaire permettant d'utiliser des marqueurs qui contiennent une grille spéciale à deux dimensions de carrés noirs et blancs (un peu comme un code à barres 2D) à la place de l'image du marqueur habituel. L'utilisation de ces marqueurs peut accélérer le suivi quand il y a beaucoup de marqueurs dans la scène.

4.2 Les fichiers de signatures ?

Les fichiers de signatures sont des fichiers qui contiennent des données qui représentent une image au milieu d'un marqueur. Quand ARToolKit se lance, il charge généralement un ou plusieurs fichiers de signatures pour qu'il sache les marqueurs à rechercher dans le flux vidéo.

Les fichiers de motifs permettent à ARToolKit de distinguer les marqueurs que vous souhaitez suivre parmi d'autres objets carrés dans la scène et de différencier un marqueur d'un autre. Vous pouvez trouver le fichier de signatures pour le marqueur Hiro dans votre distribution de ARToolKit dans le fichier `bin/Data/patt.hiro`.

4.3 Création de nouveaux marqueurs

Vous pouvez créer un nouveau marqueur en modifiant le modèle fourni dans votre distribution de ARToolKit, dans le fichier `doc/modèles/pattern.png`. Vous pouvez créer des marqueurs de toute taille, et vous pouvez mélanger différentes tailles de marqueurs. Lorsque vous utilisez le marqueur dans ARToolKit, vous pouvez spécifier à ARToolKit la taille du marqueur grâce à un fichier de configuration.

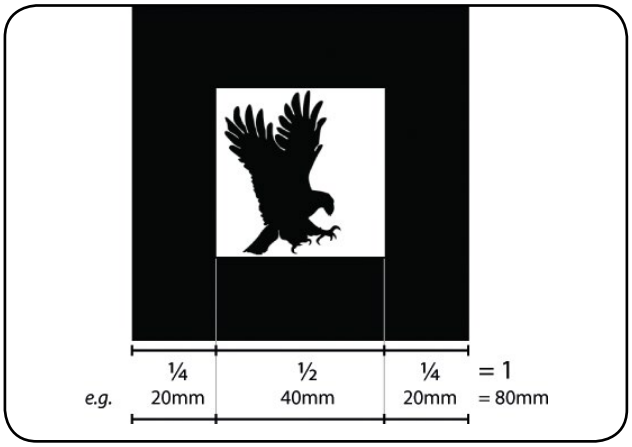


Fig. 9 : Un marqueur en forme d'aigle.

L'intérieur des 50 % du marqueur est interprété comme l'image du marqueur par ARToolKit, selon l'image de la figure 9. Notez que l'image peut être de couleur, blanche sur noire ou noir sur blanc, et peut se prolonger dans la région frontalière. Rappelez-vous que la partie de l'image en dehors de l'intérieur des 50% sera ignorée par ARToolKit, et il faut aussi être sûr de ne pas prolonger trop loin dans la frontière : ARToolKit pourrait ne pas reconnaître le marqueur du tout avec un angle très oblique de l'appareil photo.

Si vous utilisez des marqueurs 2D codes à barres, vous pouvez trouver les images de marqueurs dans votre distribution d'ARToolKit, dans le dossier **`doc/pattern/code Matrix 3x3`**.

5 Cas pratique

Dans cet exemple, nous allons créer un marqueur avec le logo de *GNU/Linux Magazine* et lorsque la camera le détectera, on affichera un Tux modélisé en 3D grâce au logiciel **Blender**. Tout en gardant ce projet fournit dans le SDK comme base, nous allons détailler comment générer un marqueur personnalisé, mais aussi un Tux en VRML. Informations complémentaires sur **simpleVRML** : https://vimeo.com/24197846.

5.1 Création du marqueur

Pour créer le nouveau marqueur avec le logo *GNU/Linux Magazine* (voir figure 10), il est possible d'utiliser le générateur en ligne suivant : http://flash.tarotaro.org/blog/2008/12/14/artoolkit-marker-generator-online-released/.

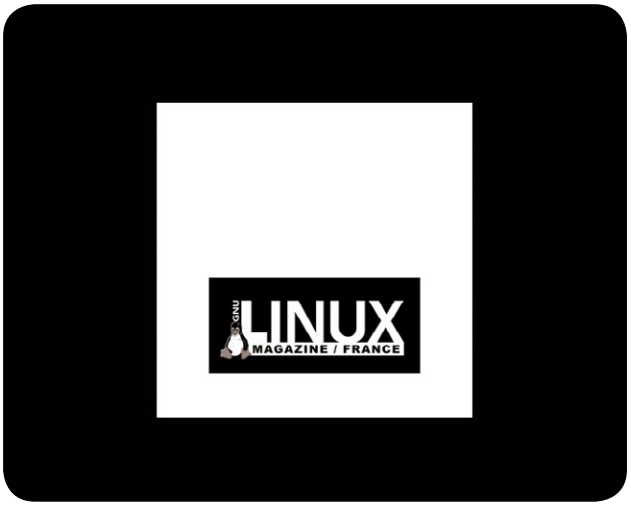


Fig. 10 : Le marqueur GNU/Linux Magazine.

Le fichier exporté donnera un fichier **`markerlinuxmag16.pat`** :

```
107 117 119 118 119 120 119 119 120 120 120 121 121 120 120 120
108 119 119 119 119 119 120 119 120 120 120 122 120 120 120 119
108 118 119 119 119 120 119 119 119 121 120 121 120 119 120 120
108 118 119 119 120 119 119 120 120 120 120 120 120 120 119 119
108 118 119 119 120 120 119 120 120 120 121 120 121 121 120 120
108 118 119 119 120 120 119 120 121 121 121 120 121 120 119 118
106 119 118 119 120 119 119 120 120 120 120 120 120 120 120 118
118 118 118 119 120 119 119 121 120 120 120 120 121 120 119 120
118 118 118 119 119 119 119 119 119 119 120 119 119 120 119 119
118 118 119 118 119 119 115 103 107 109 111 112 113 118 119 119
119 119 113  43  14  43  46  36  35  39  56  51  60  89 119 117
...
```

Une fois capturé par la caméra, on appuiera sur le bouton ***Get pattern*** puis ***save*** pour récupérer le fichier du marqueur au format pat. Ce fichier sera chargé au lancement du projet simpleVRML pour pouvoir identifier le marqueur que nous venons de créer. Il existe bien d'autres techniques, mais cet outil flash est vraiment très pratique.

5.2 Création de l'animation VRML avec un Tux en 3D

De façon à disposer d'un Tux, inutile de réinventer la roue, car il existe en ligne sur l'Internet : http://reprap.org/wiki/File:Tux.blend. Une fois le modèle blender changé dans le modeleur, il est possible de l'exporter au format x3d équivalent au format vrml (voir figure 11 page suivante).

5.3 Au cœur de l'exemple

Le code source que nous allons utiliser est localisé dans le chemin **`example/simpleVRML`** qui contient quatre fichiers : **`Makefile.in`**, **`object_vrml.c`**, **`object_vrml.h`**, et **`simpleVRML.c`**.

Les fichiers **`object_vrml .c`** et **`.h`** servent à gérer le fichier VRML ou x3b. Le fichier principal est le fichier **`simpleVRML.c`**, les fonctions principales étant :

- **`setupCamera()`** : l'initialisation de la caméra ;
- **`Keyboard()`** : gestion des interactions avec l'utilisateur ;
- **`mainloop`** : boucle principale de détection des marqueurs et de l'affichage de l'objet VRML ;
- **`main()`** : point d'entrée du programme initialisant OpenGL, lançant le setup de la caméra puis la **`mainloop`**.

Si vous modifiez le code, il vous suffira de relancer le **`Makefile`** de la racine d'ARToolKit.

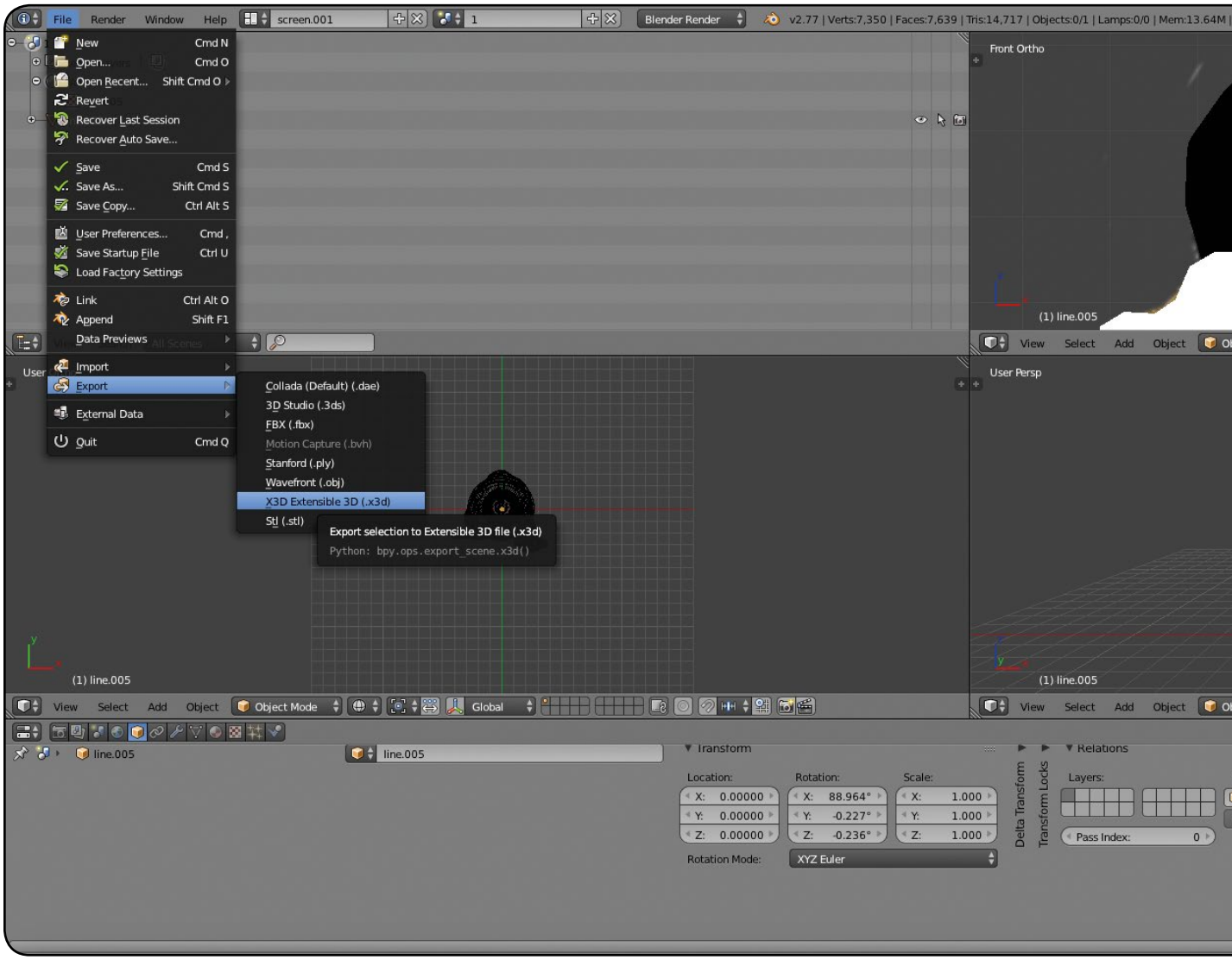


Fig. 11 : Export du modèle Blender au format x3d.

À noter que la simplicité de l'exemple est rendue possible grâce aux bibliothèques dont le code est dépendant : **libArvrml**, **libArgsub_lite**, **libArvideo**, **libAR**, **libARICP**, **libopenvrml**, **libopenvrml-gl**, **libstdc++**, **libjpeg**, **libpng**, **libz** et **libm**.

5.4 Modification de la configuration de l'exemple simpleVRML

Le premier fichier est le fichier **bin/Data/object_data_vrml** :

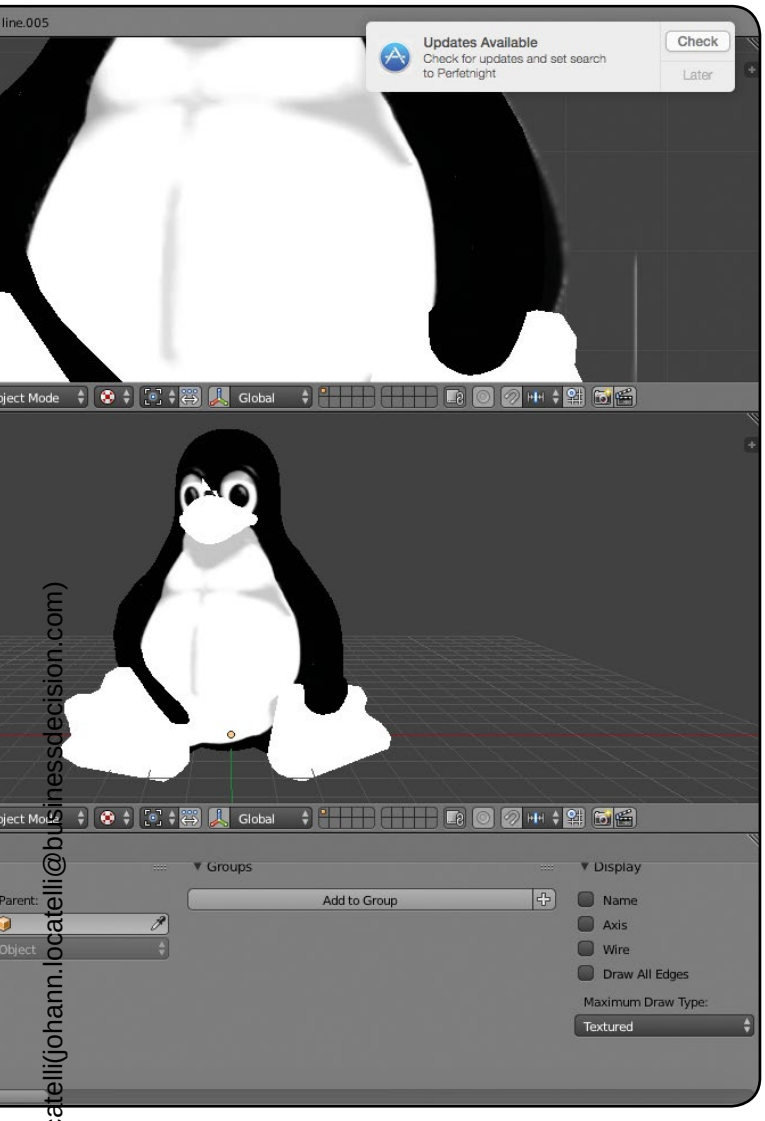
```
#the number of patterns to be recognized
1

#pattern 1
VRML Wrl/linuxmag.dat
Data/markerlinuxmag16.pat
80.0
0.0 0.0
```

On reconnaît le fichier de pattern **linuxmag16.pat** qui contient les infos de notre marqueur. Enfin, il reste le fichier du Tux en 3D **Wrl/linuxmag.dat** :

```
Tux.wrl
0.0 0.0 25.0   # Translation
90.0 1.0 0.0 0.0 # Rotation
0.75 0.75 0.75   # Scale
```

On y voit une déclaration avec le fichier de l'animation Tux en VRML, mais notre fichier x3b est lui tout aussi fonctionnel. À noter que l'on aurait pu déclarer plusieurs patterns. Pour ce faire, il aurait suffi de dupliquer la déclaration du pattern dans le fichier **bin/Data/object_data_vrml**.



5.5 Exécution

L'exécution est simple, car il suffira de lancer le binaire **simpleVRML** dans le répertoire **bin** :

```
tgayet@tgayet:~/Documents/arttoolkit/openvrml-0.18.9 $ /simpleVRML
```

À l'exécution, il chargera les différents fichiers de configuration ainsi que les animations et le fichier de pattern. Le programme est pilotable avec certaines touches :

- **a** : bascule entre les différents niveaux de seuil ;
- **c** : calcule le nombre de *frames* ;
- **d** : (dés)active le mode debug ;
- **m** : (dés)active le mode informatif ;
- **q** ou **<esc>** : quitte le programme ;
- **x** : change le mode de traitement ;
- **?** ou **/** : affiche ou cache l'aide contextuelle ;
- **-** et **+** : incrémente/décrémente de **±5** le niveau de seuil.

Conclusion

Voilà, j'espère que cela vous aura donné envie de goûter au monde de la réalité augmentée qui est en train de s'installer dans la vie de tous les jours et que même peut-être réaliserez-vous un jeu comme Pokemon Go ou bien une application industrielle pour votre entreprise. L'imagination est sans limites. ■

Pour aller plus loin

- Site officiel : https://artoolkit.org
- GitHub officiel : https://github.com/artoolkit/
- SourceForge officiel : http://artoolkit.sourceforge.net
- Doxygen des API : https://mirror.umd.edu/roswiki/doc/diamondback/api/artoolkit/html/
- Build depuis des sources : https://artoolkit.org/documentation/doku.php?id=8_Advanced_Topics:build_artoolkit
- Drivers OpenGL : http://mesa3d.sourceforge.net
- Librairies openvrml : http://www.openvrml.org
- Site web de la communauté ARToolKit : https://www.hitl.washington.edu/artoolkit/
- Détail de la compilation de l'exemple simpleVRML : http://3d.sangji.ac.kr/ppt/VR/AR_Code.pdf
- Site OpenClassroom sur ARToolKit : https://openclassrooms.com/forum/sujet/processing-artoolkit

Pour approfondir le sujet sur ARToolkit, il est intéressant de pouvoir profiter de la grosse communauté incluant une chaine YouTube https://www.youtube.com/user/artoolworks ainsi qu'un forum https://artoolkit.org/community/forums/.

Pour avoir un bon aperçu des possibilités de ce qui est faisable en réalité augmentée, je vous renvoie vers le cours en ligne suivant : http://perso-etis.ensea.fr/alexpitt/papers/cours_IHM_M2_4_2013.pdf.

CONCEPTION D'UN SYSTÈME DE TÉLÉ-INFORMATION EDF

Michel RAMBOUILLET [Professeur honoraire de Génie Électrique option Informatique et Télématique]

La télé-information peut permettre, outre la surveillance de la consommation électrique, de piloter par exemple le système de chauffage (pompe à chaleur/ chaudière fuel) en fonction des différentes périodes tarifaires EDF. Nous envisagerons de placer ce projet dans un cadre domotique général d'informatique répartie et d'insister sur la conception et la réalisation logicielle, notamment à l'aide de diagrammes de type UML.

Mots-clés : UML, Raspberry Pi, C++, Python, Web, Service

Résumé

Après avoir, dans une première partie, présenté le contexte et le cahier des charges de notre application puis décrit les cas d'utilisation [1], nous avons défini les opérations système et retenu une architecture pour notre système de télé-information (voir figure 1). Nous avons ensuite montré comment réaliser et tester la plateforme matérielle à l'aide d'un système embarqué de type Raspberry Pi équipé d'une carte de prototypage supportant le câblage des interfaces avec le bus de télé-information EDF et les leds de signalisation.

Nous allons poursuivre notre description par la conception et la réalisation logicielle du service de télé-information. Pour ce faire, nous utiliserons à nouveau une notation inspirée d'UML (*Unified Modeling Language*) [2] et [3].

1 Conception du service de télé-information

1.1 Les concepts du domaine de l'application

L'analyse des cas d'utilisation fait apparaître le concept fondamental de **trame**. L'étude de la documentation ERDF **[4]** nous renseigne sur les attributs de cette classe conceptuelle : les éléments du *contrat* souscrit, les valeurs courantes de l'*intensité* et des *index* de consommation ainsi que des éléments liés à l'option tarifaire choisie (période tarifaire en cours, événements liés aux changements de période). Afin de respecter la chronologie dans la réception des trames, nous ajouterons un marqueur temporel (date et heure).

Un autre concept important est celui d'**historique**. On pourra retenir l'attribut *compteur* associé (on pourrait envisager de gérer l'historique de plusieurs compteurs EDF). Plutôt qu'une simple liste ordonnée d'informations, nous choisissons de regrouper celles-ci en journaux. On retiendra donc comme composant de l'historique le **journal** avec en attribut la *date* du jour (servant d'ordre sur les journaux) lui-même composé d'une liste ordonnée d'informations (`InfosJournal`) dont

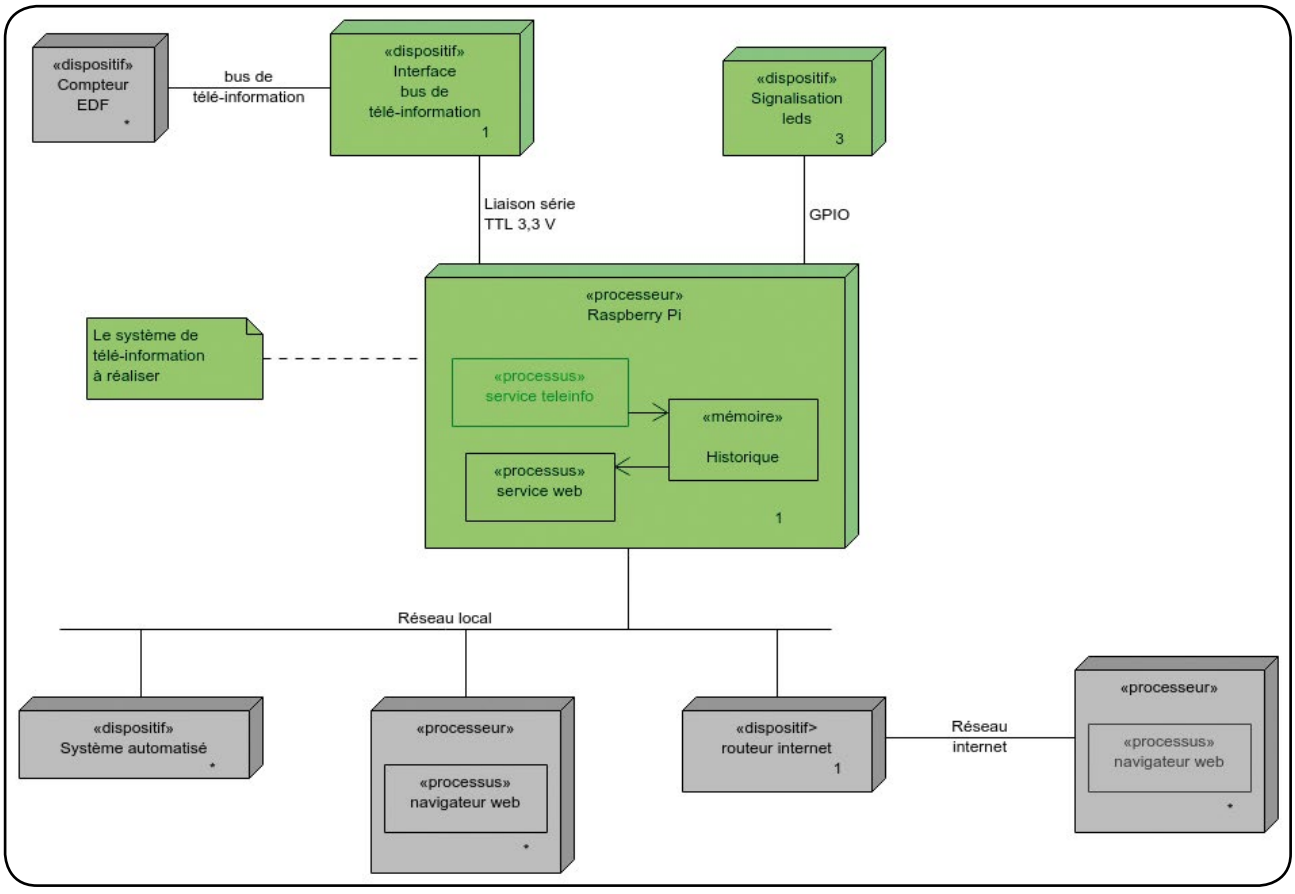


Figure 1 : Rappel de l'architecture du système de télé-information.

les attributs sont *heure* (pour l'ordre), la *période tarifaire en cours* et les différents *index* de consommation utiles.

Enfin, nous utiliserons le concept d'**événement tarifaire** sous forme d'une énumération pour laquelle nous ne retiendrons ici que les événements correspondants à un contrat de type EJP (mais il est aisé d'en ajouter d'autres). L'événement `EV_INCONNU` est utile lors d'une reprise d'activité du système.

La figure 2 récapitule les concepts du domaine.

1.2 La modélisation des opérations système

La figure 3 montre le diagramme de séquence système retenu pour le cas « Gérer télé-information ». Il correspond à l'analyse d'une trame par le système. Voyons comment réaliser ces opérations.

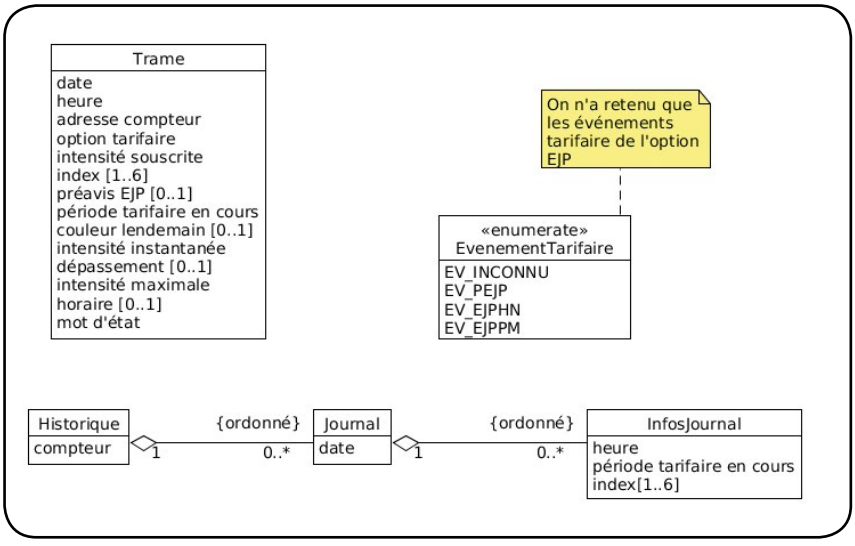


Figure 2 : Concepts du domaine de système de télé-information.

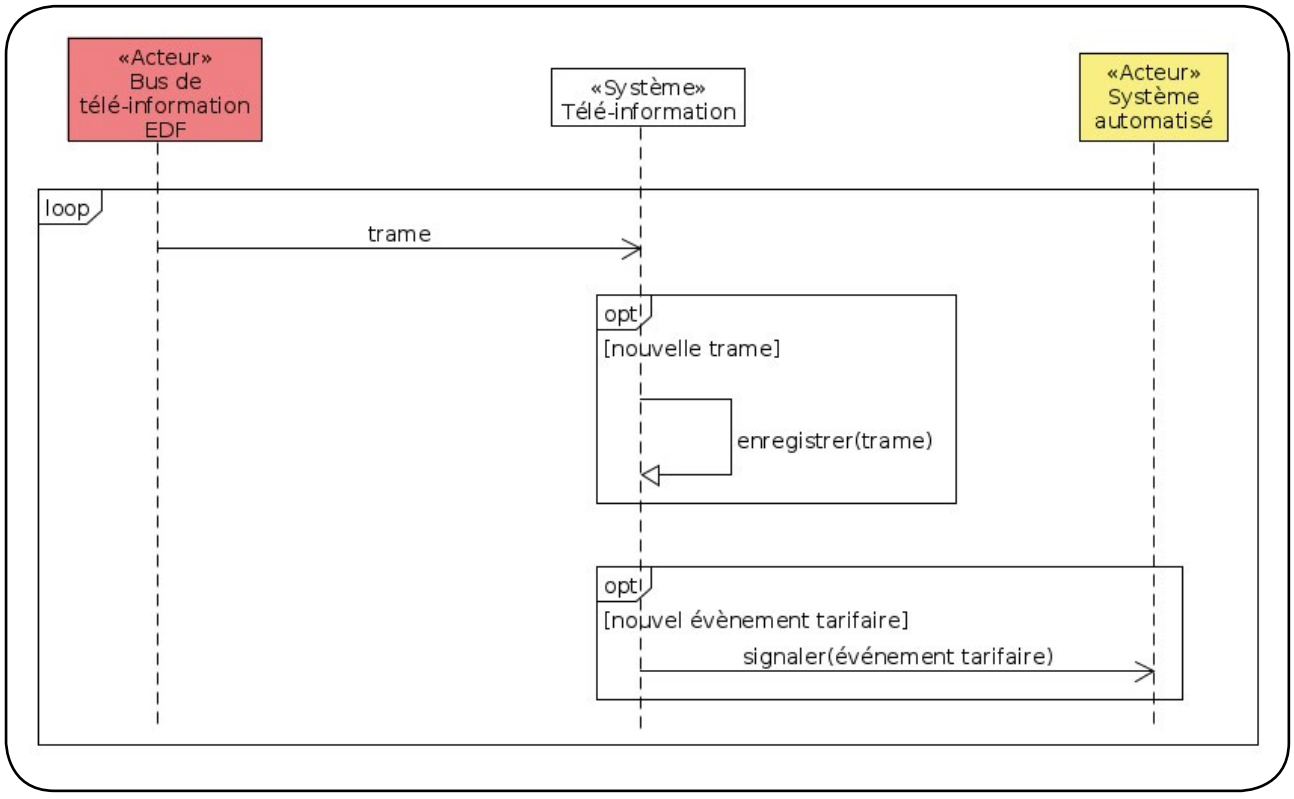


Figure 3 : DSS du cas « Gérer télé-information ».

L'analyse des données d'une trame sera confiée à un « contrôleur », instance de la classe **GestionnaireTeleinfo** dont la méthode **analyserTrames()** réalise l'enchaînement décrit dans le DSS de la figure 3. Pour y parvenir, il fait appel à un objet de type **LecteurTeleinfo** qui fournit les trames en provenance du bus de télé-information ; cet objet est responsable de la détection des trames qui sont reçues de manière asynchrone sur le port série relié au bus de télé-information : sa méthode **lireTrame():Trame** fournit la dernière trame valide reçue (on notera que, au regard de la constitution d'une trame, le système est insensible à l'occurrence de trames erronées comme à une perte momentanée de trames). L'objet **Trame** retourné par la lecture fournit les opérations utiles à la gestion des informations contenues dans celle-ci (voir figure 2).

Lorsqu'une trame est lue, le contrôleur est chargé de détecter si son contenu est différent de celui de la trame précédente (nouvelle trame) et en particulier de traiter les occurrences d'événements tarifaires. En cas de nouvelle trame, le contrôleur fait appel à une instance de la classe **Historique** pour en assurer la persistance (méthode **enregistrerTrame(:Trame)**) en respectant l'organisation en journaux décrite précédemment. Lors de l'occurrence d'un nouvel événement tarifaire, le contrôleur délègue la signalisation de cet événement à un **AgentNotifieur**.

Le cahier des charges de l'application impose au moins deux types de signalisation pour les événements : signalisation à leds et signalisation réseau. L'agent notifieur maintient une liste d'outils de signalisation : un notifieur à leds (classe **NotifieurLeds**) et un notifieur réseau (classe **NotifieurReseau**) ; ces notifieurs implémentent l'interface **NotifieurEvenement** dont la méthode **signalerEvenement(:EvenementTarifaire)** permet à l'agent de déclencher la signalisation indépendamment de la technologie utilisée.

Le **NotifieurLeds** est chargé de faire clignoter les leds pour la signalisation visuelle ; il est composé de trois instances de la classe **Led** : **led_verte**, **led_jaune** et **led_rouge**. L'implémentation de l'interface **NotifieurEvenement** fait appel aux méthodes **clignoter()**, **alterner(:Led)**, **stopper()** de la classe **Led**. La méthode **alterner(:Led)** permet de faire clignoter de manière alternative deux leds afin d'obtenir un effet visuel plus attractif.

Le **NotifieurReseau** est chargé quant à lui de diffuser les événements sur le réseau local.

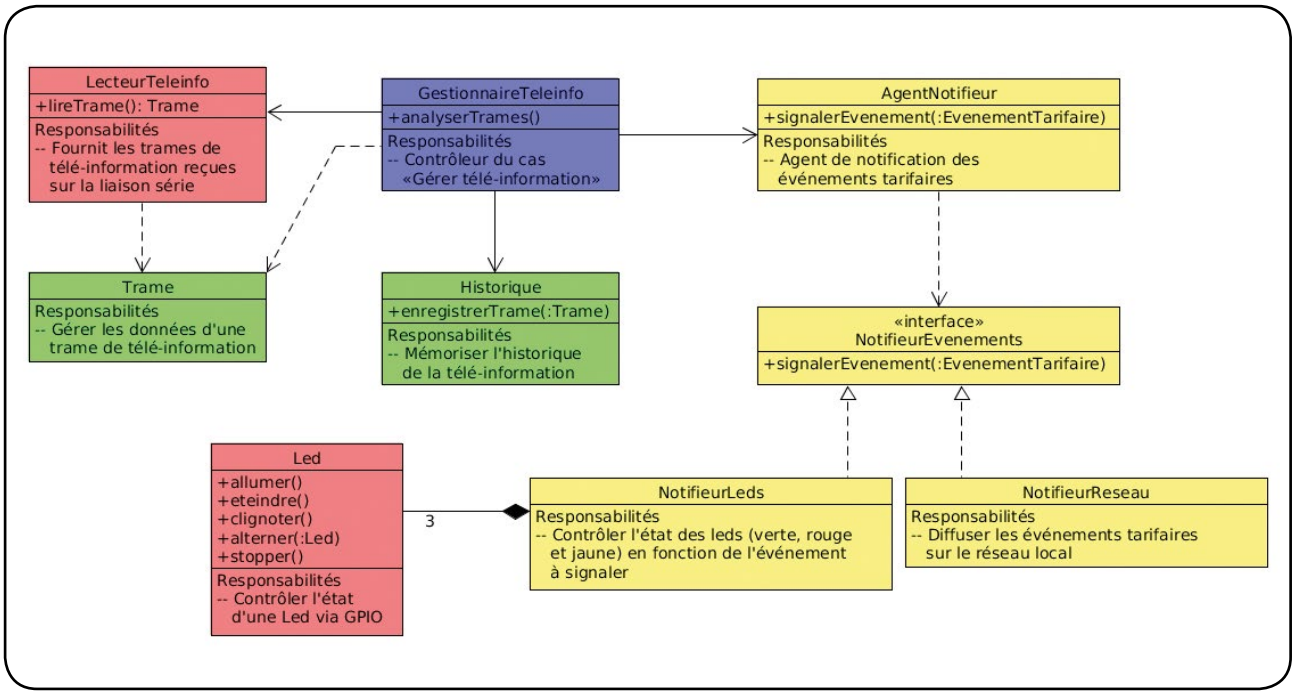


Figure 4 : Diagramme de classes participantes du système de télé-information.

On obtient finalement le diagramme de classes de la figure 4.

On trouvera en figure 5 (page suivante) un diagramme de séquence montrant la collaboration des instances de ces classes lors de la détection d'un événement de type « préavis EJP » ; on suppose que l'événement précédent était du type « heures normales » : il s'agit donc d'une nouvelle trame (qu'il convient d'enregistrer) et d'un nouvel événement (qu'il convient de signaler). L'agent notifieur fait diffuser l'événement sur le réseau, stopper le clignotement de la led verte puis fait clignoter la led jaune.

2 Conception détaillée du service de télé-information

Le service de télé-information sera réalisé en C++. Pour que l'exécutable soit lancé automatiquement au démarrage du système, nous en ferons un service système qui pourra donc être démarré/arrêté avec les commandes standards Linux. Ce service fera appel à des primitives système pour accéder à des fichiers, des entrées/sorties, etc. Il convient de réfléchir à une stratégie de gestion des erreurs.

Lorsqu'une erreur sera détectée, nous lancerons une exception qui sera captée à l'endroit adéquat du service pour être enregistrée à des fins d'analyse. Il peut par ailleurs être utile d'enregistrer l'occurrence de certains événements à titre d'information sur le fonctionnement du service.

Nous ferons pour cela appel au système de journalisation du système Linux via `syslog()` comme dans les exemples ci-après : `syslog(LOG_INFO, "INFO : Service teleinfo démarré\n");` ou bien pour une `exception e`, `syslog(LOG_ERR, "ERREUR : %s\n", e.what());`. Le niveau de journalisation sera fixé au moyen de `setlogmask(LOG_UPTO(LOG_INFO));` pour la journalisation des erreurs et des informations utiles ou `setlogmask(LOG_UPTO(LOG_DEBUG));` pour plus de détails lors de la mise au point.

Afin de renseigner utilement le message retourné par la méthode `what()` des exceptions, nous sommes amenés à définir un classe `ExceptionTeleinfo` dérivée de la classe `Exception`, dont le constructeur sera chargé de mémoriser le message adéquat et qui surchargera la méthode `what()` pour restituer ce message (voir figure 6).

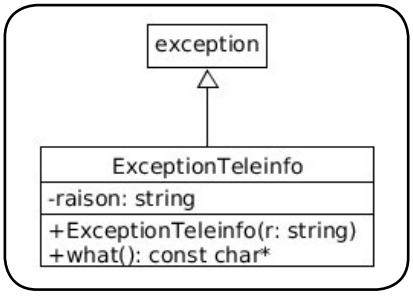


Figure 6 : La classe ExceptionTeleinfo.

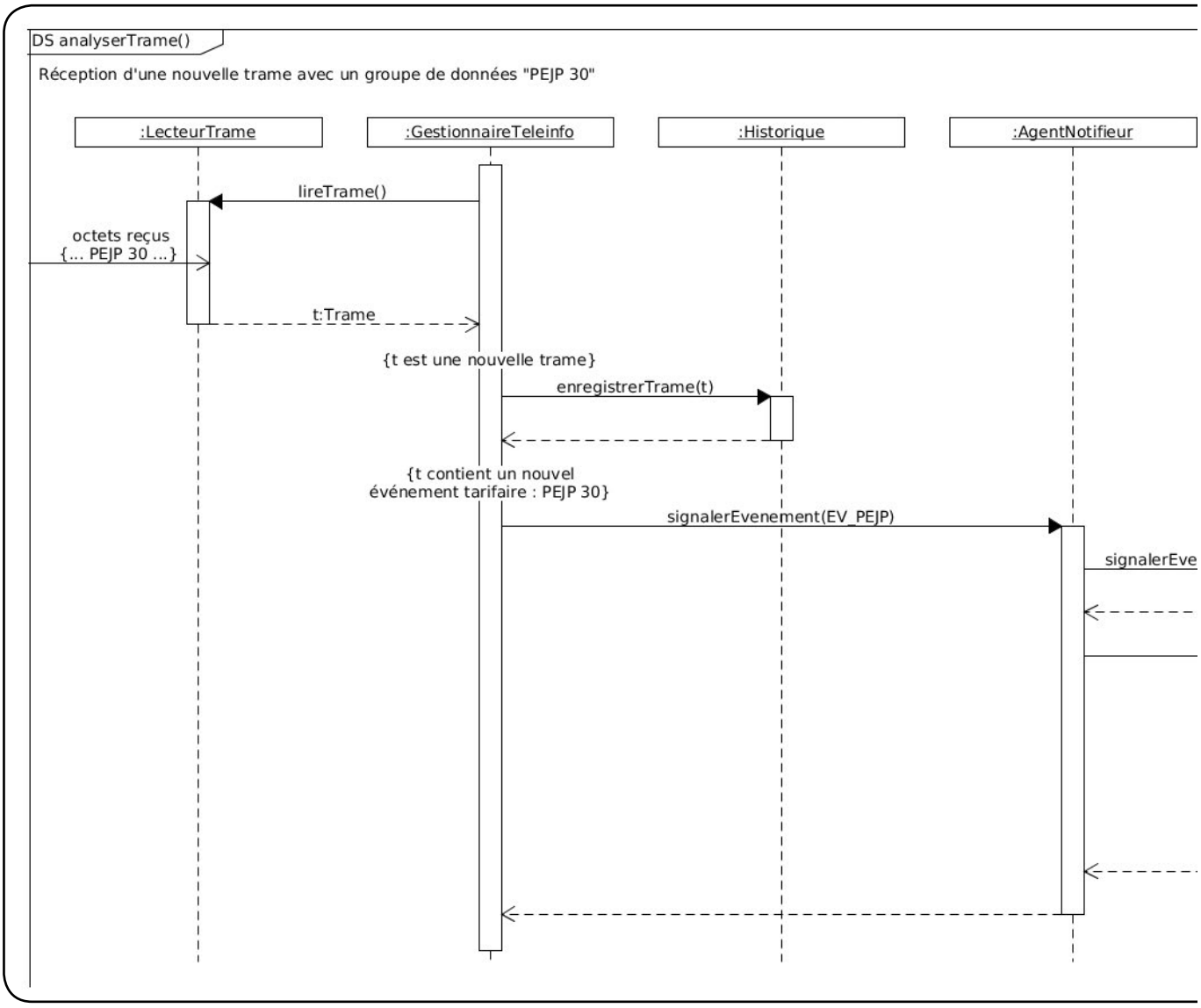


Figure 5 : Diagramme de séquence sur occurrence de l'événement « préavis EJP ».

2.1 Le module « teleinfo »

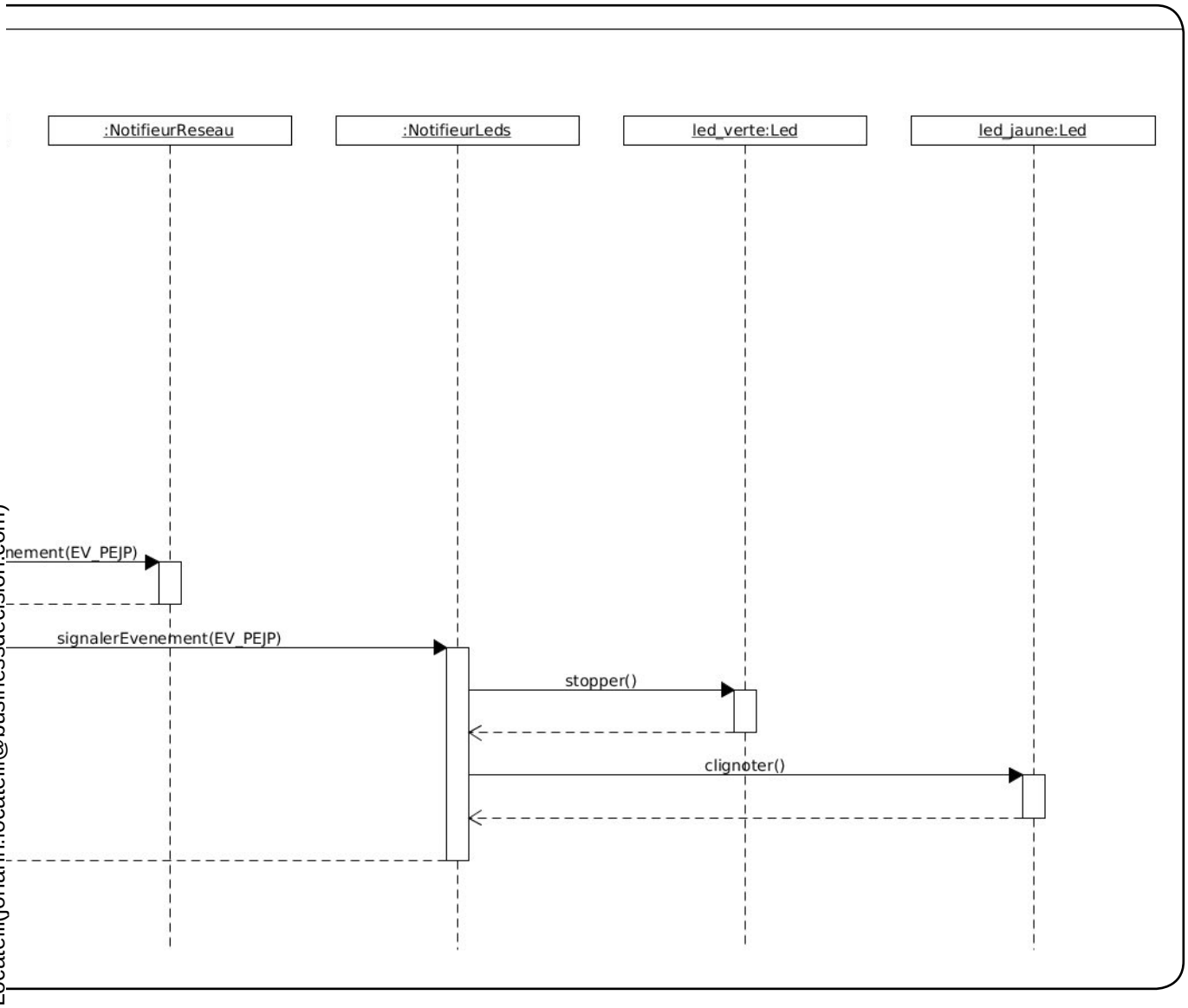
Ce module regroupe les classes directement liées à la télé-information : **`EvenementTarifaire`**, **`LecteurTeleinfo`** et **`Trame`**.

2.1.1 La classe « EvenementTarifaire »

Lors de l'étude des concepts du domaine, nous avons retenu celui d'**`EvenementTarifaire`** sous forme d'une énumération. La journalisation des événements tarifaires nécessite une représentation de ceux-ci sous la forme d'une chaîne de caractères ; aussi encapsulerons-nous ce concept dans une classe **`EvenementTarifaire`** dont la responsabilité sera de fournir un label pour chacune des valeurs possibles via sa méthode **`getLabel()`**. Le constructeur de cette classe permet de créer une instance associée à une des valeurs énumérées qui est mémorisée dans l'attribut **`code`**. Un attribut de classe (**`labels`**) mémorise les chaînes de caractères correspondant aux différentes valeurs.

2.1.2 La classe « LecteurTeleinfo »

Rappelons la structure des trames émises par le bus de télé-information EDF, telles que nous les avons obtenues avec la commande **`hexdump -C /dev/ttyAMA0`** :




```
00000230  45 54 41 54 20 30 30 30  30 30 30 20 42 0d 03 02  |ETAT 000000 B...|
00000240  0a 41 44 43 4f 20 30 33  39 35 30 31 30 32 33 36  |.ADCO 0395010236|
00000250  33 31 20 38 0d 0a 4f 50  54 41 52 49 46 20 45 4a  |31 8..OPTARIF EJ|
00000260  50 2e 20 22 0d 0a 49 53  4f 55 53 43 20 36 30 20  |P. "..ISOUSC 60 |
00000270  3c 0d 0a 45 4a 50 48 4e  20 32 34 36 38 30 30 30  |<..EJPHN 2468000|
00000280  37 35 20 45 0d 0a 45 4a  50 50 4d 20 30 30 38 35  |75 E..EJPPM 0085|
00000290  39 37 30 30 30 20 49 0d  0a 50 54 45 43 20 48 4e  |97000 I..PTEC HN|
000002a0  2e 2e 20 5e 0d 0a 49 49  4e 53 54 20 30 30 33 20  |.. ^..IINST 003 |
000002b0  5a 0d 0a 49 4d 41 58 20  30 35 39 20 4d 0d 0a 4d  |Z..IMAX 059 M..M|
000002c0  4f 54 44 45 54 41 54 20  30 30 30 30 30 30 20 42  |OTDETAT 000000 B|
000002d0  0d 03 02 0a 41 44 43 4f  20 30 33 39 35 30 31 30  |....ADCO 0395010|
```

Analysons une trame en suivant les indications de la documentation **[4]**. Le début de la trame est matérialisé par le caractère **STX** (**0x02**) ; elle se termine par le caractère **ETX** (**0x03**). Entre ces caractères, on trouve une séquence de groupes d'information encadrés par les caractères **LF** (**0x0A**) et **RC** (**0x0D**) ; chaque groupe est constitué d'une étiquette et d'une valeur séparées par un espace (**0x20**) et suivies d'un caractère de contrôle lui-même précédé d'un espace (**0x20**). Ainsi le premier groupe de la trame visible ci-dessus est formé de l'étiquette **ADCO**, dont la valeur est **039501023631** avec pour caractère de

contrôle **0x38**. Un autre caractère **EOT** (**0x04**) peut être présent indiquant la suspension de la trame par le bus et donc son annulation pour nous.

Notre objectif lors de la lecture sera de mémoriser cette trame sous une forme permettant d'extraire aisément les valeurs associées aux étiquettes et de vérifier la validité d'une trame en recalculant la somme de contrôle de chaque groupe. Nous illustrerons le processus d'identification d'une trame au moyen d'un diagramme d'état (figure 7) acceptant les caractères de la trame en entrée et produisant une nouvelle image de celle-ci en mémoire.

Le calcul de la somme de contrôle se fait en cumulant les codes ASCII des caractères du groupe, du début de l'étiquette à la fin de sa valeur. La normalisation consiste à rendre cette somme imprimable (code entre **0x20** et **0x5F**) en ne conservant que les 6 bits de poids faible, valeur à laquelle on ajoute **0x20**.

La forme retenue pour mémoriser la trame est une chaîne de caractères commençant par un marqueur temporel (date et heure) suivi des groupes d'informations séparés par des virgules, chaque groupe respectant le modèle **ETIQUETTE=VALEUR**. Ainsi la trame précédente sera mémorisée par la chaîne de caractères :

```
DATE=2016/03/11,HEURE=16:12:02,ADCO=039501023631,OPTARIF=EJP.,ISO
USC=60,EJPHN=246800075,EJPPM=008597000,PTEC=HN..,IINST=003,IMAX=0
59,MOTDETAT=000000
```

La trame sera mémorisée dans un tampon de 512 octets. Nous définirons deux méthodes pour la classe **LecteurTeleinfo** :

- Méthode publique **getTrame() : Trame**. Fournit une trame de télé-information EDF. La trame est préfixée par un marqueur temporel : **DATE=AAAA/MM/JJ,**

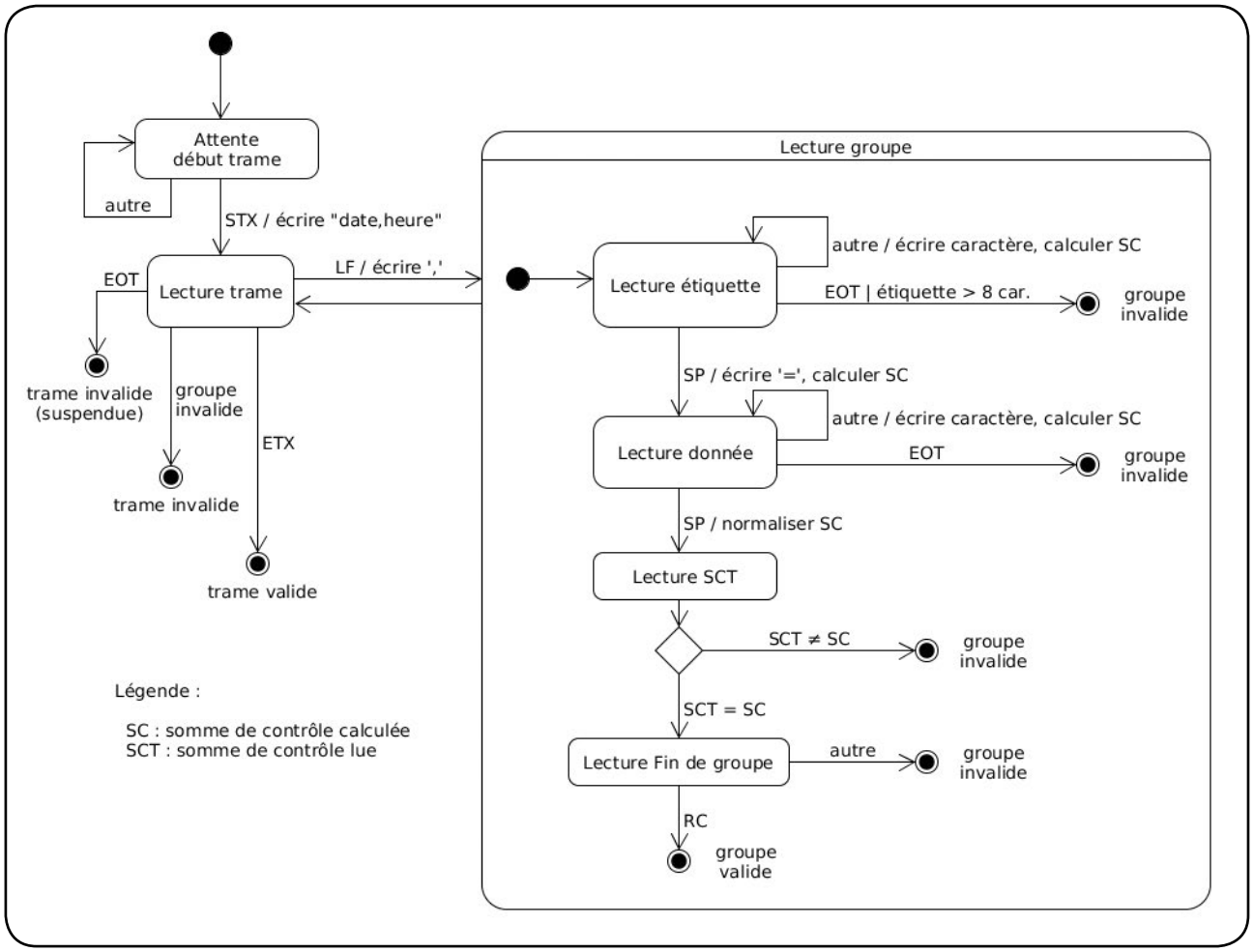


Figure 7 : Lecture d'une trame.

HEURE=HH:MM:SS qui est suivi des groupes d'informations **,ETIQUETTE=VALEUR**. Une trame peut être incomplète (suspendue) ou erronée (erreur sur la somme de contrôle, champ ou protocole incorrect). Dans ce cas, la lecture est abandonnée et on retourne une trame vide. Un nouvel appel provoquera une nouvelle synchronisation sur une trame suivante.

- Méthode privée **lireGroupe() : bool**. Cette méthode, utilisée par la méthode **getTrame()**, mémorise un groupe de télé-information dans le tampon et vérifie sa somme de contrôle (en cas d'erreur, la lecture est abandonnée). Retourne **true** si le groupe lu est valide ou **false** sinon.

Afin de rendre les méthodes précédentes indépendantes du mode de lecture des octets, nous définirons :

- Méthode privée **lireOctet() : char**. Attend un caractère sur le port série et retourne le caractère lu. Cette méthode est susceptible de lancer une **ExceptionTeleinfo**.

Outre un constructeur par défaut **LecteurTeleinfo()** utile pour les déclarations, nous définirons :

- Constructeur **LecteurTeleinfo(nomPort : const string)**. Construit une instance associée au port série dont le nom est passé en argument (pour nous **/dev/TTYAMA0**) et le configure avec les paramètres du bus EDF : 1200 bauds, parité paire, 7 bits de données, 1 bit de stop. Cette méthode est susceptible de lancer une **ExceptionTeleinfo**.

2.1.3 La classe « Trame »

La classe **Trame** a la responsabilité de gérer les données de télé-information. Nous mémoriserons ces données dans un tampon de 512 octets. Dans le DSS du cas « Gérer télé-information », on constate qu'il est nécessaire d'identifier si deux trames sont identiques et de détecter les événements tarifaires. Nous définirons les méthodes suivantes :

- Méthode **" friend " operator==(t1:const Trame&, t2:const Trame&):bool**. Teste l'égalité de deux trames. Deux trames sont considérées comme identiques si elles ne diffèrent que par leur marqueur temporel. On ne tient pas compte de l'identifiant du compteur qui est invariable dans le contexte ni du mot d'état qui est toujours nul.
- Méthode **getEvenementTarifaire() : EvenementTarifaire**. Retourne l'événement tarifaire contenu dans la trame. Cette méthode fait appel à **getValeur("PTEC")** pour extraire la période en cours et à **getValeur("PEJP")** pour le préavis éventuel (nous nous limitons à un contrat de type EJP).
- Méthode **getValeurEtiquette(e : const char*) : string**. Fournit la valeur associée à l'étiquette passée en argument.
- Méthode **getValeur() : string**. Fournit la représentation de la trame mémorisée dans le tampon.
- Méthode **valide() : bool**. La méthode **getTrame()** du lecteur fournit une trame vide en cas d'erreur. La méthode **valide()** permet de tester cette condition.

La dernière trame reçue doit être enregistrée pour alimenter l'historique, mais aussi pour fournir les données de consommation instantanée du cas « Consulter télé-information ». Nous choisissons d'assurer cette persistance dans un format adapté à l'exploitation dans une page HTML : json ; nous définirons donc une nouvelle méthode :

- Méthode **json() : string**. Fournit une représentation au format json de la trame. Ainsi pour la trame déjà donnée en exemple, on obtient :

```
{
  "marqueur" : { "date":"2016/03/11", "heure":"16:12:02" },
  "contrat" : { "compteur":"039501023631", "option":"EJP.",
"intensite":"60"},
  "consommation" : { "periode":"HN..", "intensite":"003",
"maximum":"059",
                                "index": [ "246800075",
"008597000" ] },
  "alertes": {}
}
```

Outre un constructeur par défaut qui instancie une trame vide, nous définirons :

- Constructeur **Trame(t : const char*)**. Construit une instance à partir des données du tampon **t** passé en argument.

La figure 8 (page suivante) présente le diagramme de classes du module « teleinfo ».

2.2 Le module « historique »

Reprenant le concept d'« Historique » déjà présenté, nous avons choisi d'implémenter la persistance sous la forme d'une arborescence de fichiers journaliers. La figure 9 (page suivante) montre l'organisation retenue.

Le répertoire **teleinfo/039501023631** contient les données de l'historique associé au compteur **039501023631**, à

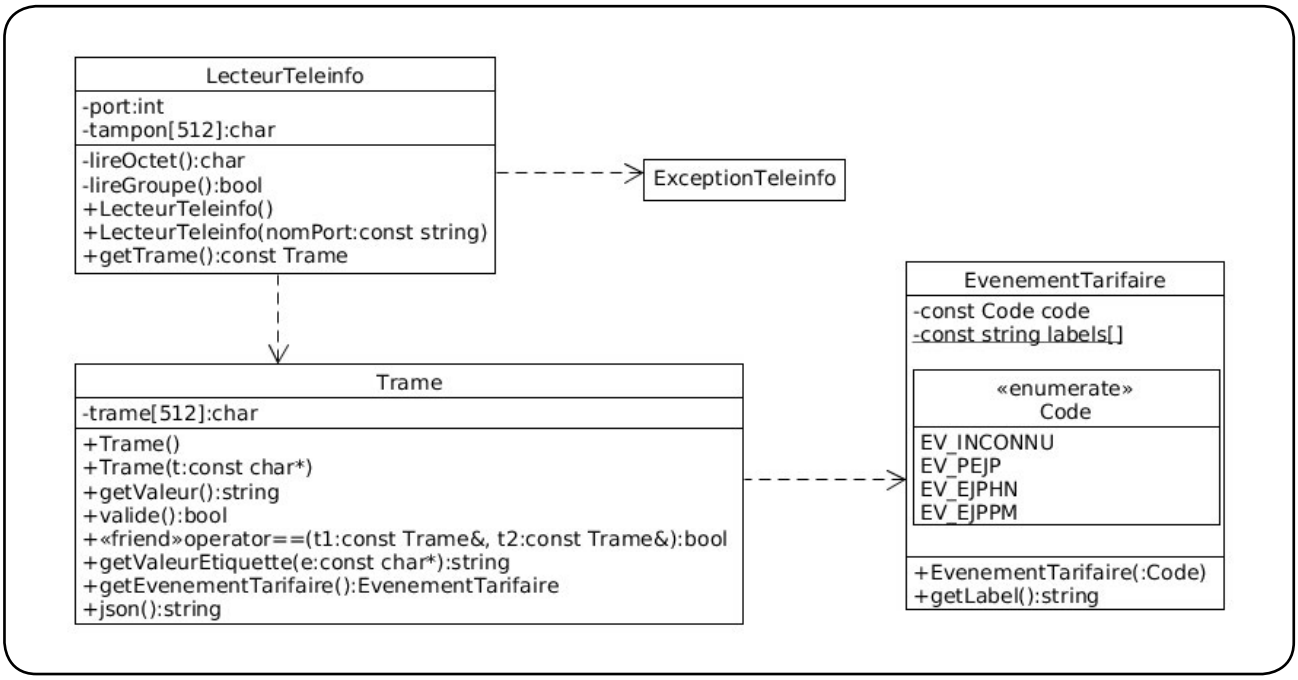


Figure 8 : Les classes du module « teteleinfo ».

savoir les fichiers **journal-24h** (les données du jour) et **trame-courante** (la dernière trame reçue). À la fin de chaque journée, les données du jour sont archivées dans le fichier **AAAA/MM/JJ** ; ainsi le fichier sélectionné sur la figure 9 contient les données du 2 mars 2016. Les attributs **compteur** et **date** des classes respectives **Historique** et **Journal** ainsi que la *relation d'agrégation* entre ces classes sont alors inscrits directement dans la structure des fichiers.

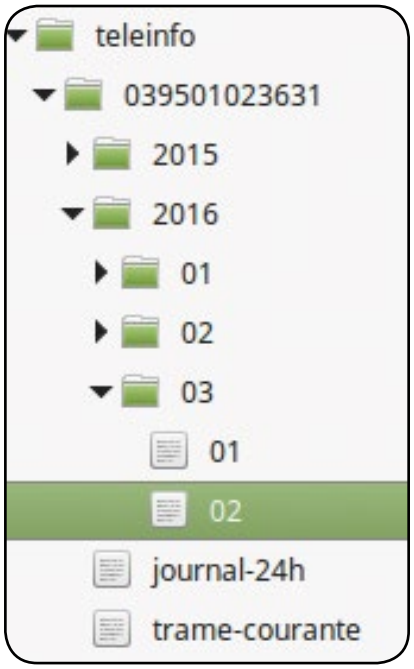


Figure 9 : La structure arborescente des fichiers journaliers.

Afin de limiter la taille des fichiers journaliers, on convient de mémoriser les informations avec une résolution temporelle fixée : les trames ne sont enregistrées qu'à intervalles de temps donnés en minutes. Cet intervalle de temps (résolution) sera un diviseur de **60**, soit une des valeurs : **1**, **2**, **3**, **4**, **5**, **6**, **10**, **12**, **15**, **20**, **30**.

Toutes les méthodes manipulant des fichiers ou répertoires sont susceptibles de lancer des exceptions de type **ExceptionTeleinfo**.

La figure 10 montre les classes collaborant à la réalisation du module « historique ».

2.2.1 La classe « Historique »

Outre la mémorisation de la trame courante, la classe « Historique », déjà évoquée, a la responsabilité de l'organisation arborescente des données ; elle délègue à la classe « Journal » la responsabilité de mémoriser l'historique journalier.

L'attribut **dateJournal** garde la trace de la date courante afin de gérer l'archivage en fin de journée (voir méthode **enregistrerTrame()**) ; la chaîne **repertoire** mémorise le répertoire de base de l'historique et **fTrame** est le flot de données utilisé pour la mémorisation de la trame courante.

Le constructeur **Historique(rep : string, resolution : int)**, construit l'instance **journal24h** de la classe **Journal** avec la résolution souhaitée et ouvre le flot de données de la trame courante ; si le répertoire de base n'existe pas, il est créé.

La méthode **enregistrerTrame(t : &Trame) : void** mémorise la trame courante dans le fichier **trame-courante**

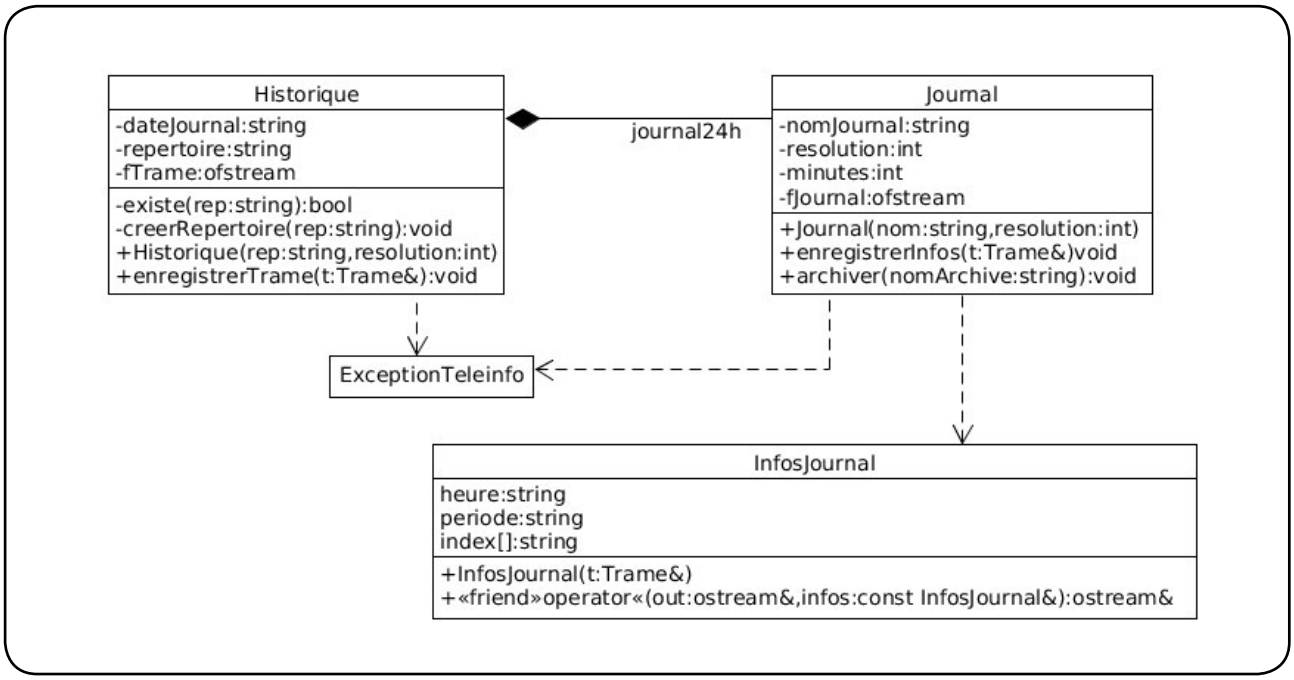


Figure 10 : Les classes du modules « historique ».

(au format json), délègue l'enregistrement des données de l'historique à l'instance **journal24h** de la classe **Journal** et déclenche (sur changement de la date des trames) la sauvegarde journalière du fichier **journal-24h** dans la structure arborescente en créant éventuellement les sous-répertoires utiles.

Afin de gérer la structure arborescente des fichiers, nous définirons les méthodes privées :

- Méthode **existe(repertoire : string) : bool**. Teste l'existence d'un répertoire afin de décider de sa création le cas échéant.
- Méthode **creerRepertoire(repertoire : string)**. Crée le répertoire fourni en argument ainsi que les répertoires parents qui n'existent pas encore.

2.2.2 La classe « Journal »

La classe **Journal** a la responsabilité d'enregistrer les données de l'historique. L'archivage est réalisé en déplaçant le fichier journal dont le nom est mémorisé par l'attribut **nomJournal** vers le fichier archive souhaité.

L'attribut **minutes** permet mémoriser le moment de la prochaine écriture dans le fichier suivant la résolution temporelle choisie.

Le constructeur **Journal(nom :string, resolution :int)** ouvre le fichier journal et fixe la résolution ; si la valeur fournie est incorrecte (i.e. n'est pas un diviseur de **60**), la valeur par défaut **30** est utilisée.

Dans cette classe on trouve aussi :

- Méthode **enregistrerInfos(t : Trame&) : void**. Si le champ minute de la trame correspond à l'attribut minutes, on construit une instance de **InfosJournal** avec les données de la trame et on insère celle-ci dans le flot du journal ; l'attribut **minutes** est alors incrémenté de la valeur de **resolution** modulo **60**.
- Méthode **archiver(string nomArchive) : void**. Ferme le journal, le déplace vers **nomArchive** puis ouvre à nouveau le journal.

2.2.3 La classe « InfosJournal »

La classe **InfosJournal** permet la sérialisation des données de l'historique, à savoir l'heure de la trame, la période tarifaire en cours et la valeur des index du compteur.

Le constructeur **InfosJournal(t : Trame&)** extrait les données utiles de la trame passée en argument et la méthode **" friend " operator<< (out :ostream&, infos : const InfosJournal&):ostream&** insère ces données dans le flot en respectant le format **HH:MM:SS,PPPP,111111111,222222222,...**.

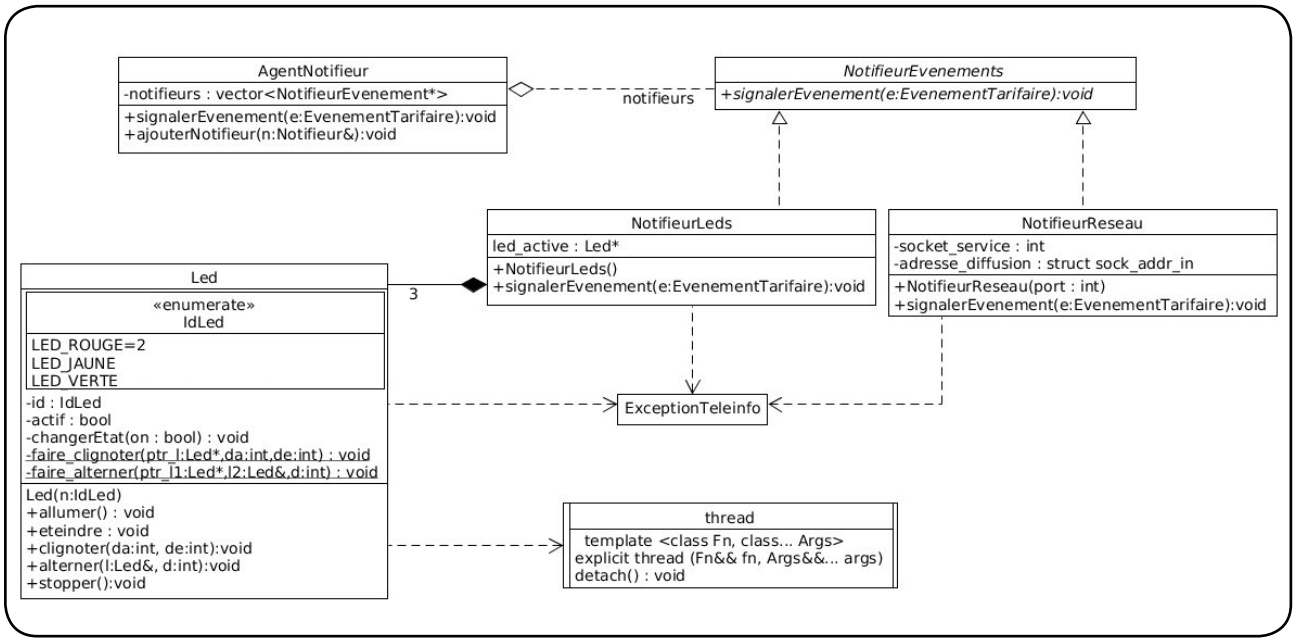


Figure 11 : Les classes du modules « service ».

2.3 Le module « service »

Ce module regroupe les classes collaborant à la signalisation des événements tarifaires ; ces classes sont représentées sur la figure 11.

2.3.1 La signalisation des événements tarifaires

Afin de signaler les événements tarifaires, la classe **`AgentNotifieur`** maintient une liste d'instances de classes implémentant l'interface **`NotifieurEvenement`**. Le constructeur par défaut crée une liste vide qui peut être remplie grâce à la méthode **`ajouterNotifieur()`**.

La classe **`NotifieurReseau`** implémente la signalisation selon le protocole UDP en diffusant des datagrammes sur le port fourni en argument de son constructeur. L'adresse de diffusion sera de la forme **`192.168.x.255`**. Les datagrammes contiendront un texte de la forme **`"EDF : <label événement>"`** où **`<label événement>`** est un texte défini dans la classe **`EvenementTarifaire`** du module « teleinfo ».

Le constructeur de la classe **`NotifieurLeds`** crée trois instances de la classe **`Led`** identifiées par la référence de la broche GPIO (2, 3 ou 4) à laquelle elles sont connectées. Le pointeur **`led_active`** permet de maintenir un lien sur l'unique led active à un instant donné (s'il est non nul) afin de pouvoir stopper son clignotement.

2.3.2 Le mécanisme de clignotement des leds

Le constructeur de la classe **`Leds`** effectue les initialisations nécessaires à l'utilisation des GPIO en sortie.

On peut allumer/éteindre une led grâce aux méthodes éponymes qui font appel à la méthode **`changerEtat(on :bool)`** avec l'argument **`true`**/**`false`**.

Pour le clignotement d'une led, une instance de la classe **`thread`** est créée sur la méthode statique correspondante avec les paramètres adéquats concernant la durée ; cette méthode est alors exécutée en parallèle avec le *thread* principal de l'application (voir méthode **`thread::detach()`**). Ainsi la méthode **`clignoter(da:int,de:int)`** appelle le constructeur de la classe **`thread`** de la manière suivante : **`thread(faire_clignoter,this,da,de)`**, **`da`** indiquant la durée allumée en millisecondes et **`de`** la durée éteinte. La méthode **`faire_clignoter(Led*,int,int)`** alterne alors les états allumé/éteint en respectant les durées (voir **`this_thread::sleep_for()`**) tant que le booléen actif (positionné à **`true`** avant démarrage du *thread*) reste à **`true`**. C'est la méthode **`stopper()`** qui arrêtera le clignotement en positionnant le booléen à **`false`**, terminant ainsi l'exécution du *thread*.

Le clignotement alternatif de deux leds est obtenu de manière analogue.

3 Conception du service web

La figure 12 rappelle le diagramme de séquence système retenu pour le cas « Consulter télé-information ». Voyons comment réaliser ces opérations.

L'utilisateur effectuera ses requêtes au moyen d'un navigateur internet. Nous mettrons donc en place un serveur HTTP sur le Raspberry Pi du système de télé-information. Ce serveur doit pouvoir fournir, outre des pages HTML, les données de la consommation électrique (trame courante, historique) à présenter à l'utilisateur. Afin de minimiser la charge sur le serveur, nous utiliserons des pages HTML avec une mise à jour dynamique des champs de données réalisée en **JavaScript** avec la méthode **AJAX**. Ceci nous permettra de réaliser un serveur HTTP élémentaire en **Python**. La figure 13 illustre ce mécanisme dans le cas d'une requête de consommation (données de la trame courante).

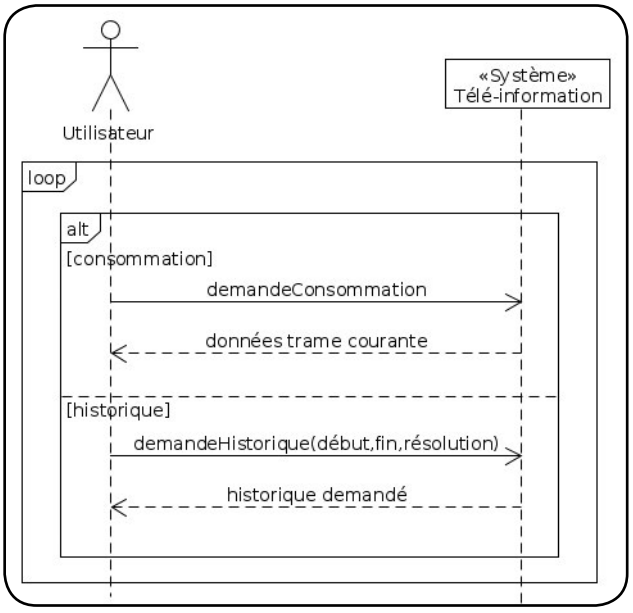


Figure 12 : DSS du cas « Consulter télé-information ».

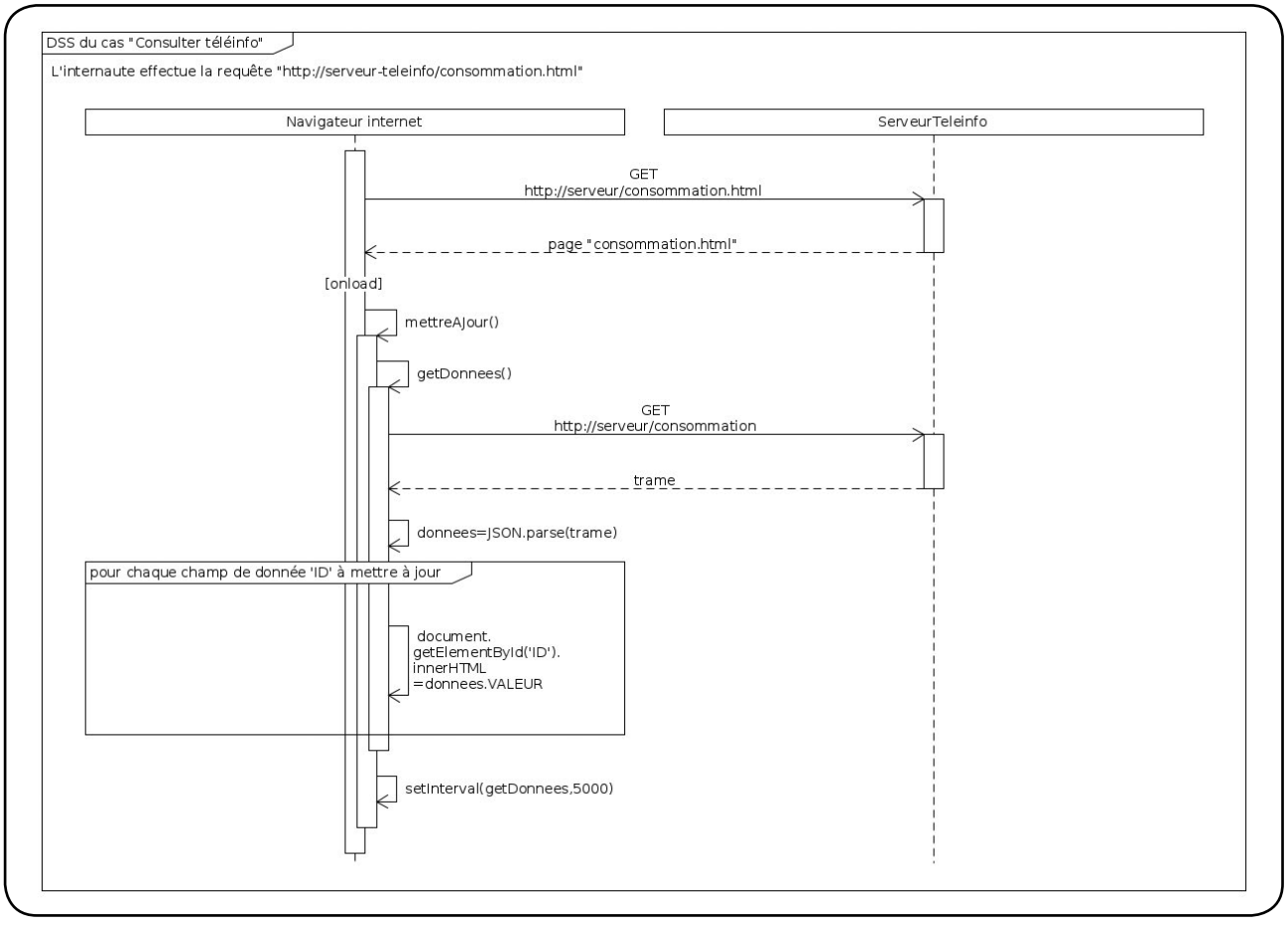


Figure 13 : Exécution d'une requête de consommation.

Ceci nous fournit le cadre JavaScript de la page **consommation.html**. L'attribut **onload** de la balise **<body>** de la page déclenche l'exécution de la méthode **mettraAjour()** qui appelle la méthode **getDonnees()** à intervalle régulier (ici 5 secondes) ; celle-ci met à jour les champs de données identifiés dans la page avec les données au format json extraites de la trame obtenue au moyen d'une requête AJAX.

La requête concernant l'historique utilisera le même mécanisme avec des requêtes AJAX de la forme :

```
http://192.168.x.y:5000/historique?d=2016/4/22&f=2016/4/24&r=30
```

Celle-ci demande l'historique du **22/4/2016** au **24/4/2016** avec une résolution de **30** minutes et fournit une réponse au format json :

```
[{"date":"2016/04/22","donnees":[{"H":"00:00:00","P":"HN..",
"I":["248129338","008597000"]},
 ... ["248335175","008597000"]}]}]
```

Il s'agit d'un tableau de données journalières qui fournit pour chaque date les données correspondant à chaque intervalle de temps : l'heure, la période tarifaire et les index.

La page web de l'historique nécessite un code JavaScript plus élaboré afin de construire un graphique (*canvas* HTML) à partir des données fournies (voir figures 14 et 15).

La réalisation du serveur utilise la bibliothèque **WSGI** ; on crée une instance de la classe **HTTPServer** (voir diagramme de classes en figure 16) au moyen de **wsgiref.simple_server.make_server(host,port,app)** où **app** désigne un objet application, c'est-à-dire une instance d'une classe implémentant une méthode **__call__** (c'est le cas de notre classe **Controleur**) chargée de gérer les requêtes.

Fig. 14 : Page de la consommation électrique.

On crée une instance de la classe **Historique** pointant sur le répertoire des données de télé-information et une instance de la classe **Controleur** pointant sur le répertoire racine contenant les pages HTML et associée à l'historique.

La méthode **__call__** du contrôleur récupère en argument un dictionnaire des variables d'environnement ainsi que la fonction **start_response(ligne_etat, entetes)** qui doit être utilisée pour préparer la réponse à la requête ; elle doit fournir en retour le corps de la réponse. En exploitant la variable d'environnement **PATH_INFO** pour analyser le texte de la requête, on peut déléguer le travail à l'une des méthodes permettant de servir les données de télé-information ou un fichier du site (page HTML, image, etc.).

L'implémentation des méthodes de la classe **Historique** est réalisée en se référant à l'étude présentée en section 2.2.

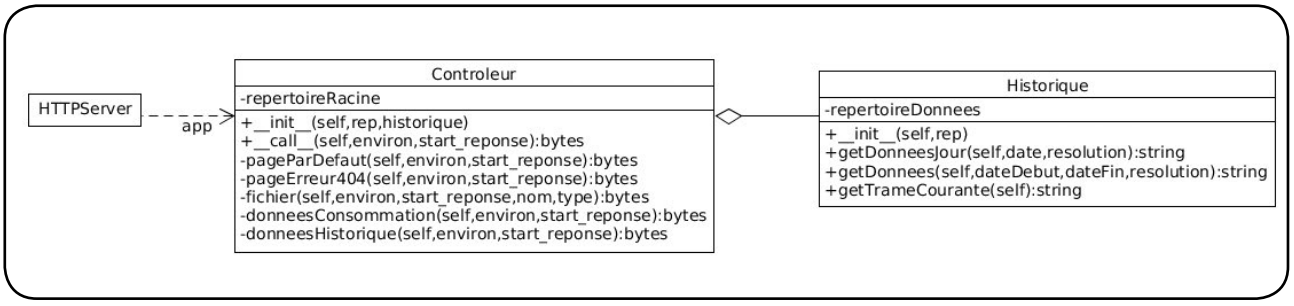


Figure 16 : Diagramme de classes du serveur HTTP.

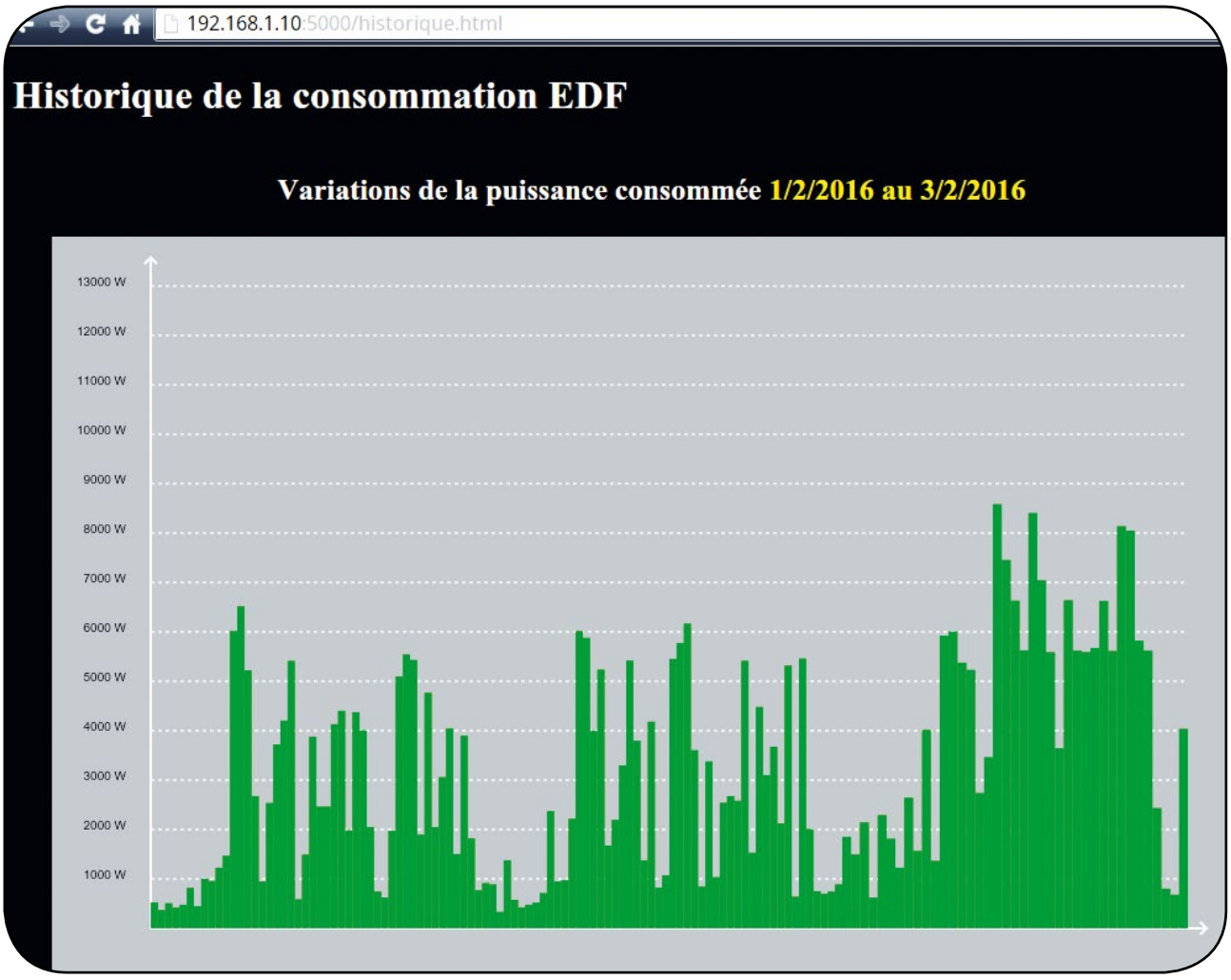


Fig. 15 : Page d'historique de la consommation.

Conclusion

Cet article nous a permis d'illustrer le *processus de conception* d'une application depuis sa spécification par les cas d'utilisation ; nous en sommes arrivés à l'étape du *codage*. La réalisation du service de télé-information devra se poursuivre par la définition des fichiers entêtes C++ qui se déduisent aisément des diagrammes de classes et par l'implémentation des méthodes décrites dans cet article ; un fichier `Makefile` pourra faciliter la compilation et l'édition de liens. La réalisation en Python du service web ne pose pas de problème particulier.

Il restera alors à installer ces services dans le système afin qu'ils soient lancés automatiquement ; on pourra au choix préparer deux scripts shell de style `init` ou deux descriptions de services pour **systemd** suivant la version de **Raspbian** utilisée. ■

Références

[1] M. Rambouillet , *« Modélisation d'un système de télé-information EDF »*, *GNU/Linux Magazine n° 176*, septembre 2016

[2] P. Roques, *« Les cahiers du programmeur UML 2 - 4ème Édition »*, Eyrolles, 2006

[3] *The Unified Modeling Language* : http://www.uml-diagrams.org

[4] Sorties de télé-information client des appareils de comptage électroniques utilisés par ERDF : http://www.erdf.fr/sites/default/files/ERDF-NOI-CPT_02E.pdf

[5] Documentation de la librairie GPIO WiringPi : http://wiringpi.com

PHARO ET LES BASES DE DONNÉES RELATIONNELLES

Olivier AUVERLOT [Membre de l'équipe RMoD, Laboratoire CRIStAL de l'université Lille 1]
Stéphane DUCASSE [Responsable de l'équipe INRIA RMoD, Laboratoire CRIStAL de l'université Lille 1]

Utilisé majoritairement dans le cadre d'applications web, Pharo a pourtant longtemps fait preuve de faiblesses dans le domaine de l'accès aux bases de données relationnelles. Heureusement, la situation a favorablement évolué avec l'arrivée d'un excellent framework nommé Garage.

Mots-clés : Pharo,Garage,SGBD,SQL,PostgreSQL

Résumé

Garage est un outil fédérateur. Il a pour mission de fournir un accès unifié aux solutions préexistantes. Avant lui, Pharo disposait certes de différents clients lui permettant de communiquer avec les principaux SGBD du marché, mais chaque implémentation avait sa propre logique d'utilisation. Tout cela manquait de cohérence. Pour les développeurs, la situation n'était pas très confortable. Il était donc nécessaire de faire un peu de ménage et c'est ce que Guillermo Polito a fait avec Garage. Ce *framework* facilite également l'installation des différents protocoles clients disponibles pour Pharo, qu'ils soient basés sur une librairie dynamique ou qu'ils soient écrits entièrement en Pharo.

Avant de partir à la découverte de Garage **[1]**, il vous faut bien évidemment installer Pharo 4.0 sur votre machine. Pour cela, téléchargez l'archive adaptée à votre distribution à partir du site officiel de Pharo **[2]**. Une fois celle-ci décompactée, lancez l'exécutable `Pharo`. Faites un clic droit sur le bureau et sélectionnez le ***Configuration Browser*** dans le menu ***Tools***. Sélectionnez ***Garage*** et cliquez sur ***Install Stable Version***. Après quelques secondes d'attente, vous êtes prêt à lancer vos requêtes SQL vers votre base de données préférée.

1 Compatibilité de bases de données

Pour vérifier les pilotes disponibles, il vous suffit d'exécuter dans le *Playground* le code suivant :

```
GADriver availableDrivers
```

Par défaut, Garage fournit un pilote pour **SQLite**, **MySQL**, **PostgreSQL** ainsi que pour **OpenDBX [3]** qui est un projet libre dont la finalité est de fournir des pilotes natifs vers un grand nombre de SGBD tels que **Sybase** ou **Oracle**.

Au sein de vos applications, il vous est également possible de vérifier qu'un pilote précis est bien chargé. Dans l'exemple suivant, la méthode `isAvailable` s'assure que le pilote pour PostgreSQL est disponible en retournant une valeur booléenne. Essayez-la en saisissant dans votre *Playground* le code suivant :

```
GADriver isAvailable: #postgresv2
```

2 Construire une base de test

Pour les différents exemples de cet article, vous allez utiliser une toute petite base de données et celle-ci fonctionnera sous PostgreSQL. Garage supporte plusieurs autres SGBD et il vous sera extrêmement simple d'adapter la base à l'outil de votre choix.

Pour cet exemple, il s'agit d'une simple table destinée à gérer une collection de disques.

```
$ createdb -U olivier -E UTF8 mescd
--template=template0
```

La structure est bien évidemment volontairement simplifiée, car il s'agit d'un article sur Garage et non pas sur les bases de données. Il n'y a donc qu'une seule table dans la base.

```
CREATE TABLE disques (
    cle serial,
    titre character varying(255) NOT NULL,
    artiste character varying(255) NOT NULL,
    annee integer NOT NULL CHECK (annee >
1900 AND annee < 2050)
);
ALTER TABLE ONLY disques
    ADD CONSTRAINT disques_pkey PRIMARY KEY
(cle);
```

Nous avons maintenant une base de données et un schéma prêt à recevoir nos données. Passons à l'étape suivante, mais cette fois-ci, avec Pharo !

3 Connexion à la base de données

Dans Pharo, ouvrez le *System Browser* et créez un paquet nommé **MesCD**. Dans celui-ci, ajoutez maintenant une classe **MesCD** qui contiendra les différentes méthodes décrites dans cet article.

Commençons par la méthode **MesCDConnectionString** qui définit les paramètres de connexion à la base PostgreSQL à l'aide d'une simple URL.

```
MesCD >> MesCDConnectionString
  ^'postgresV2://localhost:5432/mescd?user=olivier&password=motdepasse'
```

À chaque type de base de données, Garage associe un identifiant pour le protocole. Le tableau suivant vous permettra d'indiquer le bon protocole en fonction du SGBD que vous utilisez.

Base de données	Identifiant du protocole
PostgreSQL	postgresV2
MySQL	mysql
SQLite	sqlite3
Interface OpenDBX	opendbx

Pour conserver la connexion vers la base de données, vous avez besoin d'une variable d'instance. Pour cela, modifiez la définition de la classe **MesCD** en ajoutant une variable d'instance nommée **connection**.

```
Object subclass: #MesCD
  instanceVariableNames: 'connection'
  classVariableNames: ''
  category: 'MesCD'
```

Cette variable n'étant pas utilisée en dehors de la classe **MesCD**, il n'est pas utile de générer des accesseurs.

L'étape suivante consiste à définir la méthode **initialize** qui sera exécutée à chaque nouvelle instanciation de la classe **MesCD**.

```
MesCD >> initialize
  super initialize.
  connection := GADriver fromConnectionString: self MesCDConnectionString.
        connection connect
```

Vous pouvez également ajouter dès maintenant une méthode déconnectant votre application du serveur de base de données.

```
MesCD >> disconnect
  connection close
```

Pour tester ces premières étapes, exécutez directement dans le *Playground* le code suivant qui déclenche une connexion à la base **MesCD** suivie d'une déconnexion.

```
MesCD new disconnect
```

Dans le monde réel, les connexions aux bases de données (et aux serveurs en général) ne se passent pas toujours si bien. Il peut arriver que pour différentes raisons (coupures réseau, serveur saturé, pleine lune, etc.) votre client soit incapable d'établir une connexion vers le SGBD. Il est donc important de vérifier que tout s'est bien passé avant de poursuivre les opérations.

Pour cela, ajoutez une exception dans le paquet **MesCD**. Elle aura pour but de signaler un problème de connexion à la base de données. Une exception est une sous-classe de la classe **Error** et le texte expliquant l'erreur est retourné par la méthode **description**.

```
Error subclass: #MesCDConnectionException
  instanceVariableNames: ''
  classVariableNames: ''
  category: 'MesCD'

MesCDConnectionException >> description

  ^ 'Not Connected!'
```

Il faut ensuite modifier la méthode **initialize** pour qu'elle s'assure de la validité de la connexion. En cas d'échec, la méthode déclenche l'exception pour signaler le problème et stopper l'exécution du code.

```
MesCD >> initialize
  super initialize.
  connection := GADriver fromConnectionString: self
MesCDConnectionString.
  connection connect.
  connection isConnected ifFalse: [ MesCDConnectionException
signal ]
```

4 Manipuler les données avec SQL

Nous allons maintenant charger l'unique table de notre base avec quelques bons disques. Pour éviter le désordre, créez un protocole **actions** dans la classe **MesCD** afin d'y placer les différentes méthodes que nous allons définir pour manipuler le contenu de la base.

La méthode **chargeTableDisques** insère trois lignes dans la table disques.

```
MesCD >> chargeTableDisques
  connection execute: '
  insert into disques(titre,artiste,annee) values(''Crawford
Street'',''Josh Woodward'',2009);
  insert into disques(titre,artiste,annee) values(''For
tomorrow'',''Emerald Park'',2010);
  insert into disques(titre,artiste,annee) values(''We won''''t
ever be saved here'',''Lie Craze'',2015);'
```

Exécutez le code suivant dans le *Playground* de Pharo pour tester votre nouvelle méthode :

```
MesCD new chargeTableDisques disconnect
```

C'est facile à faire, mais le résultat n'est pas très satisfaisant puisque nous sommes obligés de protéger les simples quotes en les doublant. Ce qui rend le code fort peu lisible et complique forcément l'écriture de requêtes plus complexes. Une autre approche consiste à utiliser la méthode **format:** qui s'applique sur un object **String**. Cette méthode permet d'effectuer des substitutions au sein d'une chaîne de caractères. Au passage et puisque nous utilisons Pharo, utilisons un tableau de tableaux sur lequel nous faisons autant d'itérations qu'il y a de disques.

```
MesCD >> chargeTableDisques
  #(('Crawford Street' 'Josh Woodward' 2009)
  ('For tomorrow' 'Emerald Park' 2010)
  ('We won''''t ever be saved here' 'Lie Craze' 2015)) do: [
    :disque |
    connection execute: ('insert into disques(titre,artiste,annee)
values(''{1}'',''{2}'',{3});' format: { disque first. disque
second. disque third })
  ].
```

Oops ! une petite erreur s'est glissée dans les données que nous venons d'insérer. L'excellent album de Josh Woodward n'est pas paru en 2009, mais en 2010. Utilisons la commande **UPDATE** de SQL pour rectifier notre erreur.

```
MesCD >> corrigeErreurDate
  connection execute: ('UPDATE disques SET annee= 2010 WHERE
titre = ''{1}''' format: #( 'Crawford Street'))
```

Ici encore, exécutez le code suivant pour tester votre méthode :

```
MesCD new corrigeErreurDate disconnect
```

Lorsque vous modifiez le contenu d'une table, vous avez la possibilité d'obtenir le nombre de lignes concernées par votre opération. Ce nombre est accessible en appelant la méthode **affectedRows** sur l'instance de **GAResult** retournée par la méthode **execute** de Garage. Vérifions ceci en effaçant le contenu de la table **disque** à l'aide de la commande **DELETE** de SQL. Le nombre de lignes sera affiché à l'aide d'une popup surgissant en bas à gauche de l'environnement Pharo.

```
MesCD >> effaceContenuTable
  | result |
  result := connection execute: 'DELETE FROM disques'.
  UIManager default inform: ('{1} lignes effacées' format: {
result affectedRows })
```

À ce stade, nous avons déjà les notions essentielles concernant l'utilisation de bases de données relationnelles avec Pharo. Continuons notre exploration en sécurisant les opérations exécutées à l'aide des transactions.

5 Utiliser des transactions

Une transaction permet d'exécuter un groupe de requêtes SQL en s'assurant que l'intégralité des opérations a réussi ou que l'ensemble du processus a été annulé sans modification des données. L'exemple classique utilisé pour expliquer l'utilité des transactions est celui du transfert d'argent entre deux comptes bancaires. Le prélèvement d'un certain montant à partir du compte **A** et l'ajout du même montant sur le compte **B** nécessitent deux opérations SQL. Ces deux opérations doivent obligatoirement réussir et si un problème survient durant leur exécution, les deux comptes bancaires doivent récupérer le même état que celui précédent le début du transfert. Pour réussir cela, il faut que les deux requêtes SQL soient exécutées au sein d'une transaction.

```
begin;
update comptes SET montant = (montant - 10) WHERE comte = 'A';
update comptes SET montant = (montant + 10) WHERE compte = 'B';
commit;
```

Essayons de faire une transaction sur notre base musicale. Pour cela, nous allons modifier la méthode **ChargeTableDisques** afin de nous assurer que l'ensemble des insertions a bien été réalisé. En cas d'erreur, tout sera annulé. Pour indiquer le début d'une transaction, Garage dispose de la méthode **beginTransaction**. Pour valider ou annuler une transaction, il vous faut utiliser les méthodes **commitTransaction** et **rollbackTransaction**.

```
MesCD >> chargeTableDisques
  #(('Crawford Street' 'Josh Woodward' 2010)
  ('For tomorrow' 'Emerald Park' 2010)
  ('We won''''t ever be saved here' 'Lie Craze' 2015)) do: [
    :disque |
    [
      connection beginTransaction.
      connection execute:
                              ('insert into
disques(titre,artiste,annee) values(''{1}'',''{2}'',{3});'
                              format: { disque first. disque
second. disque third }).
      connection commitTransaction.
    ] on: Error do: [ connection rollbackTransaction ]
  ]
```

Nous avons fait le tour des principales commandes SQL permettant de modifier le contenu d'une base de données. Nous pouvons maintenant l'interroger afin d'en extraire des informations.

6 Faire des requêtes sur la base

Pour l'instant, nous avons essentiellement manipulé les données présentes dans la base. Il nous faut également être capables de les lire. Pour cela, il suffit d'exécuter des requêtes SQL contenant l'instruction **SELECT**. Le résultat est une instance de la classe **GAResult**, contenant l'ensemble des lignes lues dans la base de données.

Étant donné que nous allons essayer les différentes possibilités de Garage, nous allons tout d'abord créer une méthode **disques** retournant le contenu de la table **disques**. Ceci nous évitera de ressaisir plusieurs fois cette ligne de code.

```
MesCD >> tableDisques
  ^connection execute: 'SELECT * FROM disques'
```

La méthode **lireTableDisques** lit l'ensemble des enregistrements présents dans la table **disques**. Le contenu de chaque ligne est affiché dans le *Transcript*.

```
MesCD >> lireTableDisques
  self tableDisques do: [ :row | self printCD: row ]
```

La méthode **printCD:** est une méthode utilitaire qui formate le résultat avant de l'afficher.

```
MesCD >> printCD: aRow
  | disque |
  disque := WriteStream on: String new.
  disque
    nextPutAll: (aRow atName: #titre);
    nextPutAll: ' / ';
    nextPutAll: (aRow atName: #artiste);
    nextPutAll: ' / ';
    nextPutAll: (aRow atName: #annee) asString.
  Transcript show: disque contents; cr.
```

Dans le menu ***Tools***, ouvrez la fenêtre *Transcript* si celle-ci n'est pas encore à l'écran et saisissez le code suivant dans le *Playground* :

```
MesCD new lireTableDisques; disconnect
```

Vous avez probablement remarqué que la méthode **lireTableDisques** n'utilise pas une boucle classique pour

lire les différentes lignes obtenues. Elle utilise la méthode **do:** qui autorise le parcours d'une collection. C'est une approche bien plus dans l'esprit **Smalltalk**. Pour Garage, un résultat est donc simplement une collection de lignes qui sont ici des instances de la classe **GAAsciiRow**.

Au sein de la méthode **printCD:**, la méthode **atName:** permet d'accéder au contenu d'une colonne en fournissant son nom comme paramètre. Il est également possible d'utiliser l'index de la colonne à l'aide de la méthode **atIndex:**. Garage autorise également une certaine forme de méta-programmation puisqu'il est possible d'obtenir la description des colonnes présentes dans le résultat. Il suffit pour cela d'utiliser la méthode **rowDescription**.

Et puisque **GAResult** est une collection, cela ouvre de nombreuses possibilités. Tout d'abord, il vous est possible d'accéder à chaque ligne lue dans la base en fonction de sa position. La première ligne lue peut être obtenue à l'aide de la méthode **first**. La méthode **at:** permet de lire une ligne en fonction de sa position. Avec un manque flagrant d'inspiration et sachant qu'il n'y a que trois disques dans la base de données, la méthode **lireTableDisques** aurait pu ainsi être très mal écrite de la manière suivante :

```
MesCD >> lireTableDisquesMoche
  | data |
  data := self tableDisques.
  data isEmpty ifFalse: [
    1 to: 3 do: [ :i | self printCD: (data at: i) ] ]
```

Nous avons également la possibilité d'utiliser d'autres méthodes s'appliquant à une collection telles que **collect:** et **select:**. Ces deux méthodes vont vous permettre de filtrer le résultat obtenu ou de produire de l'information à partir du résultat. La méthode **collect:** permet d'appliquer une transformation aux données. Pour chaque élément de la collection constituant le résultat, la méthode **collect:** exécute un bloc de code destiné à construire une autre collection. Dans l'exemple ci-dessous, nous cherchons le nombre de disques publiés en 2015. La méthode **collect:** retourne alors une collection de valeurs booléennes. Il suffit ensuite de comptabiliser celles ayant la valeur **true** :

```
MesCD >> lireTableDisquesAnnee2015
  | nbr |
  nbr := (self tableDisques collect: [ :row | (row atName: #annee)
= 2015 ]) occurrencesOf: true.
  Transcript show: (nbr asString) , ' CD de 2015 dans la base'.
```

Avec **select:**, vous avez la possibilité de construire une nouvelle collection regroupant les éléments répondants aux critères évalués dans le bloc. Dans l'exemple ci-dessous, la méthode ne conserve que les disques ayant été publiés jusqu'en 2010 :

```
MesCD >> lireTableDisquesJusque2010
  (self tableDisques select: [ :row | (row atName: #annee) <= 2010 ])
    do: [ :row | self printCD: row ]
```

Attention, ce traitement sera fait sur le client et il est en général préférable de confier la sélection des données au serveur de base de données en écrivant des requêtes SQL dans lesquelles sont définies des conditions.

7 Instructions préparées

Les instructions préparées (*Prepared Statements*) permettent d'exécuter de manière optimisée des requêtes SQL. Celles-ci sont compilées et mises en cache par le serveur de base de données avant d'être utilisées. Elles sont également paramétrables. Les valeurs des paramètres sont transmises par le client avant chaque exécution. En dehors de l'aspect performance, les instructions préparées sont également un bouclier contre les tentatives d'injection de code SQL et contribuent donc à la sécurisation d'une application.

La méthode **lireTableDisquesPourAnnee:** permet d'extraire de la base l'ensemble des disques publiés durant l'année indiquée en paramètre. L'instruction préparée est construite à l'aide de la méthode **prepare:** et est affectée à la variable locale **statement**. Le caractère **?** permet de définir au sein de la requête SQL des marqueurs. Grâce à ceux-ci, la méthode **bind:at:** sait où placer les valeurs correspondantes. Dans cet exemple, le premier point d'interrogation est remplacé par la valeur du paramètre contenant l'année. Une requête SQL peut bien évidemment contenir plusieurs points d'interrogation afin d'autoriser plusieurs substitutions. Si la valeur est une chaîne de caractères, le point d'interrogation doit être entouré de simples quotes.

```
MesCD >> lireTableDisquesPourAnnee: aYear
  | statement data |
  statement := connection prepare: 'SELECT * FROM disques WHERE
annee=?'.
  statement at: 1 bind: aYear.
  data := statement execute.
  data isEmpty ifFalse: [ data do: [ :row | self printCD: row ] ]
```

Pour obtenir la liste des disques publiés en 2015, saisissez et exécutez ce code dans votre *Playground* :

```
MesCD new lireTableDisquesPourAnnee: 2015; disconnect
```

Les instructions préparées ne sont pas toujours accessibles via Garage. Pour vous assurer que vous pouvez les utiliser, vous pouvez vérifier leur support à l'aide de la méthode **supportsPreparedStatements** qui retourne une valeur booléenne. La méthode **lireTableDisquesPourAnnee:** peut être réécrite de la manière suivante :

```
MesCD >> lireTableDisquesPourAnnee: aYear
  | statement data |
  connection supportsPreparedStatements
    ifTrue: [
      statement := connection prepare: 'SELECT * FROM disques
WHERE annee=?'.
      statement at: 1 bind: aYear.
      data := statement execute.
      data isEmpty ifFalse: [ data do: [ :row | self printCD: row
] ]
    ]
```

Cette méthode, tout comme celles que nous avons définies jusqu'à maintenant, récupère la réponse de la base de données et travaille ensuite en mémoire. Mais dans certains cas, le volume d'informations retournées ne permet pas de les charger et de les stocker en mémoire, il faut alors utiliser des curseurs.

8 Les curseurs

Les curseurs sont importants lorsque vous devez travailler avec une base de données volumineuse. Ils permettent d'économiser de la mémoire et de réduire le trafic réseau. Le principe de fonctionnement est simple puisqu'il s'agit pour l'application cliente de ne recevoir que ce dont elle a besoin. Lorsqu'une requête SQL est exécutée, le serveur de base de données fait parvenir les lignes par paquets et ceci à la demande du client.

Les différents pilotes de base de données disponibles dans Garage ne supportent pas forcément l'usage des curseurs. Il faudra donc s'en assurer avant de les utiliser à l'aide de la méthode **supportsCursoredFetch** qui retourne une valeur booléenne. Certains pilotes, tels que celui pour PostgreSQL, nécessitent également que les curseurs soient utilisés au sein d'une transaction.

Pour notre base de disques, imaginons que nous ne voulons recevoir que deux lignes à la fois et pas une de plus. Nous devons donc définir une requête SQL au sein d'une transaction et fixer à l'aide la méthode **fetchSize:**, le nombre de lignes retournées par la base de données à chaque fois que le client lui demande un résultat.

```
MesCD >> lireUneLigneAvecUnCurseur
  | statement readStream data |
  (connection supportsCursoredFetch) ifTrue: [
    connection beginTransaction.
    statement := connection createStatement: 'SELECT * FROM disques'.
    statement fetchSize: 2.
    readStream := statement execute readStream.
    [
      data := readStream next: 1.
      data isNotEmpty
    ] whileTrue: [ self printCD: data first ].
    connection commitTransaction.
  ]
```

Pour lire la liste des disques à l'aide d'un curseur, saisissez et exécutez ce code dans votre *Playground* :

```
MesCD new lireUneLigneAvecUnCurseur; disconnect
```

La lecture des données est basée sur un flux en lecture seule (**readStream**). Chaque paquet de lignes est contenu au sein d'un tableau (**Array**). Il suffit maintenant au client de consulter ligne par ligne ces paquets afin d'afficher leur contenu dans le *Transcript*. Lorsqu'un paquet est vide, cela signifie que la base de données n'a plus aucune information à transmettre au client.

Conclusion

Accéder aux bases de données relationnelles avec Pharo est extrêmement simple. Les fonctionnalités des principales bases de données sont parfaitement gérées. L'expressibilité du langage et la richesse du *framework* Garage permettent de développer des applications agiles, robustes et se basant sur une architecture n-tiers en un minimum de temps. Sachez également que Pharo dispose du *framework* **Glorp** permettant de faire un *mapping* objet-relationnel.

De jour en jour, Pharo est une solution de plus en plus efficace pour la conception d'applications professionnelles et, comme en témoignent les nombreuses *success-stories* **[4]** référencées sur le site du projet Pharo, de nombreuses entreprises font confiance à ce langage et à son écosystème. ■

Références

[1] Le site du *framework* Garage : https://guillep.github.io/DBXTalk/garage/index.html

[2] Le site du projet Pharo : http://pharo.org

[3] Le projet OpenDBX : http://www.linuxnetworks.de/doc/index.php/OpenDBX

[4] Une sélection de *success-stories* : http://pharo.org/success

ÉCRIRE UNE BIBLIOTHÈQUE PERFORMANTE POUR NODE.JS

Sylvain NAYROLLES [Développeur]

Il existe différents moyens de développer des bibliothèques performantes pour Node.js ; soit par la réalisation d'une extension native au travers d'une API C/C++, soit via le projet Emscripten utilisant asm.js, un sous-ensemble de JavaScript.

Mots-clés : natif, Node.js, node-gyp, c, c++, emscripten, asm.js

Résumé

Quand on parle de langage interprété, on se soucie rarement des performances. Mais il existe de nombreux cas où les ressources commencent à manquer, et il faut donc envisager d'utiliser d'autres moyens que la « simple » optimisation algorithmique quand la limite est intrinsèque au langage. Node.js permet la programmation de bibliothèques en C et C++, mais offre aussi le moyen d'exploiter asm.js, un sous-ensemble de JavaScript permettant de manipuler des concepts de programmation d'un bien plus bas niveau que ce qu'il manipule d'habitude, via emscripten, un *backend* LLVM.

Dans cet article, et au travers d'un exemple d'algorithme simple, nous allons étudier deux méthodes de développement de code dit « performant » pour Node.js. Ces deux méthodes peuvent être complémentaires, car elles n'adressent pas les mêmes problématiques. L'extension native sera de toute manière la plus performante, mais aura des contraintes lors d'un déploiement sur des plateformes non maîtrisées ; tandis que la seconde, asm.js, qui est un sous-ensemble de JavaScript, sera portable, mais le gain de performance sera nettement moins sensible, même si certaines littératures peuvent dire le contraire...

1 Extension native

Node.js offre la possibilité, comme en **Python**, de développer des extensions en C et C++. Pour ce faire, il fournit une API permettant de communiquer entre le monde JavaScript et le monde natif. Il permet aussi d'abstraire une grande partie des dépendances systèmes afin de rendre chaque extension la plus portable possible. De plus, node étant lui-même compilé avec des bibliothèques de façon statique, il est donc possible d'en tirer partie, comme par exemple si nous voulions utiliser **OpenSSL**.

Enfin, il existe un outil, **node-gyp**, permettant de facilement développer, compiler et distribuer une extension node. Ce dernier est développé par Nathan Rajlich, qui est aussi impliqué dans de nombreux projets node. Il est livré maintenant avec la dernière version de node.

Toutefois, il est fortement conseillé de mettre à jour node-gyp via npm :

```
$ sudo npm install -g node-gyp
```

Afin que le bon node-gyp soit dans le **$PATH**, il suffit de lancer la commande suivante :

```
$ sudo npm explore npm -g -- npm install
node-gyp@latest
```

1.1 Code

Nous allons créer un fichier **filter.cc** contenant notre algorithme :

```
#include <algorithm>
#include <stdint.h>

void Threshold(std::vector<uint32_t>& image,
uint32_t thresholdValue) {
  std::replace_if(image.begin(), image.
end(), [thresholdValue](uint32_t element) ->
uint32_t {
    return element < threshold_value;
  }, 0);
}
...
```

Puis, nous allons utiliser l'API node afin de réaliser le *wrapping* de notre fonction précédente :

```
...
#include <node.h>
#include <v8.h>
void Threshold_Wrapper(const v8::FunctionCa
llbackInfo<v8::Value>& args) {
  v8::Isolate* isolate = args.GetIsolate();

  std::vector<uint32_t> image;
  v8::Local<v8::Array> input = v8::Local<v8
::Array>::Cast(args[0]);

  for (uint32_t i = 0; i < input->Length();
i++) {
    image.push_back(input->Get(i)-
>Uint32Value());
  }
  uint32_t thresholdValue =
args[1]ÐUint32Value() ;

  Threshold(image, thresholdValue);

  v8::Local<v8::Array> result =
v8::Array::New(isolate);
  for (uint32_t i = 0; i < image.size();
i++) {
    result->Set(i, v8::Number::New(isolate,
image[i]));
  }

  args.GetReturnValue().Set(result);
}

void init(v8::Local<v8::Object> target) {
  NODE_SET_METHOD(target, "threshold",
threshold_Wrapper);
}

NODE_MODULE(filter, init);
```

Nous exposons la fonction **Threshold_Wrapper**, via le nom **threshold** :

```
NODE_SET_METHOD(target, "threshold",
Threshold_Wrapper);
```

Puis, nous déclarons notre module **filter** avec comme point d'entrée notre fonction d'initialisation.

Nous allons maintenant nous attarder sur la fonction **Threshold_Wrapper**. Cette dernière consiste essentiellement à convertir des types de données node vers des types C++ natifs. Nous aurions pu faire le choix d'appliquer notre algorithme directement sur les types v8 afin d'éviter au maximum les recopies (ô combien coûteuses), mais il nous est apparu intéressant de décorréler la partie purement native, de la fonction de *wrapping* propre à node. Ce comportement se retrouve souvent dans le cas où le programmeur veut faire un *binding* d'une bibliothèque existante.

L'objet **args** est notre interface d'entrée-sortie de notre fonction. C'est ici que nous allons pouvoir récupérer les paramètres d'appel (**args[0..n]**) ainsi que définir notre valeur de retour (**args.GetReturnValue().Set(result)**). Notre signature JavaScript correspond exactement à notre signature C++, car nous définissons deux paramètres :

- **arg[0]** étant notre tableau ;
- **arg[1]** est notre seuil :

```
std::vector<uint32_t> image;
v8::Local<v8::Array> input = v8::Local<v8
::Array>::Cast(args[0]);

// store parameter
for (uint32_t i = 0; i < input->Length();
i++) {
  image.push_back(input->Get(i)-
>Uint32Value());
}
uint32_t thresholdValue =
args[1]ÐUint32Value() ;
```

Une fois les valeurs rapatriées, nous appelons notre algorithme :

```
Threshold(image, thresholdValue);
```

Puis, nous exportons notre résultat via un nouveau tableau :

```
v8::Local<v8::Array> result =
v8::Array::New(isolate);
for (uint32_t i = 0; i < image.size(); i++)
{
  result->Set(i, v8::Number::New(isolate,
image[i]));
}

args.GetReturnValue().Set(result);
```

1.2 Compilation

Afin de générer notre *addon*, nous allons nous servir de node-gyp. Ce dernier nécessite la création d'un fichier descriptif de notre module se nommant **filter.gyp** et respectant le format JSON d'entrée suivant :

```
{
  'targets': [
    {
      'target_name': 'filter',
      'sources': [ 'filter.cc' ]
    }
  ]
}
```

Nous définissons ici le nom de notre module (**filter**) ainsi que l'ensemble des sources associées (**filter.cc**).

Il ne nous reste plus qu'à appeler **node-gyp** afin qu'il nous génère notre *addon* :

```
node-gyp configure build
```

Le résultat est sous **./build/Release/filter.node**. Il ne nous reste plus qu'à l'utiliser !

1.3 Exécution

Essayons un peu notre algorithme. Pour ce faire, rien de plus simple, nous allons le tester via le fichier JavaScript suivant :

```
const filter = require('./build/Release/
filter');
image = [ 34, 176, 178, 98 , ... ]
image = filter.threshold(image, 100);
console.log(image);
```

Et ça marche !

```
> [ 0, 176, 178, 0, ... ]
```

2 Asm.js

Une bibliothèque native comporte de nombreux avantages, surtout au niveau des gains de performances. Mais sa mise en œuvre est complexe, et nécessite une certaine expertise sur le contexte d'exécution (gestion mémoire, *multi-threading*, structure de données, communication avec le monde JavaScript, etc.). De plus, sa diffusion n'est pas évidente, car il n'existe pas de moyens simples de diffuser un programme compilé via npm. Python et pip proposent des mécanismes comme les *wheels*, permettant de fournir des binaires pré-compilés pour des cibles particulières, mais il n'existe aucune équivalence sous npm.

En JavaScript, il existe un sous-ensemble du langage permettant d'accéder à des structures de données, à typage statique, permettant un net accroissement des performances. Ces dernières sont regroupées sous la dénomination de asm.js. Elles ont été introduites avec l'avènement du **WebGL** et donc la nécessiter de manipuler des structures mémoires bas niveau. Eh oui, il ne faut pas oublier que Node.js est basé sur v8 qui est le moteur JavaScript du navigateur **Chrome** de **Google**.

Il existe de nombreux projets mettant en avant les qualités des implémentations de asm.js par les navigateurs. Certains jeux ont même été portés sur navigateurs de façon réellement bluffante **[1]**. Il existe même un portage de la **ScummVM [2]** (Monkey Island) ! Mais ces derniers n'ont pas été entièrement réécrits pour l'occasion et ont été générés par un logiciel : emscripten.

2.1 Emscripten

Asm.js n'est pas destiné à être utilisé directement. Emscripten est un *backend* LLVM permettant de générer du code JavaScript, utilisant asm.js, depuis n'importe quel type de langage.

Il va donc exploiter asm.js afin d'implémenter des concepts très connus des programmeurs C ou C++. Par exemple, il va simuler une zone de pile ainsi qu'une zone de tas. Il permet de manipuler des pointeurs dans chacune de ces zones etc. Pour toutes ces raisons, nous allons donc utiliser le *frontend* C de LLVM (CLANG), afin de réutiliser le code de la section précédente.

Nous allons donc tout d'abord installer emscripten :

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

Ce paquet va vous fournir l'ensemble des outils nécessaires à la conversion depuis un fichier C ou C++ vers une source JavaScript optimisée avec asm.js. Ceci implique LLVM lui-même, CLANG pour la partie *frontend* compatible C/C++ et le *backend* JavaScript. Il fournit deux exécutables : **emcc** pour le C et **em++** pour le C++.

CLANG n'est pas encore entièrement compatible avec le C++11, au contraire de GCC utilisé par node-gyp. Nous devons donc légèrement adapter notre source afin qu'il le devienne :

```
#include <stdint.h>

void Threshold(uint32_t* image, uint32_t
size, uint32_t threshold) {
  for(uint32_t i = 0 ; i < size ; i++) {
    if(image[i] < threshold) {
      image[i] = 0;
    }
  }
}
```

On va convertir cette source en JavaScript via l'appel à **emcc** en lui précisant d'exporter la fonction **Threshold** :

```
emcc filter.cpp -s EXPORTED_
FUNCTIONS="['Threshold']"
```

Il ne nous reste plus qu'à utiliser le module produit dans un fichier JavaScript :

```
var filter = require('./filter');

// use asm.js for my image
var image = new UInt32Array( [ 34, 176, 178,
98, ... ] );

// allocate in heap of filter module
var imagePtr = filter._malloc(image.length);
var imageHeap = new Uint32Array( filter.
HEAPU32.buffer, imagePtr, image.length);

// call filter
filter.threashold(imageHeap.byteOffset,
image.length, 100);

// back result
image = new Uint32ClampedArray(imageHeap.
buffer, imageHeap.byteOffset, image.length);

    console.log(image);
```

Ce code n'est pas le plus optimisé en terme de copies, mais décorrèle les données de la bibliothèque. Il est intéressant, car il a l'avantage de présenter les concepts intrinsèques à emscripten.

Historiquement, emscripten était utilisé pour convertir un programme en entier. Mais il existe de plus en plus de bibliothèques du monde natif (**zlib**, **sqlite**) implémentées en JavaScript via la formidable moulinette d'emscripten, pour qu'elle soit utilisée dans un navigateur, ou dans node lui-même.

Cette méthode présente l'avantage de produire du code JavaScript et donc permet de faciliter le processus de livraison. Mais il faut être conscient que le gain de performances sera bien moins sensible qu'avec un *addon* natif.

Conclusion

Le développement de bibliothèques performantes pour node est donc grandement facilité par un écosystème fleurissant et éprouvé. Les deux méthodes présentées dans cet article ont leurs avantages et inconvénients et doivent donc être employées avec intelligence selon ce que l'on veut faire. Mais on peut quand même se dire qu'il est possible de réaliser des programmes, ou tout du moins des parties, rapides et efficaces avec un minimum d'effort ! ■

Références

[1] Site du jeu Quake 3 en JavaScript : https://www.quakejs.com

[2] La ScummVM en asm.js : http://clb.demon.fi/html5scummvm/monkey/monkey.html

1ER ÉVÉNEMENT EUROPÉEN
LIBRE & OPEN SOURCE

îledeFrance

MAIRIE DE PARIS

EMPOWERING
OPEN INNOVATION
opensourcesummit.paris
#OSSPARIS16

PARIS OPEN SOURCE SUMMIT

16 & 17 NOVEMBRE 2016

DOCK PULLMAN
Plaine Saint-Denis

SPONSOR DIAMOND

Inria INVENTEURS DU MONDE NUMÉRIQUE

SPONSORS PLATINUM

alter way · Microsoft · Smile OPEN SOURCE SOLUTIONS

SPONSORS GOLD

BlueMind Messagerie & espaces collaboratifs · henix

SPONSORS SILVER

axelor ACCÉLÉRATEUR DE PERFORMANCE · centreon · cozy.io · redhat · XiVO open-minded telecom systems · YaZiba.net Powered by zimbra

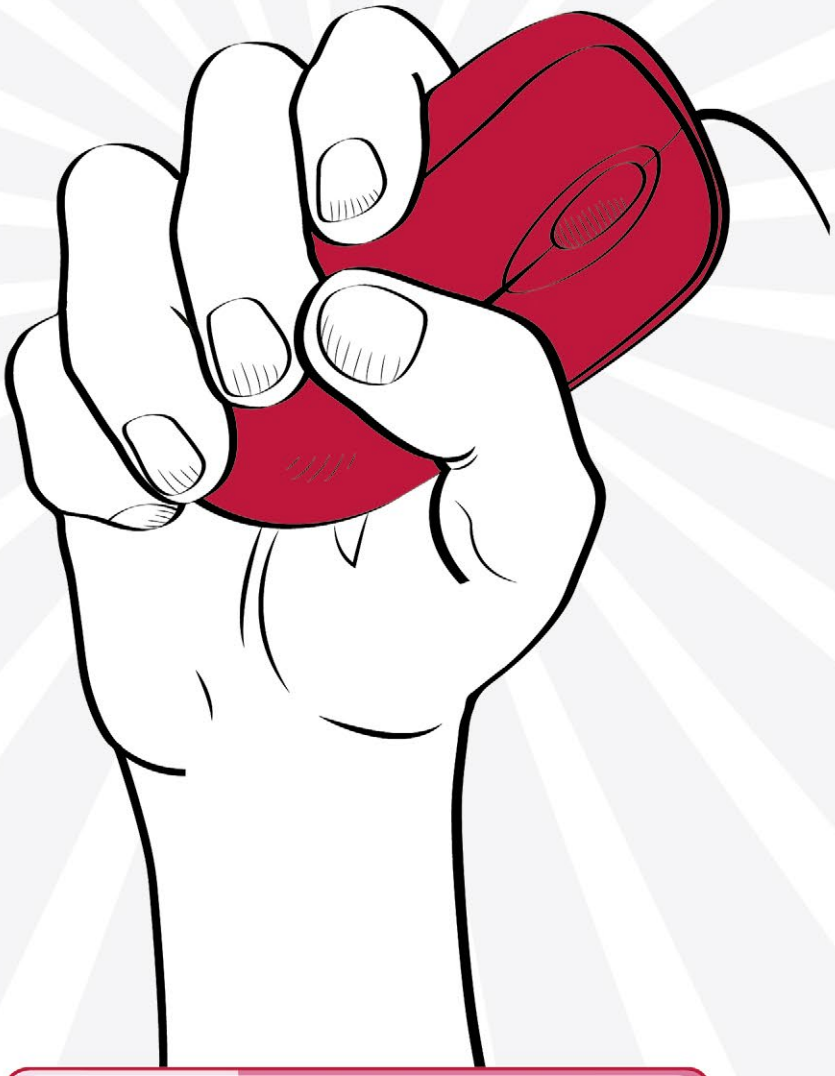
POUR TOUTE INFORMATION COMPLÉMENTAIRE :
Email : contact@opensourcesummit.paris – Tel : 01 41 18 60 52

un événement

Systematic Paris Region Digital Ecosystem · Tarsus FRANCE

Ne pas subir le numérique :

LE FAIRE



LINAGORA

Les logiciels libres

www.linagora.com