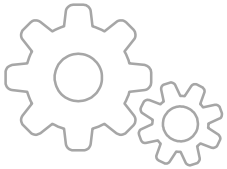
SUPRALOG
CONSEIL ▲ TECHNOLOGIE ▲ PROGICIEL



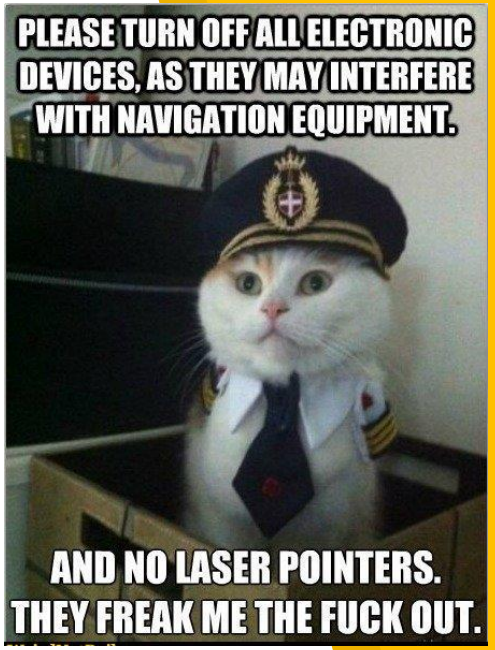
DSL INTERNES
DANS LA VRAIE VIE

Thomas Moreau
Software Architect @Supralog
tmoreau@supralog.com

09/01/2019

DSL INTERNES
AU PROGRAMME…

- tmoreau@supralog.com
- Définition
  - DSL interne vs DSL externe
- Retour d'expérience : HP Customer Care Dashboard
  - Pourquoi des DSLs
- Concevoir un DSL interne
  - Méthode
  - Exemple avec Groovy
- Mise en pratique : ArduinoML avec Groovy

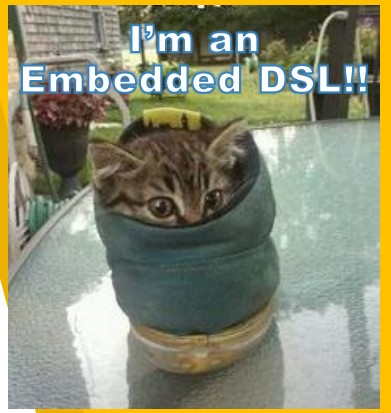


1.

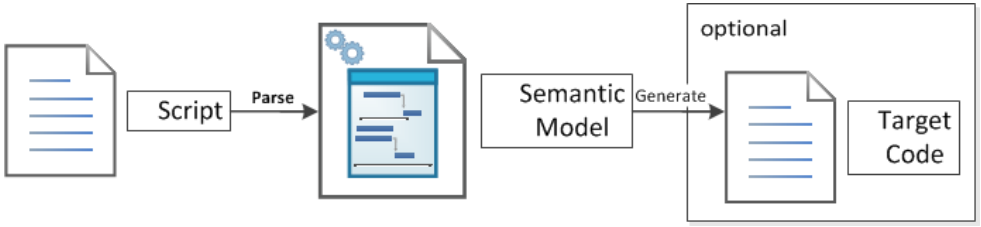
DEFINITION

DSL interne

DEFINITION
DSL INTERNE

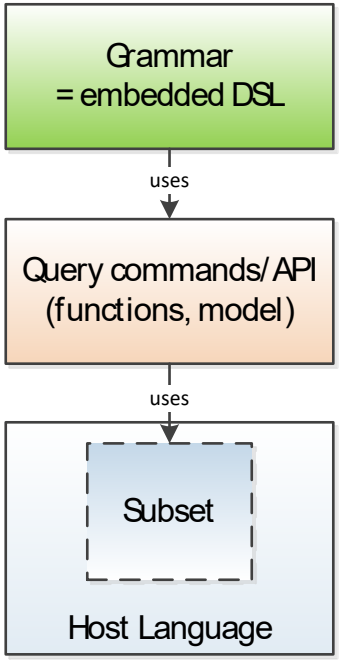


- DSL



- Un DSL interne est assez similaire à une API enjolivée « fluent interface »
- Un DSL interne est défini comme une librairie pour un langage hôte
- Un DSL interne hérite et exploite les fonctionnalités du langage hôte et en ajoute de nouvelles – spécifiques au domaine – créant un niveau d'abstraction supérieur

- DSL interne



DEFINITION
EXEMPLES & PATTERNS

- Exemples

```
Processor p = new Processor(2, 2500, Processor.Type.i386);
Disk d1 = new Disk(150, Disk.UNKNOWN_SPEED, null);
Disk d2 = new Disk(75, 7200, Disk.Interface.SATA);
return new Computer(p, d1, d2);
```

- Java **method chaining**

```
computer()
    .processor()
        .cores(2)
        .speed(2500)
        .i386()
    .disk()
        .size(150)
    .disk()
        .size(75)
        .speed(7200)
        .sata()
    .end();
```

Java **method sequence**

```
computer();
    processor();
        cores(2);
        speed(2500);
        i386();
    disk();
        size(150);
    disk();
        size(75);
        speed(7200);
        sata();
```

- Un schéma XSD peut être considéré comme un DSL interne
- Ruby (interne à Emacs Lisp)
- Lisp permet de faire facilement des DSLs internes
- (function arg1 … argn)
- … Python, Haskell, Javascript aussi

- Exploiter : pattern command, surcouche à une API / semantic model, fonctions, maps / lists, closures, annotations, … « tricks »

DEFINITION
EXEMPLES CONCRETS

- Jenkins Pipeline


```
pipeline {
  agent { docker 'maven:3-alpine' }
  stages {
    stage('Example Build') {
      steps {
        sh 'mvn -B clean verify'
      }
    }
  }
}
```

- Gatling Scenario


```
class D2siSimulation extends Simulation {
  val httpConf = http
    .baseURL("http://computer-database.gatling.io")
    .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8")

  val scn = scenario("D2siSimulation")
    .exec(http("Get Computers").get("/computers"))
    .pause(5)
    .exec(http("Add Computer").post("/computers")
    .formParam("name", "Beautiful Computer")
    .formParam("introduced", "2012-05-30")
    .formParam("discontinued", "")
    .formParam("company", "37"))

  setUp(scn.inject(atOnceUsers(100)).protocols(httpConf))
    .assertions(global.responseTime.max.lt(2000))
}
```

DEFINITION
DSL INTERNE vs DSL EXTERNE

- Coût de développement
  - Facilité à appréhender, développer et à utiliser. Sorte d'API enjolivée VS grammaire + parseur + …
  - Un DSL interne repose souvent sur des utilisations de subtilités "cachées" du langage hôte pour parvenir à façonner la syntaxe souhaitée
  - Maintenabilité, stabilité, apprentissage, utilisation
  - Pas besoin de maintenir l'interpréteur d'un DSL interne
  - profiter de l'infrastructure et des outils de développement du langage hôte

- Adaptation au domaine
  - Un DSL interne est limité par les possibilités du langage hôte
  - La flexibilité d'un DSL interne dépend du langage hôte sélectionné, celle d'un DSL externe dépend du parseur développé

	Interne	Externe
	+	-
	+/- (dépend du langage hote)	+
	+	-
	+	-
	- (dépend du langage hote)	+

SUPRALOG
CONSEIL ▲ TECHNOLOGIE ▲ PROGICIEL

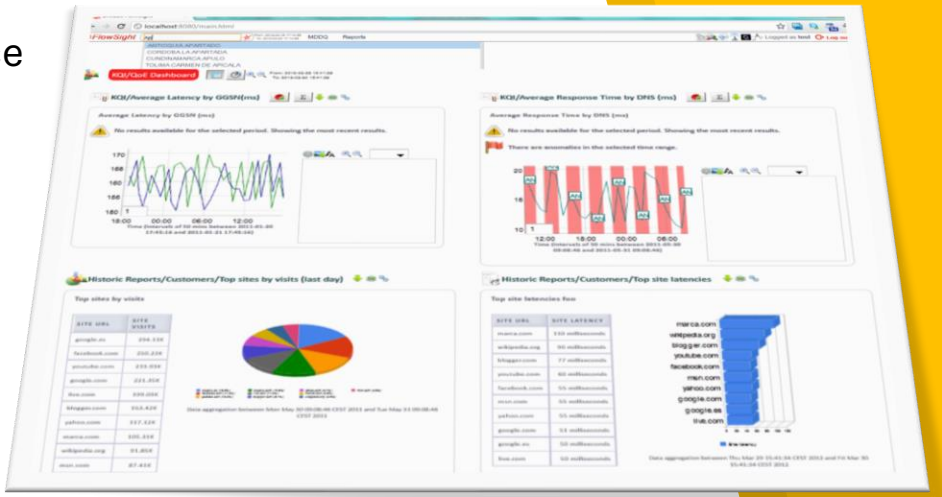
2.

RETOUR D'EXPERIENCE

HP Customer Care Dashboard

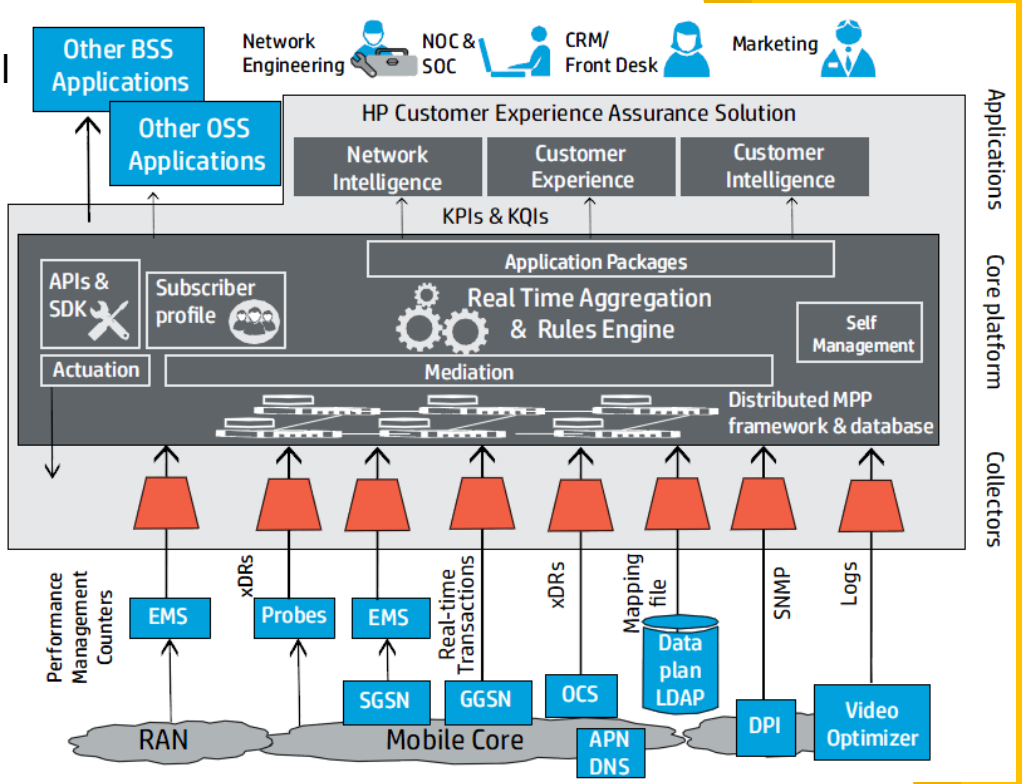
RETOUR D'EXPERIENCE
L'APPLICATION HP CEA

- HP Customer Experience Assurance solution
- Contexte "Big Data"
- Visualisation en temps réel de l'expérience
- utilisateur d'un client
- Détection d'anomalies
- "drill-down"



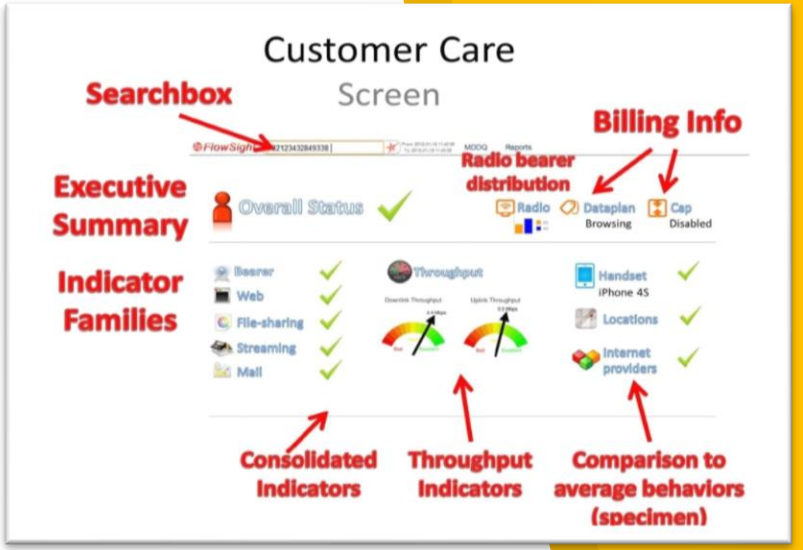
RETOUR D'EXPERIENCE
L'APPLICATION HP CEA

- Architecture MPP (Massive Parallel Processing)
- Bases de données orientées colonnes "Zstore"
- UI GWT



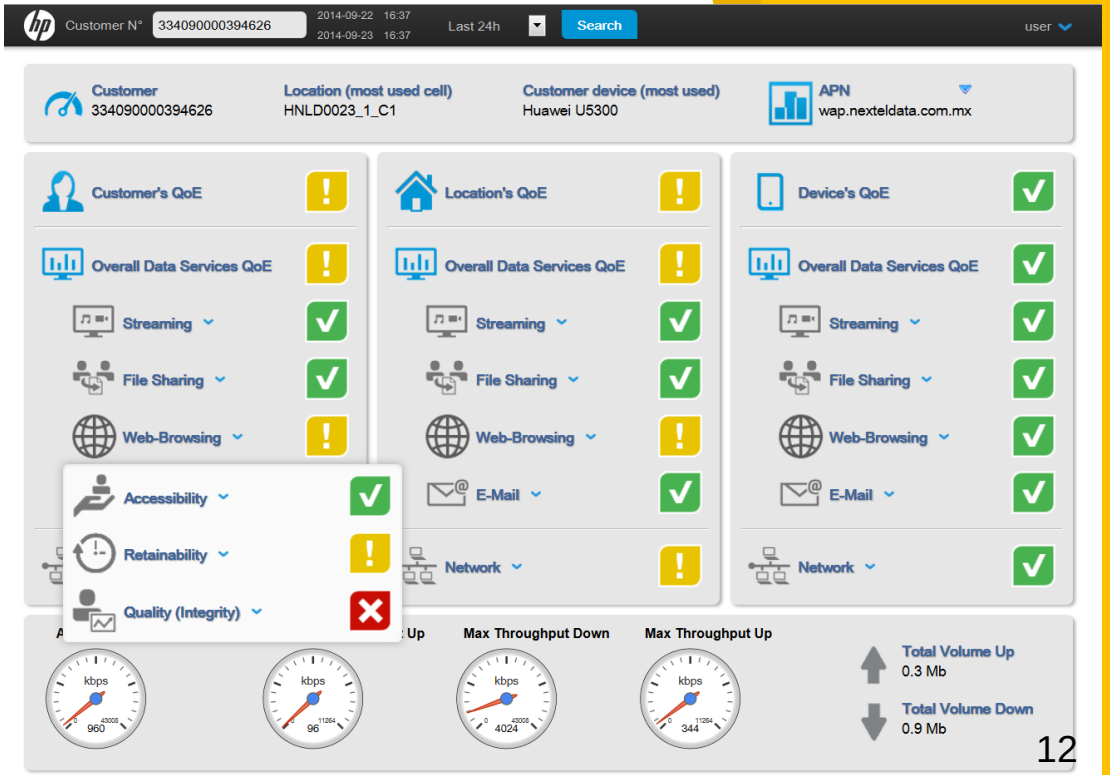
RETOUR D'EXPERIENCE
BESOINS

- Affichage d'indicateurs synthétiques de QoE
  - Utilisateur/abonné
  - Plage de temps
  - Customer Care Console
- Utilisateur: technicien de niveau 1
- Identification par numéro téléphonique/IMSI/IMEI
- Visualisations adaptées
  - Indicateur d'état, Jauges, Informations
- **Hautement configurable** (diversité des exigences des clients)
- Lien avec CEA
  - Data sources



RETOUR D'EXPERIENCE
HP CUSTOMER CARE DASHBOARD

- Liens avec CEA
- Surcouche de calculs, agrégations, optimisations (nombre de requêtes)
- Configurable



RETOUR D'EXPERIENCE
POURQUOI DES DSL

- Besoins
  - Pour qui ? **expert du domaine**, donc doit leur être accessible/compréhensible
  - Configurable à chaud sans besoin de connaissances en programmation particulière
  - Java (Google Web Toolkit)
  - Lisibilité, simplicité, compréhension rapide de la configuration (client, support, …)
  - Coût de développement vs Editeur graphique

- Deux configurations nécessaires
  - UI: agencement de widgets
  - Scripting: définition et traitement des données à afficher (calculs, formules, …)
  - Cette configuration de type "scripting" doit être extensible/flexible pour répondre à des besoins clients très spécifiques (sans délivrer de nouvelle version)

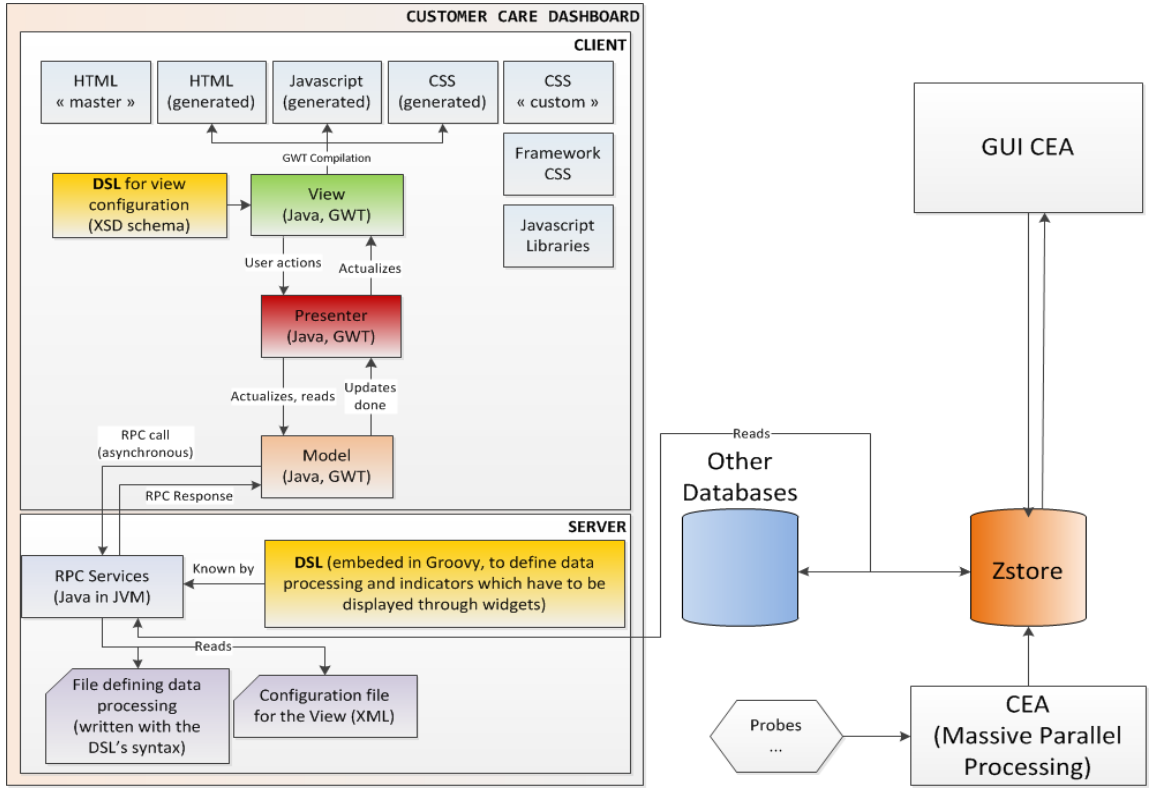
RETOUR D'EXPERIENCE
HP CCD - DSLs

- Choix du langage pour l'accès aux données : DSL Interne
  - Raisons précédentes (DSL interne vs DSL externe) + API existante

- Version alpha / proto en Lisp
  - API Java + interpréteur Lisp enrichi
  - Configuration au runtime rendue possible par le fait d'avoir un DSL interne à un langage interprété aux cotés de Java (langage compilé)

- Finalement : Groovy
  - API Java + GroovyShell
  - Beaucoup plus de flexibilité/possibilités
  - Configuration au runtime rendue possible par l'aspect scripting de Groovy

RETOUR D'EXPERIENCE
HP CCD - ARCHITECTURE



RETOUR D'EXPERIENCE
HP CCD – EXTENSIBILITE & ADAPTABILITE

- Une vue configurable (Schema XSD)

```
1  <?xml version="1.0"?>
2  <gui>
3      <row>
4          <column>
5              <widget/>
6              ...
7          </column>
8          ...
9      </row>
10 </gui>
```

- Facilité d'ajout/modification de visualisations

RETOUR D'EXPERIENCE
HP CCD – EXTENSIBILITE & ADAPTABILITE

- Un langage dédié pour définir des indicateurs
  - Lié à la partie vue par des identifiants communs
  - Configurable "à chaud"

```
1  connect "db" "a confidential database url" "user" "password"
2  read db "SELECT speedup, speeddown FROM ... WHERE ..."
3
4  streamingKpi = 100 - avg(speedup) / avg(speeddown)
5  kpi "streaming" streamingKpi
6
7  streamingInfo = streamingKpi > 75 ? "tout va bien: " : "ça va mal: "
8
9  info "streaming" streamingInfo + streamingKpi
```

3.

CONCEVOIR UN DSL INTERNE

Méthode & exemple
avec Groovy

CONCEVOIR UN DSL INTERNE
APPROCHE

- Phases de développement de *Sloane* :
  - **Décision** : Définir « pourquoi » mettre en place un langage dédié pour ce domaine est pertinent
  - **Analyse** : Analyser le domaine et les besoins spécifiques à ce langage, ainsi que les choix technologiques
  - **Conception** : Définir l'approche technique de la mise en œuvre du langage
  - **Implantation** : Programmer le langage en lui-même
  - **Déploiement** : Mettre à disposition le langage dédié ainsi conçu

CONCEVOIR UN DSL INTERNE
APPROCHE

2 approches :

- A partir de la syntaxe souhaitée

  Syntaxe → Fonctionnement

- A partir des fonctionnalités

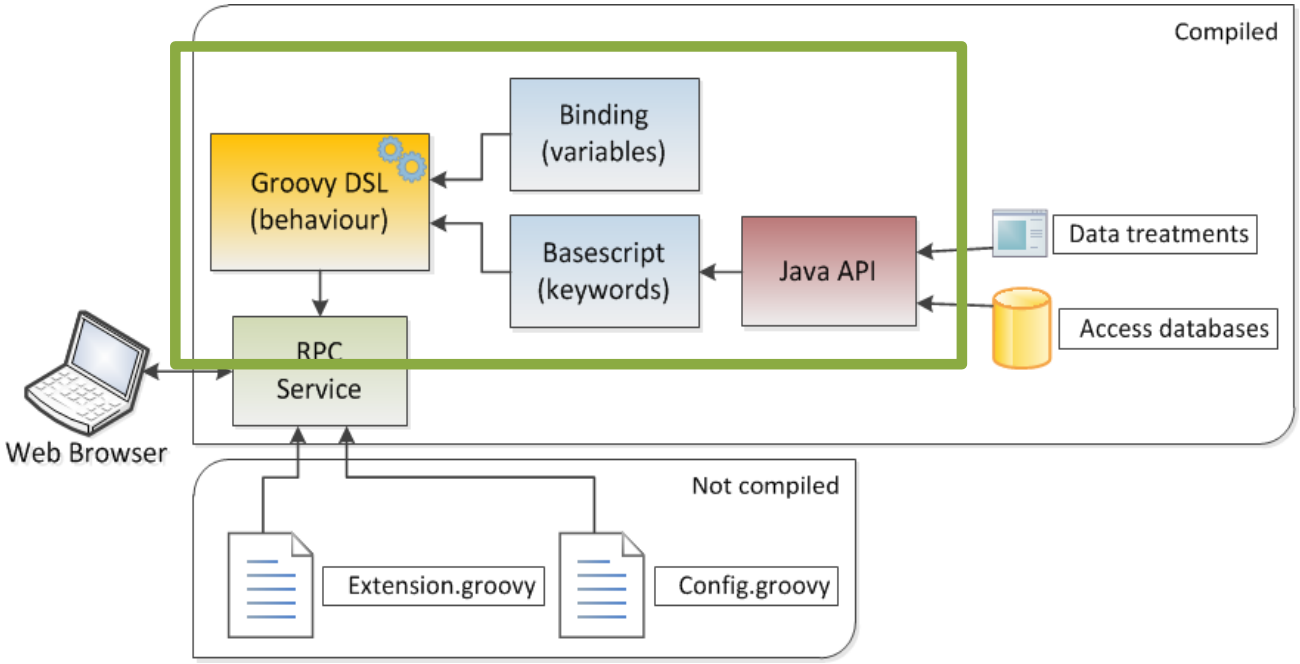
  Fonctionnement → Syntaxe

Le plus souvent en pratique :

- Création d'une API ou déjà existante
- Utilisation de cette API au sein d'un langage hôte
- Adaptation incrémentale de la syntaxe
- Boucler avec le domaine

CONCEVOIR UN DSL INTERNE
HP CCD – DSL INTERNE A GROOVY

- Architecture mise en place



GROOVY
C'EST QUOI ? (1/6)

- Inspiré de Java, Python, Ruby et Smalltalk
- Langage OO pour la JVM
- Alternative au langage Java
- Compilé ~Java
- Ou compilé à la volée/interprété ~Javascript

- Java bytecode
- Typage dynamique ou statique
- Grandes libertés syntaxiques
- Méta-programmation

(méta-classes, operator overloading, AST transformations)

GROOVY
C'EST QUOI ? (2/6)

- Script vs class


```java
public class Java {
    public static void main(String[] args) {
        System.out.println("Hello");
    }
}
```

```groovy
println "Hello"
```

- Typage optionnel / dynamique


```java
// given API method
public Rate<LoanType, Duration, BigDecimal>[] lookUpTable(){}
```


```java
Rate<LoadType, Duration, BigDecimal>[] table = lookUpTable();
```

```groovy
def table = lookUpTable()
```

GROOVY
C'EST QUOI ? (3/6)

- Déclaration de listes/map facilitée

```
// List
def days = [Monday, Tuesday, Wednesday]

// Map
def countries = [FR: 'France', DE: 'Germany']

// Range
def workDays = Monday..Friday
```

- Parenthèses et points-virgules optionnels


```
take(coffee);
take("black", coffee)
```

```
take coffee
take "black", coffee
```

GROOVY
C'EST QUOI ? (4/6)

- ExpandoMetaClass

```
Number.metaClass.getCoffee { -> delegate }
Number.metaClass.getCoffees { -> delegate }
3.coffees
```

- Command chain expressions

```
take coffee with sugar
take coffee with sugar, milk and liquor

// Equivalent to
take(coffee).with(sugar)
take(coffee).with(sugar, milk).and(liquor)
```

Code

```
def take(beverage) {
    [with: { thing1, thing2 ->
        [and: { other -> }]
    }]
}
def coffee, sugar, milk, liquor

take(coffee).with(sugar,
milk).and(liquor)
```

GROOVY
C'EST QUOI ? (5/6)

- Sécurité d'un GroovyShell : ImportCustomizer, SecureASTCustomizer

- @ThreadInterrupt
- @TimedInterrupt
- @ConditionalInterrupt

Statements can be checked using
*org.codehaus.groovy.control.customizers.SecureASTCustomizer.**StatementChecker***

Expressions can be checked using
*org.codehaus.groovy.control.customizers.SecureASTCustomizer.**ExpressionChecker***

```
secure.with {
    // disallow closure creation
    closureAllowed = false
    // disallow method definitions
    methodDefinitionAllowed = false
    // empty white lisls => forbid imports
    importsWhiteList = []
    staticImportsWhiteList = []
    staticStarImportsWhiteList = []
    // types allowed to be used (including primitive types)
    constantTypesClassesWhiteList = [
        int, Integer, Number, Integer.TYPE, String, Object
    ].asImmutable()
    // classes that are allowed to be receivers of method calls
    receiversClassesWhitelist = [
        int, Number, Integer, String, Object
    ].asImmutable()
    // language tokens allowed
    tokensWhitelist = [PLUS, MINUS, MULTIPLY].asImmutable()
    // etc...
}
```

GROOVY
C'EST QUOI ? (6/6)

- Operator overloading

```
a + b    // a.plus(b)
a - b    // a.minus(b)
a * b    // a.multiply(b)
a.divide(b)
a / b
a % b    // a.modulo(b)
a ** b   // a.power(b)
a | b    // a.or(b)
a & b    // a.and(b)
a ^ b    // a.xor(b)
a[b]     // a.getAt(b)
a << b   // a.leftShift(b)
a >> b   // a.rightShift(b)
+a       // a.unaryPlus()
-a       // a.unaryMinus(b)
~a       // a.bitwiseNegate()
a < b    // a.compareTo(b) < 0
a > b    // a.compareTo(b) > 0
```

- AST Transformations
  - Modifier l'Abstract Syntax Tree pour faire de la "compile-time" méta-programmation
- Plus complexe mais plus puissant que Java
  - metaClass > class > instance
- methodMissing est appelée si une méthode n'existe pas

```
class Foo {
    def methodMissing(String name, args) {
        return null
    }
}
```

- Et plein d'autre choses…

http://groovy-lang.org/index.html

4.

CONCEVOIR UN DSL INTERNE

Exemple avec Groovy

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- Considérons un cas simple : un lance missile USB
- API Java disponible

```
public class USBMissileLauncher {
    public void fire() {
        // some code
    }
    public void pivot(Direction d) {
        // some code
    }
}
```

```
public enum Direction {
    left, right, up, down
}
```

- Besoin d'une surcouche à cette API ? → Non car simple et proche d'un pattern commande
- Approche : API → Syntaxe

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- Ce qu'on écrirait en Java

```
1  import static com.hp.id.Direction.*;
2  import com.hp.idm.USBMissileLauncher;
3
4  public class Command {
5      public static void main(String[] args) {
6          USBMissileLauncher missileLauncher = new USBMissileLauncher();
7          missileLauncher.pivot(up);
8          missileLauncher.pivot(left);
9          missileLauncher.fire();
10     }
11 }
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- Ce qu'on écrirait en Java

```
1  import static com.hp.id.Direction.*;
2  import com.hp.idm.USBMissileLauncher;
3
4  public class Command {
5      public static void main(String[] args) {
6          USBMissileLauncher missileLauncher = new USBMissileLauncher();
7          missileLauncher.pivot(up);
8          missileLauncher.pivot(left);
9          missileLauncher.fire();
10     }
11 }
```

- Compréhensible que par un développeur
- Beaucoup de "bruit"

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- En rouge, ce qui ne sert à rien

```
1  import static com.hp.id.Direction.*;
2  import com.hp.idm.USBMissileLauncher;
3
4  public class Command {
5      public static void main(String[] args) {
6          USBMissileLauncher missileLauncher = new USBMissileLauncher();
7          missileLauncher.pivot(up);
8          missileLauncher.pivot(left);
9          missileLauncher.fire();
10     }
11 }
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- En vert, ce qui ne contribue pas à la compréhension du programme

```
1  import static com.hp.id.Direction.*;
2  import com.hp.idm.USBMissileLauncher;
3
4  public class Command {
5      public static void main(String[] args) {
6          USBMissileLauncher missileLauncher = new USBMissileLauncher();
7          missileLauncher.pivot(up);
8          missileLauncher.pivot(left);
9          missileLauncher.fire();
10     }
11 }
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- En jaune, ce qui n'est pas indispensable non plus

```
import static com.hp.id.Direction.*;
import com.hp.idm.USBMissileLauncher;

public class Command {
    public static void main(String[] args) {
        USBMissileLauncher missileLauncher = new USBMissileLauncher();
        missileLauncher.pivot(up);
        missileLauncher.pivot(left);
        missileLauncher.fire();
    }
}
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- On voudrait plutôt écrire…

```
up left fire
```

```
pivot up
pivot left
fire
```

- Ou bien à terme…

```
up 30.degree at 10.tr/min
left 42.degree at
25.radian/s
right 42.degree at 3.km/h
fire
fire
fire
```

```
pivot up by 10.degree at 10.tr/min
pivot left by 42.degree at 1.m/s
fire
fire
fire
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- Première étape : utiliser un GroovyShell dans un main Java

```
import groovy.lang.GroovyShell;
import groovy.lang.Script;
import java.io.File;

public class Command {
    public static void main(String[] args) throws Exception {
        GroovyShell shell = new GroovyShell();
        Script script = shell.parse(new File("metalGearScript.groovy"));
        script.run();
    }
}
```

- Script (équivalent au main Java)

```
import com.hpe.idm.api.USBMissileLauncher
import static com.hpe.idm.api.Direction.*

def missileLauncher = new USBMissileLauncher()
missileLauncher.pivot(left)
missileLauncher.pivot(right)
missileLauncher.fire()
```

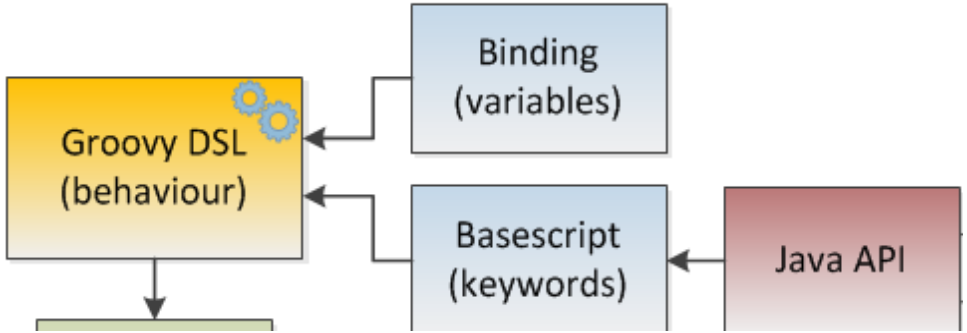
- *Remarque:* On peut par exemple n'implémenter que le cœur du DSL en Groovy, le reste en Java

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- Structure
  - **Binding** =~ Map de variables attachées au script
    → Pour partager des données entre le script et la classe appelante
  - **Basecript** =~ fonctions prédéfinies du script
    → instructions, mots clés réservés du DSL

- Premier objectif

```
pivot up
pivot left
fire
```



EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- Binding (Java or Groovy class) →

```groovy
package com.hpe.idm.dsl;

import groovy.lang.Binding;
import groovy.lang.Script;

public class metalGearBinding extends Binding {
    // The groovy script using this Binding
    private Script script;

    public metalGearBinding() {
        super();
    }

    public metalGearBinding(Script script) {
        super();
        this.script = script;
    }

    public void setScript(Script script) {
        this.script = script;
    }

    public Object getVariable(String name) {
        // TODO
        return super.getVariable(name);
    }

    public void setVariable(String name, Object value) {
        super.setVariable(name, value);
    }
}
```

- Basescript (Groovy class)

```groovy
package com.hpe.idm.dsl;

import groovy.lang.Script;

public abstract class MetalGearBasescript extends Script {
    // TODO
}
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- DSL utilisant
  - Binding
  - Basescript
  - GroovyShell

```groovy
package com.hpe.idm.dsl

import org.codehaus.groovy.control.CompilerConfiguration

class MetalGearDSL {
    private GroovyShell shell
    private CompilerConfiguration configuration
    private MetalGearBinding binding
    private MetalGearBasescript basescript

    MetalGearDSL() {
        binding = new MetalGearBinding()
        configuration = new CompilerConfiguration()
        configuration.setScriptBaseClass("com.hpe.idm.dsl.MetalGearBasescript")
        shell = new GroovyShell(configuration)
        // TODO
    }

    void eval(File scriptFile) {
        Script script = shell.parse(scriptFile)

        binding.setScript(script)
        script.setBinding(binding)

        script.run()
    }
}
```

- Nouveau *main*

```groovy
package com.hpe.idm.main;

import java.io.File;
import com.hpe.idm.dsl.MetalGearDSL;

public class Command {
    public static void main(String[] args) {
        MetalGearDSL dsl = new MetalGearDSL();
        dsl.eval(new File("metalGearScript.groovy"));
    }
}
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

```
import com.hp.idm.api.USBMissileLauncher
import static com.hp.idm.api.Direction.*

def missileLauncher = new USBMissileLauncher()
missileLauncher.pivot(left)
missileLauncher.pivot(right)
missileLauncher.fire()
```

- ?

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

```
import com.hp.idm.api.USBMissileLauncher
import static com.hp.idm.api.Direction.*

def missileLauncher = new USBMissileLauncher()
missileLauncher.pivot(left)
missileLauncher.pivot(right)
missileLauncher.fire()
```

- Injecter le missileLauncher dans le Binding (MetalGearDSL)

```
binding.setVariable("missileLauncher", new USBMissileLauncher())
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

```
import static com.hp.idm.api.Direction.*

missileLauncher.pivot(left)
missileLauncher.pivot(right)
missileLauncher.fire()
```

- ?

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

```
import static com.hp.idm.api.Direction.*

missileLauncher.pivot(left)
missileLauncher.pivot(right)
missileLauncher.fire()
```

- Injecter chaque valeur dans le binding (il existe une syntaxe Groovy plus courte)

```
binding.setVariable("left", Direction.left)
binding.setVariable("right", Direction.right)
binding.setVariable("up", Direction.up)
binding.setVariable("down", Direction.down)
```

- Ou bien utiliser un ImportCustomizer

```
def importCustomizer = new ImportCustomizer()
importCustomizer.addStaticStars Direction.class.name
configuration.addCompilationCustomizers importCustomizer
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

```
missileLauncher pivot(left)
missileLauncher pivot(right)
missileLauncher fire()
```

- ?

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

```
missileLauncher.pivot(left)
missileLauncher.pivot(right)
missileLauncher.fire()
```

- Utiliser le Basescript pour définir les fonctions *pivot* et *fire*

```groovy
def pivot(Direction d) {
    ((USBMissileLauncher) this.getBinding().getVariable("missileLauncher")).pivot(d)
}

def fire() {
    ((USBMissileLauncher) this.getBinding().getVariable("missileLauncher")).fire()
}
```

- *Remarque :* Une solution plus élégante serait de stocker le missileLauncher en tant que variable privée de la classe MetalGearBinding (car plus besoin de la variable dans le script)

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

```
pivot(left)
pivot(right)
fire()
```

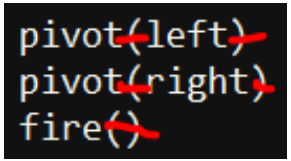
- ?

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

```
pivot(left)
pivot(right)
fire()
```

- *pivot left* et *pivot right* fonctionnent nativement
- Mais fire renvoie une **Exception**. Comment faire ?

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY



- *pivot left* et *pivot right* fonctionnent nativement
- Pour *fire*, on peut ajouter un « hook » dans la méthode getVariable du binding, et supprimer la méthode fire créée dans le Basescript précédemment

```
public Object getVariable(String name) {
    // this is useful to treat a function without any argument like a variable
    // this way, parenthesis become optional (if void function, return script)
    if("fire".equals(name)) {
        ((USBMissileLauncher) this.getVariable("missileLauncher")).fire();
        return script;
    }
    return super.getVariable(name);
}
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

```
pivot up by 10.degree at 10.tr/min
pivot left by 42.degree at 1.radian/s
fire
fire
fire
```

- Comment supporter *10.degree* et *42.tr/min* ?

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

```
pivot up by 10.degree at 10.tr/min
pivot left by 42.degree at 1.radian/s
fire
```

- Enrichir la metaClass de Integer et utiliser des classes simples d'angle, durée et vitesse

→ MetalGearDSL

- metaClass
- Closures
- Binding

```
private void setupUnits() {
    Number.metaClass {
        getS { -> new Duration(delegate, TimeUnit.second) }
        getMin { -> new Duration(delegate, TimeUnit.minute) }
        getH { -> new Duration(delegate, TimeUnit.hour) }
        getD { -> new Duration(delegate, TimeUnit.day) }
        getDegree { -> new Duration(delegate, AngleUnit.degree) }
        getTr { -> new Duration(delegate, AngleUnit.tour) }
        getRadian { -> new Duration(delegate, AngleUnit.radian) }
    }
    binding.setVariable("min",  new Duration(1, TimeUnit.minute))
    binding.setVariable("s",  new Duration(1, TimeUnit.second))
    binding.setVariable("h",  new Duration(1, TimeUnit.hour))
    binding.setVariable("d",  new Duration(1, TimeUnit.day))
}
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- class AngleUnit

```
package com.hpe.idm.unit;

public enum AngleUnit {
    degree("degree", 1.0), tour("tr", 360.0), radian("radian", 57.2957795);

    String abbreviation;
    Double multiplier;

    AngleUnit(String abbreviation, Double multiplier) {
        this.abbreviation = abbreviation;
        this.multiplier = multiplier;
    }

    @Override
    public String toString() {
        return this.abbreviation;
    }
}
```

- class TimeUnit

```
package com.hpe.idm.unit;

public enum TimeUnit {
    second("s", 1.0), minute("min", 60.0), hour("h", 3600.0), day("d", 86400.0);

    String abbreviation;
    Double multiplier;

    TimeUnit(String abbreviation, Double multiplier) {
        this.abbreviation = abbreviation;
        this.multiplier = multiplier;
    }

    @Override
    public String toString() {
        return this.abbreviation;
    }
}
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- class AngleUnit

```groovy
package com.hpe.idm.unit;

public class Angle implements Comparable<Angle> {
    Double amount;
    AngleUnit unit;

    public Angle(Double amout, AngleUnit unit) {
        this.amount = amount;
        this.unit = unit;
    }

    public Angle plus(Angle, angle) {
        AngleUnit angleUnit = this.unit.multiplier < angle.unit.multiplier ? this.unit : angle.unit;
        return new Angle((this.amount * this.unit.multiplier + angle.amount * angle.unit.multiplier) / angleUnit.multiplier, angleUnit);
    }

    public Angle minus(Angle, angle) {
        AngleUnit angleUnit = this.unit.multiplier < angle.unit.multiplier ? this.unit : angle.unit;
        return new Angle((this.amount * this.unit.multiplier - angle.amount * angle.unit.multiplier) / angleUnit.multiplier, angleUnit);
    }

    public RotationSpeed div(Duration duration) {
        return new RotationSpeed(this, duration);
    }

    @Override
    public String toString() {
        return this.amount.toString() + this.unit.toString();
    }

    @Override
    public int compareTo(Angle o) {
        return new Double(this.amount * this.unit.multiplier).compareTo(o.amount * o.multiplier);
    }
}
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- class Duration

```groovy
package com.hpe.idm.unit;

import groovy.transform.TupleConstructor;

@TupleConstructor
public class Duration implements Comparable<Duration> {
    Double amount;
    TimeUnit unit;

    public Duration(Double amount, TimeUnit unit) {
        this.amount = amount;
        this.unit = unit;
    }

    public Duration plus(Duration duration) {
        TimeUnit timeUnit = this.unit.multiplier < duration.unit.multiplier ? this.unit : duration.unit;
        return new Duration((this.amount * this.unit.multiplier + duration.amount * duration.unit.multiplier) / timeUnit.multiplier, timeUnit);
    }

    public Duration minus(Duration duration) {
        TimeUnit timeUnit = this.unit.multiplier < duration.unit.multiplier ? this.unit : duration.unit;
        return new Duration((this.amount * this.unit.multiplier - duration.amount * duration.unit.multiplier) / timeUnit.multiplier, timeUnit);
    }

    @Override
    public String toString() {
        return this.amount.toString() + this.unit.toString();
    }

    @Override
    public int compareTo(Duration o) {
        return new Double(this.amount * this.unit.multiplier).compareTo(o.amount * o.unit.multiplier);
    }
}
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- class RotationSpeed

```
package com.hpe.idm.unit;

import groovy.transform.TupleConstructor;

@TupleConstructor
public class RotationSpeed {
    private Angle angle;
    private Duration duration;

    public RotationSpeed(Angle angle, Duration, duration) {
        this.angle = angle;
        this.duration = duration;
    }

    @Override
    public String toString() {
        return this.angle.amount.toString() + this.angle.unit.toString() + "/" + this.unit.toString();
    }
}
```

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- On peut maintenant écrire

```
10.degree
42.tr/min
```

```
pivot up by 10.degree at 10.tr/min
pivot left by 42.degree at 1.radian/s
fire
```

- Et après, comment arriver à supporter la syntaxe ci-dessus ?

EXEMPLE
CRÉER UN DSL INTERNE AVEC GROOVY

- On peut maintenant écrire

```
10.degree
42.tr/min
```

```
pivot up by 10.degree at 10.tr/min
pivot left by 42.degree at 1.radian/s
fire
```

- Equivaut à

```
pivot(up).by(10.degree).at(10.tr/min)
pivot(left).by(42.degree).at(1.radian/s)
fire
```

→ Utiliser le « method-chaining »

- Remplacer la méthode pivot dans le Basescript

```
def pivot(direction) {
    [by: { distance ->
        [at: { speed ->
            // some code
        }]
    }]
}
```

EXEMPLE
CONCLUSION

- On y est ! Le lance missile USB est facilement pilotable !
- Cette approche d'élaboration d'un DSL interne avec Groovy serait la même avec un autre langage hôte (ex: Scala) ou un modèle different, plus complexe.
- Approche incrémentale, en partant des fonctionnalités, et en gardant la syntaxe cible en tête

DES QUESTIONS ?

5.

ARDUINOML AVEC GROOVY

Application

SUPRALOG
CONSEIL ▲ TECHNOLOGIE ▲ PROGICIEL

ARDUINOML AVEC GROOVY

- Appliquons tout ça à ArduinoML
- … Elaborons GroovuinoML

```
sensor "button" pin 9
actuator "led" pin 12

state "on" means "led" becomes "high"
state "off" means "led" becomes "low"

initial "off"

from "on" to "off" when "button" becomes "high"
from "off" to "on" when "button" becomes "high"

export "Switch!"
```

```
// Wiring code generated from an ArduinoML model
// Application name: Switch!

void setup(){
    pinMode(9, INPUT); // button [Sensor]
    pinMode(12, OUTPUT); // led [Actuator]
}

Long time = 0; Long debounce = 200;

void state_on() {
    digitalWrite(12, HIGH);
    boolean guard = millis() - time > debounce;
    if( digitalRead(9) == HIGH && guard ) {
        time = millis();
        state_off();
    } else {
        state_on();
    }
}

void state_off() {
    digitalWrite(12, LOW);
    boolean guard = millis() - time > debounce;
    if( digitalRead(9) == HIGH && guard ) {
        time = millis();
        state_on();
    } else {
        state_off();
    }
}

void loop() {
    state_off();
}
```

ARDUINOML AVEC GROOVY

- https://github.com/mosser/ArduinoML-kernel/tree/master/embeddeds/groovy

BONUS
SUBTILITES AVANCEES DE GROOVY

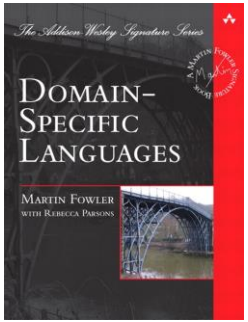
- 2 aspects du DSL Groovy de HP Customer Care Dashboard :
  - Fichier(s) de scripting limité/sécurisé/restreint
  - Fichier(s) d'extension dynamique, ouvert(s), maximisant l'adaptabilité du DSL

  → Exploitation des capacités de Groovy (Méta-programmation, langage fonctionnel, closures, …)
- Pour permettre l'extension dynamique du DSL, le Basescript ne contient plus toutes les fonctions
  - Fonctions complexes et extensions ajoutées au Binding, sous forme de closures
  - Sinon : de nombreux problèmes lors d'accès concurrents (fuites mémoires car certaines modifications de metaClass ne peuvent être garbage collectées, …)

- Tout comportement de Groovy ou de Java, visible dans le scope de notre DSL peut être modifié, à la volée !

BIBLIOGRAPHIE & LIENS UTILES

- Domain Specific Languages (Martin Fowler)
- Going to Mars with Groovy (Guillaume Laforge)



- Bons articles pour commencer avec Groovy
  - Installation

    http://www.vogella.com/tutorials/Groovy/article.html#install_springgroovytools
  - Documentation officielle pour l'élaboration d'un DSL

    http://docs.groovy-lang.org/docs/latest/html/documentation/core-domain-specific-languages.html
  - Méta-programmation

    http://fr.slideshare.net/zenMonkey/metaprogramming-techniques-in-groovy-and-grails

THANKS!

Any questions?

You can find me at @grichkanoff &
tmoreau@supralog.com