Présentation rapide du langage Smalltalk

Christophe Dony

1 Généralités

1.1 Création et Infos

Créé au Xerox Parc, définitions: 1972, 1976, 1980

- ParcPlace-Digitalk - VisualWorks, VisualSmalltalk Enterprise
- IBM - IBM VisualAge for Smalltalk
- Cincom - ObjectStudio (ex product of VMARK)
- QKS - SmalltalkAgents
- Object Connect - Smalltalk MT
- Intuitive Systems Limited - Dolphin Smalltalk
- RoboWiz Corporation - RoboWiz Shareware Smalltalk for Windows95
- Apple Research Laboratories Puis Dysney - Squeak (www.squeak.org)
- GemStone Systems - GemStone: Products

1.2 Références

A. Goldberg, D. Robson : SMALLTALK 80, the language and its implementation. Addison Wesley 1983.
Simon Lewis. The Art and Science of Smalltalk, Prentice Hall / Hewlett-Packard Professional Books,
Tutorial online:
http://zeus.enst-bretagne.fr/Tutoriaux/smalltalk/s80part1/s80part1.html
Infos diverses:
http://www.esug.org/
http://www.panasoft.com/stlinks/
Syntaxe :
http://www.chimu.com/publications/JavaSmalltalkSyntax.html
http://www.csci.csusb.edu/dick/samples/smalltalk.syntax.html

1.3 Généralités

- Smalltalk est un **langage à objets** basé sur un petit nombre de concepts.
- C'est un environnement de programmation interactive et multimedia
- C'est un système complet: compilateur, Metteur au point, éditeur de programmes, détection d'erre
  gestion des références croisées, ...
- C'est un environnement intégré de prototypage.

1.4 Caractéristiques générales du langage

- Langage à objets influencé par Simula (abstraction de données, surcharge des noms d'opérations d'inclusion)
- influencé par Lisp
  - typage dynamique,
  - types "référence" généralisés,
    Passage de paramètres par valeur (on passe donc les adresses par valeur).
  - gestion automatique de la mémoire, allocation dynamique et ramasse-miette,
  - style applicatif, toute instruction est une expression, y compris les structures de contrôle.
  - Langage compilé en instruction d'une machine virtuelle dédiée.

1.5 Interprétation, Compilation, machine virtuelle

Principe d'exécution: compilation des instruction en byteCodes ou instructions d'une machine virtuelle déc mation par objets puis interprétation de ces instructions.

Machine virtuelle Smalltalk: ensemble d'instructions dédiées à la programmation par objets.

1.6 Exemple de méthode et de bytecode généré

```
fact
    ^self = 0
        ifTrue: [1]
        ifFalse: [self * (self - 1) fact]

normal CompiledMethod numArgs=0 numTemps=0 frameSize=12
literals: (#fact )
1 <44> push self
2 <49> push 0
3 <A6> send =
4 <EC 07> jump true 13
6 <44> push self
7 <44> push self
8 <4A> push 1
9 <A1> send -
10 <70> send fact
11 <A8> send *
12 <66> pop
13 <60> push self; return
```

2 Elements de syntaxe

Voici les éléments de syntaxe de base permettant d'écrire des programmes en Smalltalk.

2.1 constantes littérales

Constantes car leur valeur ne peut être modifiée et littérale parce qu'on les manipule directement en les référenca nécessaire de les définir.

- nombres 3,3.45, -3,
- charactères $a, $M, $$
- chaines 'abc', 'the smalltalk system'
- symboles #bill, #a22
- tableaux de constantes littérales.
  #(1 2 3)
  #('vincent' 'francois' 'paul')

2.2 envois de message

On distingue les messages

- unaires, qui possèdent un seul argument : le receveur
  `1 class`
  `5 fact`
- binaires
  1+ 2*
- les "keywords"
  `1 log: 10`
  `anArray at: 2 put: 3`
- Précédence:
  unaire > binaire > keyword,
  exemple: `1  5 fact`+ égale 121 et pas 720.
- Associativité A précédence égale les messages sont composés de gauche à droite.
  exemple:
  `2 + 3 * 5 = 25`.
  `2 + (3 * 5) = 17`
- Cascade de messages
  r s1; s2; s3. est équivalent à: r s1. r s2. r s3.

2.3 instructions spéciales

```
i := 3 :: affectation
^33 :: return(33)
```

2.4 Instructions

Les instructions sont séparées par un point.

i := 1. j := 2.

2.5 Les blocks

Les *block* sont des fermetures lexicales.

Définition: Une fermeture est la liaison d'un corps de fonction (ou lambda list ou bloc Smalltalk) à un envi
ensemble de couples "identificateur - valeur"

Les *blocks* permettent entre autre d'écrire des structures de contrôle. Comparer aux lambda expressions lisp.

```
(number \\ 2) = 0 ifTrue: [parity := 0] ifFalse: [parity := 1]
```

un block, comme toute fonction, peut avoir des paramètres.

```
1 to: 20 do: [:i | Transcript show: i printSting].
```

2.6 Classes et méthodes

Les classes et les méthodes sont définies par envoi de messages.

L'environnement de développement permet d'ignorer leurs noms.

3 Les grands principes

- Toute entité est un objet.

Un objet est une entité individuelle, repérée par une adresse unique, formé de plusieurs champs, connus par leur
fixé) et contenant une valeur qui peut varier au cours du temps.

- tout objet appartient à (est instance) d'une classe qui définit sa structure et ses comportements.

- un comportement d'un objet est activé par un envoi de message.

Plus généralement, **tout calcul** en Smalltalk s'effectue par l'envoi d'un message à un objet.

Figure 1: La classe Date

4 Programmation basique

4.1 Création d'une classe

Toute classe est créée comme sous-classe d'une autre classe.

Variables d'instance, variables de classes (statiques), variables de *pool*.

4.2 Méthodes d'instance

Autre exemple : étudier les méthodes de la classe `Point`.

4.3 Envoi de message

La résolution de l'envoi de message est dynamique.

Recherche de la méthode dans la classe du receveur puis dans ses superclasses.

Diverses stratégies d'optimisation sont implantées dans les divers environnements dont les techniques de cache.

4.4 Méthodes de classe

Méthodes applicables aux classes et invocable par envoi de message aux classes.

Ex : Date new. Date today.

Comparables aux méthodes de classe de Java mais sont intégrées de manière cohérente.

Reposent sur l'existance de métaclasses.

Toute classe C est instance d'une métaclasse, générée automatiquement au moment de la création de C et nom

Une méthode de classe est une méthode d'instance de la métaclasse.

4.5 Différentes sortes de variables accessibles au sein des méthodes

Les variables accessibles au sein d'une méthode M:

- les paramètres de M,
- les variables temporaires de M, (exemple, réécrire théta en utilisant des variables temporaires)
- les variables d'instance de l'objet receveur O,
- les pseudo-variables self, super, true, false, nil.
- les variables de classe de la classe C de O (celle ou est définie M)

Figure 2: Méthode < de la classe Date

5 Instantiation

- On instancie une classe en lui envoyant le message `new`.
- `new` est une méthode définie sur une superclasse commune à toutes les métaclasses (cf. métaclasses). `new` réalise l'allocation mémoire et rend un nouvel objet.
- Exemple: `Date new`

5.1 Initialisation des objets

Il n'y a pas de constructeurs.

L'initialisation s'effectue via des méthodes appelées dans des méthodes de classe dédiés ou des redéfinitions de la `new`.

Exemples : les méthodes `new` et `x:y:` de la classe Point.

5.2 Classes abstraites

Une classe ne peut être déclarée abstraite mais elle être rendue abstraite en redéfinissant la méthode new pour sign

Ceci pose néanmoins de graves problèmes pour les futures sous-classes concrètes. (utilisation obligatoire de `bas`

5.3 Methode abstraite

Une méthode ne peut être déclarée abstraite mais elle être rendue abstraite en la définissant de la façon suivante:

```
method
    self subclassResponsibility
```

5.4 Sous-types et Héritage

Mêmes principes que dans les autres langages à objet.

Pas d'héritage multiple.

5.5 Schémas de spécialisation

Figure 3: Une méthode de classe.

6 Un exemple de bibliothèque

La bibliothèque des Magnitude et des Numbers.

```
Magnitude ()
    ArithmeticValue ()
        Number ()
            FixedPoint ('numerator' 'denominator' 'scale')
            Fraction ('numerator' 'denominator')
            Integer ()
                LargeInteger ()
                    LargeNegativeInteger ()
                    LargePositiveInteger ()
                SmallInteger ()
            LimitedPrecisionReal ()
                Double ()
                Float ()
        Point ('x' 'y')
    Character ()
    Date ('day' 'year')
    MethodDefinition ('inheritingClass' 'implementingClass' 'selector' 'extraText')
    Time ('hours' 'minutes' 'seconds')
```

7 La méta-programmation en Smalltalk

Réflexivité : Capacité qu'a un système à donner à ses utilisateurs une représentation de lui-même en connexi
représentation effective en machine.

Entité de première classe : entité possédant une représentation dans le langage et que l'on peut lire, et évent

En Smalltalk, les classes, les méthodes compilées, méthodes, certaines structures de contrôle, éventuellement
l'environnement de programmation, le compilateur ... sont des entités de première classe.

7.1 Le cas des objets primitifs (rock-bottom objects)

Il existe une classe décrivant chacun des types primitifs : String, Float, SmallInteger, ...

Il est possible de modifier les méthodes de ces classes ou d'en créer de nouvelles (exemple : la méthode fact su

```
Number ()
    FixedPoint ('numerator' 'denominator' 'scale')
```

```
        LargeInteger ()
            LargeNegativeInteger ()
            LargePositiveInteger ()
        SmallInteger ()
    LimitedPrecisionReal ()
        Double ()
        Float ()
```

Les objets primitifs sont utilisables comme les autres objets du système (différence avec Java), ils sont représe
mais leur implantation n'est pas entièrement définie par cette classe.

7.2 Les structures de contrôle

Utilisation de la classe booléen et des fermetures lexicales.

7.3 Les méta-classes

7.3.1 Rappel : la solution Objvlisp

- `Object est la racine de l'arbre d'héritage`,
- `Class` est instance d'elle-même, sous-classe de `Object` et racine de l'arbre d'instantiation.
-

7.3.2 La solution Smalltalk

La solution Smalltalk est plus complexe et à un pouvoir d'expression inférieur mais elle ne pose pas de problèmes
métaclasses.

- La relation instance-de entre les classes et les métaclasses est une bijection.
- `ObjectMetaclass` est une sous-classe de `Class`.

```
Object ()
    Behavior ('superclass' 'methodDict' 'format' 'subclasses')

        ClassDescription ('instanceVariables' 'organization')
            Class ('name' 'classPool' 'sharedPools')
                ... all the metaclasses ...
            Metaclass ('thisClass')
```

7.3.3 Le problème de la compatibilité des métaclasses

Compatibilité Ascendante :

Classe A, methode foo : 'self class bar'.

CLasse B, sous-classe de A, 'b foo' peut provoquer une erreur.

Compatibilité Descendante :

Métaclasse MA, méthode bar : 'self new foo'.

Métaclasse MB, sous classe de MA, 'mb bar' peut provoquer une erreur.

7.4 La pile d'exécution

```
!Symbol methodsFor: 'catch-throw'!

catch: aBlock
    "le nom de la fonction et le receveur font office de marque dans l
a pile"
    aBlock value.!

throw: aValue
        "Looks for a catch, the mark of which is self,
        if found, transfer control while executing recovery blocks."
```

Figure 4: Inspection d'une classe.

```
        currentContext _ thisContext.
        catchMethod _ Symbol compiledMethodAt: #catch:.
        [currentContext method == catchMethod and: [currentContext rec
eiver == self]]
                whileFalse: [currentContext _ currentContext sender].
        thisContext sender: currentContext sender.
        ^aValue! !
```